A Dissertation

titled

A Novel Algorithm for the Reconstruction of an Entrance Beam Fluence from Treatment Exit Patient Portal Dosimetry Images

by

Nicholas Niven Sperling

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the Doctor of Philosophy Degree in Physics

Dr. E. Ishmael Parsai, Committee Chair

Dr. Patricia R. Komuniecki, Dean
College of Graduate Studies

The University of Toledo
December 2013
An Abstract of

A Novel Algorithm for the Reconstruction of an Entrance Beam Fluence from Treatment Exit Patient Portal Dosimetry Images

by

Nicholas N. Sperling

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the Doctor of Philosophy Degree in Physics

The problem of determining the \textit{in vivo} dosimetry for patients undergoing radiation treatment has been an area of interest since the development of the field. Most methods which have found clinical acceptance work by use of a proxy dosimeter, e.g.: glass rods, using radioluminescence; thermoluminescent dosimeters (TLD), typically CaF or LiF; Metal Oxide Silicon Field Effect Transistor (MOSFET) dosimeters, using threshold voltage shift; Optically Stimulated Luminescent Dosimeters (OSLD), composed of Carbon doped Aluminum Dioxide crystals; RadioChromic film, using leuko-dye polymers; Silicon Diode dosimeters, typically p-type; and ion chambers. More recent methods employ Electronic Portal Image Devices (EPID), or dosimeter arrays, for entrance or exit beam fluence determination.

The difficulty with the proxy \textit{in vivo} dosimetry methods is the requirement that they be placed at the particular location where the dose is to be determined. This precludes measurements across the entire patient volume. These methods are best suited where the dose at a particular location is required.

The more recent methods of \textit{in vivo} dosimetry make use of detector arrays and reconstruction techniques to determine dose throughout the patient volume. One method uses an array of ion chambers located upstream of the patient. This requires a special
hardware device and places an additional attenuator in the beam path, which may not be desirable.

A final approach is to use the existing EPID, which is part of most modern linear accelerators, to image the patient using the treatment beam. Methods exist to deconvolve the detector function of the EPID using a series of weighted exponentials (1). Additionally, this method has been extended to determine in vivo dosimetry.

The method developed here employs the use of EPID images and an iterative deconvolution algorithm to reconstruct the impinging primary beam fluence on the patient. This primary fluence may then be employed to determine dose through the entire patient volume. The method requires patient specific information, including a CT for deconvolution/dose reconstruction. With the large-scale adoption of Cone Beam CT (CBCT) systems on modern linear accelerators, a treatment time CT is readily available for use in this deconvolution and in dose representation.
# Table of Contents

Abstract ........................................................................................................................................ iii

Table of Contents ............................................................................................................................. v

List of Tables ..................................................................................................................................... x

List of Figures ................................................................................................................................. xi

List of Equations ............................................................................................................................. xiii

Preface ........................................................................................................................................... xiv

1 Radiation Therapy ....................................................................................................................... 1
  1.1 Modern Linear Accelerator (Linac) ...................................................................................... 1
    1.1.1 MultiLeaf Collimator (MLC) ....................................................................................... 4
  1.2 Intensity Modulated RadioTherapy (IMRT) ..................................................................... 6
    1.2.1 IMRT Quality Assurance (QA) .................................................................................. 8

2 Monte Carlo ................................................................................................................................ 12
  2.1 Monte Carlo codes .............................................................................................................. 14
    2.1.1 MCNP5 ......................................................................................................................... 14
    2.1.2 BEAMnrc ...................................................................................................................... 15
  2.2 Variance Reduction Techniques ......................................................................................... 18
2.2.1 MCNP5 ........................................................................................................... 19
2.2.2 BEAMnrc ........................................................................................................ 19

3 Cluster Design ........................................................................................................ 25
  3.1 Parallelization considerations ......................................................................... 26
  3.2 The Two Clusters ............................................................................................. 28
  3.2.1 Torque Cluster ............................................................................................ 28
  3.2.2 Blade Cluster ............................................................................................... 30
  3.3 TORQUE Resource Manager .......................................................................... 33
  3.4 Custom Code Modifications ............................................................................ 35

4 Accelerator Model Creation .................................................................................. 36
  4.1 Component Module Sequence ......................................................................... 37
  4.2 Simulation Input Parameters ........................................................................... 38
    4.2.1 Accelerator Head Model ............................................................................ 40
    4.2.2 Cylindrical Phantom ................................................................................ 56
    4.2.3 Air Slab .................................................................................................... 57
  4.3 Phase space file format ...................................................................................... 57

5 Virtual Electronic Portal Image Device (vEPID) .................................................. 59
  5.1 vEPID Detector Deconvolution ....................................................................... 61
    5.1.1 Deconvolution Parameter Fitting ............................................................... 62

6 Parameter Space construction ................................................................................. 67
7 Fluence Calculation ........................................................................................................ 73
8 Fluence Solver ................................................................................................................ 75
  8.1 Derivative calculation function .............................................................................. 76
  8.2 Initial Guess Calculation ....................................................................................... 77
  8.3 Fluence Solver Program Design .............................................................................. 78
9 Results .......................................................................................................................... 81
10 Conclusion .................................................................................................................. 86
References ...................................................................................................................... 88
Appendix A Live-Build Customizations .................................................................... 94
  A.1 auto/build ............................................................................................................ 94
  A.2 auto/config .......................................................................................................... 94
  A.3 auto/clean ............................................................................................................. 94
  A.4 auto/chroot_local-preseed/nis.cfg ................................................................... 95
  A.5 auto/chroot_local-packagelists/blade_live.lst ................................................. 95
  A.6 auto/chroot_local-includes/etc/ganglia/conf.d/hpasmcli.pyconf ................... 95
  A.7 auto/chroot_local-includes/etc/ganglia/conf.d/modpython.conf ....................... 96
  A.8 auto/chroot_local-includes/etc/ganglia/gmond.conf ........................................ 96
  A.9 auto/chroot_local-includes/etc/init.d/nfsswap .................................................. 100
  A.10 auto/chroot_local-includes/lib/live/config/001-hostname ............................... 101
  A.11 auto/chroot_local-includes/usr/lib/ganglia/python_modules/hpasmcli.py ....... 102
List of Tables

3.1: Component wise comparison of clusters. ................................................................. 30
3.2: Resources allocated by node identifier ................................................................. 35
4.1: Accelerator Head Model Module Components and Description ......................... 38
4.2: Phantom Model Component Modules and Description .......................................... 38
5.1: Comparison of number of particles in phase space source and average relative error in dose calculation by field size in vEPID simulation. ............................. 61
5.2: Calculated Parameters from Deconvolution Parameter Solver. ............................ 64
List of Figures

4-1: Comparison of simulated spectral distribution to 6MV spectra published in Mohan, et al. ........................................................................................................................................... 40
4-2: Representation of primary collimator......................................................................................... 43
4-3: FLATFILT CM as used in the simulations. The materials from center out are: Lead, air, and Tungsten. ........................................................................................................................................ 44
4-4: The radially symmetric monitor chamber component module................................................. 46
4-5: Mylar mirror component module, angled at 55 degrees to the z-axis.................................... 47
4-6: Secondary collimators shown in XZ view.................................................................................. 49
4-7: Secondary collimators shown in YZ view.................................................................................. 50
4-8: MLC CM shown in the XZ plane at the Y axis................................................................. 52
4-9: MLC CM shown in the XY plane at the Z=51 cm SSD..................................................... 53
4-10: MLC CM shown in the YZ plane at the X axis.................................................................. 54
4-11: Air gap and PMMA window marking the end of the accelerator head............................... 55
5-1: Histogram of percent difference values for pixels where fluence is greater than 2% of maximum. ........................................................................................................................................ 65
5-2: Colormap image of percent difference values for pixels where fluence is greater than 2% of maximum.................................................................................................................................. 66
6-1: Histogram of density of parameter space data for a resample dataset of 32x32 arrays. ........................................................................................................................................... 70
6-2: Histogram of array density for 128x128 grid parameter space. .............................. 71
9-1: Histogram for Smile ................................................................................................. 83
9-2: Histogram for Questionmark .................................................................................. 83
9-3: Visual comparison of entrance fluence for the first IMRT field. (Left: planned fluence, Right: computed fluence) ................................................................. 84
9-4: Visual comparison of entrance fluence for the second IMRT field. (Left: planned fluence, Right: computed fluence) ................................................................. 84
List of Equations

6-1: Inequality describing the point at which a CSR stored matrix requires less space than a square dense matrix............................................................................................................. 69

6-2: Definition of density, and restatement of 6-1 in terms of density............................ 70

7-1: Exit fluence is calculated from the parameter space weights.................................... 74

8-1: Residual used in the calculation of match quality....................................................... 75

8-2: Prototype function used in derivative computation................................................. 77

8-3: Derivation of appropriateness of initial guess normalization................................. 78
Preface

The algorithm we have created involves an iterative approach to deconvolving the scatter component of the image at the EPID from the attenuated primary fluence at the level of the EPID. The EPID is designed with the intent of reducing contributions to the image from patient scatter, as these components reduce image quality. This design consideration aids in the removal of the remaining component of patient scatter. Once the scatter component of the image has been removed the remaining component is assumed to be primary fluence attenuated by the patient, which may be traced back through the patient volume, and amplified by the effective depth of the traversed path. This final result, the primary fluence at entrance, may be used to determine the dose in the patient volume via several different dose calculation algorithms as employed in treatment planning systems (TPS).

In this study, we intend to demonstrate the feasibility of this method by the creation of a virtual accelerator head/patient/EPID system which will produce both entrance fluence and exit EPID images. This approach will require the creation of a program to deconvolve the detector function of the virtual EPID (vEPID) from the dose array produced. Additionally, a method for computing and removing the scatter dose from the generated exit fluence will be devised using the patient component of the system as the primary scattering medium.
The accelerator head/patient/EPID system will be created in the BEAMnrc Monte Carlo code, an extension of the Electron Gamma Shower code produced by the National Research Council of Canada (EGSnrc). This code was created to simulate “the coupled electron-photon transport” (2) in materials of an arbitrary geometry. The accelerator head design was based on the head design of the Varian Trilogy series linear accelerator, with a millennium MLC.
Chapter 1

Radiation Therapy

The field of radiation therapy developed shortly after the discovery by Roentgen of X-rays. It has progressed from simplistic low energy linear accelerators and Van De Graaff generators to modern advanced high-energy linear accelerators for external beam treatments.

1.1 Modern Linear Accelerator (Linac)

The most common method employed today for the generation of high energy x-rays for use in radiation therapy is the linear accelerator (3), named in contrast to the methods of generating high energy particles through the acceleration through a cyclic process (e.g. betatron, cyclotron, synchrotron, etc.). Significant advantages exist in using a linear acceleration column over a cyclic approach, as the charged particles (in this case electrons) are not subject to bremsstrahlung losses during bending, and through advances in acceleration cavity design, fairly high energies may be accomplished in a short distance.

The components of the modern linear accelerator can be considered in three parts: the acceleration components, the collimation components (head), and the patient.
alignment components. The first segment is responsible for the bunching and acceleration of groups of electrons to MeV energies within a vacuum. After exiting the vacuum system, the electrons are typically not traveling in the direction of the patient and must be bent toward the patient. Two methods are in common use for accomplishing this without producing significant chromatic dispersion of the beam (3) by bending through 270° or 112.5°.

After bending, the electron beam enters the ‘head’ section of the accelerator. It is in this section where the finely focused electron beam is transformed into a clinically useful beam. In this research we focus on this segment of the accelerator as it has the most important and complicated role in the shaping of the treatment beam for patient delivery. In photon mode, the electron beam is made to impinge on a ‘target’ composed of high Z materials, typically Tungsten (Z=74) and Tantalum (Z=73) with the intent of converting the kinetic energy stored in the electron beam into bremsstrahlung photons. At the beam energies used in clinical treatment – between 4MV to 25MV – the conversion of electron kinetic energy to photon energy is between 10% - 30% (4). The photons are generated in a very forward peaked but relatively uniform spread, which may be considered a uniform radiator for simplification in our simulations (5).

The beam then passes through a series of collimators whose function is to attenuate the beam outside of the region intended to be delivered to the patient. These collimators are constructed of high Z materials to attenuate the high energy photons in minimal space, though this results in large contributions to the scatter radiation from these components. The first collimator is a thick plate with a conic section removed to define the largest radiation aperture the machine can treat. After passing through the
primary collimator, the beam retains a highly forward peaked angular distribution which is not desirable for uniform dose delivery to the patient. To correct this, the beam passes through the flattening filter: a high Z, cylindrically symmetric beam modulating device which is designed to produce a uniform dose profile at depth under treatment conditions. Each photon energy in the machine requires a different level of flattening to produce a flat profile, so the filter is mounted on a carousel to allow for simple selection for an energy.

Subsequent elements in the beam path inside head include a monitor chamber to detect in real time the beam parameters, typically consisting of a pair of thin transmission ion-chambers which are used to determine the amount of radiation that is being delivered (6). A pair of independent ion chambers are used to provide a redundant measure of the radiation being delivered since this is the proxy measure used to control the total amount of radiation delivered to the patient. The monitor chamber output is required to be calibrated using equipment which has a calibration traceable to the National Institute of Standards and Technology calibration laboratories. The procedure involves calibrating a unit measure from the monitor chamber – the Monitor Unit (MU) – to a reading from an ionization chamber in water (7).

Another component in the beam path not used for collimation is a thin aluminized Mylar mirror designed to provide a visible light verification of the field to be delivered. Finally, the secondary collimator – often termed X and Y jaws – and the Multileaf Collimator (MLC) (if fitted) are used to define the final treatment aperture. Prior to the advent of the MLC, if non-rectangular field blocking was required, a final field defining aperture device would be placed at the bottom of the treatment head. These blocks would
be composed of an eutectic alloy of Bismuth, Lead, Tin, and Cadmium often known as “Wood’s metal,” which is desirable for its low melting point, high effective Z, and low cost.

The final section of the linac, the patient positioning section, consists of the treatment couch and the movable gantry. The linear accelerator system is mounted on a rotating gantry with a fixed spatial center of rotation, termed the isocenter as it is the ‘same center’ for all axes of rotation. The patient is located on a treatment couch which typically has the ability to move in 4 dimensions: up/down, into gantry/out of gantry, left/right, and yaw rotation about the isocenter point.

In addition to the devices present in the radiation field for the delivery of the dose, there exist several ancillary devices to assist in positioning the patient for treatment. The most common of these devices are Electronic Portal Image Devices (EPID), and OnBoard Imaging (OBI) devices. The EPID consists of a semiconductor imaging panel designed to measure the radiation fluence downstream of the patient. The OBI system consists of a kV x-ray source and kV EPID panel, mounted such that the rotation axis corresponds with the rotation axis of the accelerator head. Due to the primary mode of interaction of keV photons being the photoelectric effect, and the primary mode of interaction of MeV photons being Compton effects, the kV imaging system provides significantly better delineation of bony anatomy from soft tissue for localization.

1.1.1 MultiLeaf Collimator (MLC)

With the advent of the MLC, custom blocking for individual treatment was made significantly simpler, allowing for a unique field to be defined and changed via computer control. The MLC consists of a series of thin (typically between 0.25 – 0.5 cm wide)
tungsten ‘leaves’ oriented vertically in the beam path. The leaves are of sufficient height in the beam path to produce around 97% attenuation at the level of the patient (8), with several tricks being employed to minimize transmission in the space between leaves, namely adding a tongue and mating groove on adjacent leaves. The leaves are also typically configured to be thinner at the end closer to the source and angled inward to account for the divergent nature of the photon beam. This helps reduce the radiation penumbra at the field edge in the direction perpendicular to the leaves motion.

There are two approaches to handling the field edge effects of the leaves parallel to their direction of motion, called double- and single-focused respectively: the first is to use flat leaf ends and retract/extend the leaves in a manner which maintains the appropriate divergence based on the position; the second method allows the motion of the leaves to be linear and relies on a curved leaf end design. The second approach allows for a simpler mechanical control system at a cost of enhanced transmission at the edges of the field.

An additional consideration when using leaves with a rounded end is the effect of divergence on the positioning of the leaves. The position of the leaf tip does not correspond linearly with the position of the 50% transmission through the leaf edge, which is the location typically defined as the block edge. For this reason, a correction to the physical position of the leaf is applied based on the desired radiation field size at the level of the isocenter (9). This correction is applied transparently by the accelerator based on parameters determined by the manufacturer; however, it is critical that the corrections used be known in order to accurately model the MLC.
1.2 Intensity Modulated RadioTherapy (IMRT)

As computer systems have advanced in recent years, and with the advent of modern treatment planning systems and diagnostic imaging systems, radiotherapy treatment has been able to more selectively identify and quantify dose to regions of interest (ROI). With these advances in identification of target tumor volumes and the ability to delineate potentially normal and functional tissue from target structures, the natural response is to focus more intently on those tissues known to be diseased, while attempting to spare those that do not express tumor indicators. With the technology available in external beam radiotherapy prior to the advent of IMRT, any manipulation of the dose delivery in an attempt to achieve a greater conformality with the target would rely on the increase of the number of treatment beams or the use of so-called ‘tissue compensating devices,’ e.g. wedges, bolus, etc. The goal of a tissue compensating device is to modify the typically ‘flat’ beam profile into a profile which varies significantly with position in order to compensate for changes in density, or tissue thickness on a per patient basis.

The concept of tissue compensation is extended to an extreme with the consideration of IMRT, where manipulation of the beam profile is performed, but not with the intent of compensating for tissue non-uniformity, but instead with the intent of generating non-uniformity at depth, even with a uniform dose deposition medium. The goal of the non-uniformity is to provide as high a dose as possible to the targeted tissues, while minimizing dose to certain critical tissues identified during treatment planning. The goal is then in contrast to the historical goals of radiation therapy of creating a uniform
dose distribution, and is instead to create as non-uniform a dose distribution as possible in specific regions.

The non-uniformity of treatment beam is typically accomplished in the modern clinical environment through the manipulation of exposed radiation field via the MLC discussed in 1.1.1. Two common approaches exist currently for the manipulation of the MLC during treatment: the first is to design a radiation aperture using the MLC, deliver a set amount of dose through this aperture, then manipulate the aperture to a new configuration; the second method is similar to the first, but allows the dose to be delivered while the aperture is moving. The common names for these two methods are step-and-shoot and sliding-window respectively. An advantage to step-and-shoot over sliding-window is that the aperture definition may be accomplished in a relatively time independent manner, so high temporal precision in the motion of the MLC system is not required; in contrast, the sliding-window technique requires good correlation between dose delivery rate and MLC motion speed, or significant inaccuracies in delivery may result. Conversely, a significant reduction in beam on treatment time may be accomplished using sliding-window over step-and-shoot reducing the potential for intra-fraction motion and potentially increasing department throughput.

In considering the development of a per-patient treatment plan for IMRT, a number of factors must be considered. The historical goal of treatment planning has been to produce a uniform dose in the target tissue, as it has been shown (10) that uniform dose provides the greatest tumor control probability (TCP); however, the goal IMRT is to trade a strictly uniform dose for decreased critical structure dose, allowing one to increase the total dose delivered while not increasing normal tissue toxicity – termed dose escalation.
Another consideration is the dose delivery to the patient which is not accounted for in treatment planning. One source of such error is the lack of consideration of photoneutrons in most treatment planning systems. Photoneutron production in a linear accelerator is primarily from the interaction of high energy photons with the collimation components of the accelerator, and so is dependant on the amount of radiation generated at the target and not necessarily the amount of photon and electron dose delivered to the level of the patient. As IMRT is performed by selectively reducing the dose output per MU, the number of monitor units – and thus the number of high energy photons generated in the head of the accelerator – is increased significantly over that which would be needed to deliver the same total dose to the patient using conventional radiotherapy. This increase in photoneutron dose results in a significant increase in the potential for photoneutron scatter dose to the patient and requires a significant increase in the shielding required for the accelerator vault. To mitigate this, many facilities use only low energies (~6MV) for IMRT, as the photoneutron cross sections for the materials primarily responsible for photoneutron production in the head of the linear accelerator have a threshold level of around 6.1MV and have a very small cross-section up to 10MV (11).

1.2.1 IMRT Quality Assurance (QA)

The complexity of the delivery process, and the significant reliance on computer designed plans required in IMRT planning creates a situation where one cannot be certain that the delivered dose will match the dose profile calculated in the treatment planning system. The potential for significant deviations from desired dose delivery mandates
caution in delivery to the patient, as there is no way to remove dose that has been delivered.

Verification of the treatment planning system calculation is often performed using an independent computer calculation system which often uses a much simpler calculation method than employed in the treatment planning system to verify the dose to a single point. The methods used in this calculation are typically a simplistic monitor unit calculation based on machine characterization parameters measured during the commissioning of the accelerator (12). This helps assure that no significant errors are made in the configuration of certain dosimetric treatment parameters, but cannot provide a verification of the deliverability of the treatment plan.

To develop a QA program for IMRT treatment plans, one must consider in what ways errors may be introduced to the delivery to the patient. Some ways in which errors may be introduced beyond those present in conventional radiotherapy include: inaccuracies in the commissioning of the accelerator in the treatment planning system, such as failure to provide appropriate corrections for rounded leaf edges as discussed in 1.1.1; failure in transmitting the treatment plan from the planning system to the record and verify system (if used); failure in the transmission of the plan from the record and verify system to the accelerator; and failure of the accelerator to properly modulate the field as intended.

Three of the four identified potential causes for error involve potential computer system errors. The field of radiation therapy has a very high reliance on computer systems, and this reliance has resulted in several high profile accidents when the computer system did not operate as expected. One example is the Therac-25 series of
accidents (13), documented as a case study in computer science regarding the danger of race conditions and code provability in code used in medical applications. A second example involving IMRT highlights the potential for significant error involves the events reported in the New York Times (14) in the beginning of 2010. In the event in question (15), the planning system failed to transfer MLC positioning information to the delivery database, while reporting that MLC positions were present; the result was a treatment of an IMRT plan with no MLC aperture present to define the field, resulting in a six-fold increase in the dose delivered. In both cases, the manufacturer of the product disclaimed any responsibility for the software failure; though, in the first case the U.S. and Canadian regulatory agencies responded after significant evidence was presented demonstrating the serious nature of the failure.

These incidents demonstrate the clear need for an additional layer of QA in situations where the correctness of a treatment delivery cannot be verified through simple inspection. To this end, many different methods of quality assurance have been devised for IMRT. The current requirements from the accrediting bodies for radiation therapy facilities are that IMRT QA be performed on a per-plan basis (16) prior to delivery on the patient which would detect any persistent systematic errors and any transient errors which happen to occur during the QA. The QA process is typically performed prior to treatment through the use of some form of planar detector. The current recommendations (16) are that the QA be performed in a calibrated manner allowing for verification of total delivered dose, as well as verification of the composite planar fluence from the linear accelerator.
In the case of transient errors which are not detected at the time of QA it is highly unlikely that the error will be detected at all, as most sites do not perform in-vivo measurements of delivered dose throughout the patient’s course of treatment. The development of a method of verifying the delivered IMRT dose using the EPID device available on most modern linear accelerators seeks to fill this potential void in quality assurance by performing post delivery verification on each treatment. While this method may not be able to prevent an error from occurring during the treatment, it could be used to detect an insidious intermittent error which could go unnoticed for multiple treatments.
Chapter 2

Monte Carlo

A fundamental difficulty in the measure or simulation of high energy particles involves the stochastic nature of their interaction with matter, precluding the direct calculation of macroscopic properties of a beam of high energy properties interacting with matter (energy deposition, beam attenuation, \textit{etc.}) using discrete methods. The calculation of radiative transport of high energy particles is typically solved using a method known as Monte Carlo simulations -- so named because of the ‘rolling the dice’ component of random interactions of simulated particles with matter, similar to the random chance of the games in the famous gambling city.

The simulation algorithm is typified by the use of cross sectional data describing interaction types and probabilities, and path lengths to determine if an interaction has occurred – producing appropriate secondary particles, including energy deposition along the particle path. The calculation is then performed repeatedly, following a large number of particles representing the known distribution in energy and position. The central limit theorem is then used to infer the mean value of the system from the average of the simulated particles. As the statistical uncertainty of the calculation is dependant only on the number of particle histories run, a simulation may be made as statistically accurate as
desired, given enough time. The requirements are then that an appropriate set of the
cross-section data, the source characterization parameters, and the physical system
properties (position, material, dimension) of everything in the region of the interaction be
available to provide a highly accurate calculation of the results of a high energy particle
beam interacting with matter.

Most Monte Carlo codes available do not track every possible interaction of each
particle generated, but instead apply various variance reduction techniques to simplify the
problem, often in a way which does not introduce significant inaccuracies in the
calculation, though it is important to identify what simplifications are acceptable for each
problem. The choice of Monte Carlo code used in a simulation is then highly dependant
on the known configuration of the problem, and what simplifications can be introduced
without significant error. Additionally, codes will often provide for user adjustable
parameters in the calculation details to allow further variance reduction while
maintaining acceptable levels of accuracy.

In selecting a Monte Carlo code for use in radiation oncology simulations it is
important to consider the primary measure of concern: dose deposited, measured in Gray
– or Joules/kg – it is the measure of energy deposited per unit mass in a phantom (17).
The radiative source type will also play a large role in the selection of Monte Carlo code,
as some systems are optimized for certain calculation types, providing significant
variance reduction techniques not available in a more generic code; often, though, this
must be tempered with the loss of generality of the code, and typically the reduction of
the set of problems for which the code is able to provide an accurate solution.
2.1 Monte Carlo codes

There exists a large number of Monte Carlo codes in common use in the radiation oncology community, including many which are commercial systems for treatment planning. Of those available in a non-commercial environment, there are several in common use including MCNP (18), and BEAMnrc (19). The following section deals with the consideration of these codes, with particular emphasis on the BEAMnrc system. A review of the literature (20) as of 2007 demonstrates the strong leading role played by MCNP and the EGS-BEAM code in the field of medical physics. MCNP is by far the most referenced in nuclear science and technology, with EGS-BEAM being the most referenced in Oncology, and “Radiology, nuclear medicine, and medical imaging.”

2.1.1 MCNP5

The MCNP5 code is the fifth major revision of the Monte Carlo N-Particle code released by Los Alamos National Laboratories. It finds its largest audience in the field of nuclear engineering, as it is one of the few codes available with full neutron transport calculation. For this reason, it is also export restricted in the United States, and requires government oversight for the acquisition of a copy of the code. The features of particular interest to radiation oncology is the ability to calculate transport in systems involving neutrons, photons, and electrons/positrons, and the ability to record a results on particle flux, and energy deposition, providing the ability to calculate dose.

The MCNP5 system provides a very flexible method for defining problem geometry and composition, as well as ‘tallies’ which are used to define what results are recorded during the simulation run. The system also provides the ability to define a
radiative source with particles being generated with a dependence on particle, energy, 
time, position, direction, cell, surface, and any combination of these (18). Thus, one may 
fairly precisely define a source to match any source used commonly in radiation 
oncology, from an Ir-192 High Dose Rate brachytherapy source (21), to the neutron 
production in a Linear Accelerator head (22).

One of the key advantages of this code over other codes available is the ability to 
simulate neutron interactions, as few other codes have this function. As discussed in 1.2 
above, IMRT planning is typically done at photon energies below 10MV due to the 
relative increase in photoneutron contamination at energies above 10MV. Many modern 
accelerators provide only one energy below 10MV, typically using a beam of 6MV. 
Given that the primary producer of photoneutrons in a medical linear accelerator is the 
target, typically made of Tungsten, and that the threshold energy for Tungsten has been 
shown to be above 6 MeV (11), accurate neutron treatment is not necessary for the 
purposes of this project.

2.1.2 BEAMnrc

The BEAMnrc code is an extension to the EGSnrc (23) code with a primary focus 
on simulating medical linear accelerators. The EGSnrc code is an update to the EGS4 
(24) code designed to simulate an Electron Gamma Shower using Monte Carlo. EGSnrc 
is focused on the transport of electrons, positrons, and photons through and into 
materials. The materials may be defined using a set of existing cross section data created 
by the EGSnrc maintainers, based on reported density effect corrections (25). The two 
supplied cross section templates provide 45 materials commonly found in simulations in 
radiation oncology, such as: air, water, Tungsten, kapton, mylar, Copper, Cerrobend®,
Lead, etc. This allows one to simulate a linear accelerator head without needing to produce additional cross-section data; though, the EGSnrc system does provide a tool for creating cross-section data from an arbitrary mixture of elements from Hydrogen through Fermium inclusive.

The BEAMnrc code is designed around a single source, defined by the ISOURCE parameter (incident source), entering a sequence of component modules and proceeding through them with results being calculated at up to four ‘scoring planes’ in the simulation. The component modules (CM) are stacked together, in order of increasing Z (typically used as distance from the source), to form the accelerator model. Space between component modules is treated as being air-filled.

The source routines available in BEAMnrc are designed to provide generation of history start points for the Monte Carlo calculation. The current version of BEAMnrc provides 16 source types which are defined in the simulation parameters. For most source types, it is logical to treat the source as occurring at a Z position less than the start of the accelerator and impinging on the first CM. Many source types allow for the user to specify the energy spectrum of the source, in addition to the charge of the source particles, and various geometric parameters specific to the source type. Source types typically used in the simulation of a medical linear accelerator are: type 0, a parallel circular beam which may be used to simulate the electron beam exiting the acceleration column; type 1, an isotropic point source directed in the positive Z direction with a given size, which may be used to simulate the result of an electron beam impinging on a target; or type 21/24, which allow the use of phase space files generated as output of previous simulations to be used on an arbitrary CM of the model.
Each component module is a self-contained element in the accelerator model, with a front and back plane. The modules must conform to a particular interface for communicating with the BEAMnrc code and may be designed to simulate an arbitrary physical component in a linear accelerator system. Some components provided with the BEAMnrc code include MLC, JAWS, MIRROR, SIDETUBE, FLATFILT, etc., providing simulations for components matching their name. The code provides a full description of the code requirements of a component module, so users may create their own modules if none of the existing ones are sufficient, or may modify existing modules to suit their needs.

The outputs available from the BEAMnrc are defined in the scoring zones section of the input file, where one may define up to four zones to score fluence and dose results. Additionally, one may request a phase space file be output at the location of each scoring plane, which is a file containing each particle passing through the plane of the scoring zone. The format of this phase space file is discussed in more detail in Chapter 6.

The BEAMnrc code also allows for the setting of LATCH bits on a per particle basis. The typical use for these LATCH bits is to store the regions in which a particle has interacted, and the code is designed to set the LATCH bits based on the configuration given to the component modules and the LATCH parameter specified in the simulation input.

Given the directed nature of this code towards the type of problem we are trying to solve, BEAMnrc is the Monte Carlo code that was selected for use in this project. Our primary reason for selecting this code is that it is particularly well suited to the design of a linear accelerator and simulation for clinically relevant energies. The verification of the
quality of this *monte carlo* code has been performed by multiple authors. One such study, Chibani et al. (26), demonstrates good agreement with measurements within the clinically useful range of energies; as well as good agreement with the commonly used MCNP code. Further details of implementation specifics are given in Chapter 4.

Additionally, the BEAMnrc code includes several other EGSnrc user codes focused on typical uses in radiation oncology. These codes include the DOSXYZnrc user code which is designed to calculate dose deposition in a Cartesian coordinate system of voxels, including in a CT.

### 2.2 Variance Reduction Techniques

Variance reduction is the process of manipulating the problem definition to reduce the calculation uncertainty. The uncertainty of a given Monte Carlo problem is inversely dependent on the square root of the number of histories, and a directly related to a fixed uncertainty based on the approximations used in the problem definition. The naive way to decrease uncertainty then is to increase the number of histories run; however, as the simulation time is directly related to the number of histories run (ignoring any problem setup time), attempting to gain a tenfold decrease in uncertainty requires a hundredfold increase in simulation time.

Due to the time prohibitive nature of this approach, each Monte Carlo code implements some method of providing reductions in the complexity of the problem to reduce the amount of time spent per history, which can allow for a significant increase in the number of histories run for a given time. The difficulty then becomes in selecting which approximations do not introduce significant uncertainties not related to the pure statistical uncertainty of the simulation.
It is often the case that a significant portion of the time spent designing a Monte Carlo simulation for any particular code involves the selection of variance reduction techniques suited to the particular problem. Each code discussed devotes a significant portion of the manual to implementing appropriate variance reduction techniques in the problems solved by that code (18; 19). It is well that they do so, as implementing variance reduction techniques without a clear evaluation of the potential effects can result in a simulation which appears to have a very low measure of uncertainty, but does not provide an accurate model of the physics being simulated.

2.2.1 MCNP5

The MCNP5 manual describes four categories of variance reduction techniques available for manipulation by the user: truncation, population control, modified sampling, and partially-deterministic methods (18). Truncation involves terminating histories based on various criteria, effectively truncating that particle history at that point. Population control involves increasing the number of histories run for more ‘interesting’ particles (as defined by the use), adjusting the result weight of those particles appropriately. Modified sampling involves manipulation of the tally sampling to increase likelihood of a tally being performed while decreasing the weight appropriately. Finally, partially-deterministic methods replace the history following with a direct calculation, or modifications to the history, for specified conditions.

2.2.2 BEAMnrc

The BEAMnrc manual lists four major categories of variance reduction techniques offered as tunable parameters in the BEAMnrc code: range rejection,
bremsstrahlung splitting/Russian roulette, photon forcing, and Bremsstrahlung Cross-Section Enhancement (BCSE). The first two techniques are adaptations/replacements of the techniques provided in the EGSnrc code. In addition to these techniques, variance reduction may be accomplished by careful selection of calculation parameters provided to the EGSnrc code underlying the BEAMnrc code.

2.2.2.1 Range Rejection

This is an extension of the range rejection algorithm provided by the EGSnrc package. The basic principle involves calculating the expected range of the current particle and comparing this to the cutoff value for the current region. If the electron/positron does not have sufficient range, it is considered as having deposited its entire energy in the local cell. For a conservative estimate, the range is calculated using the restricted stopping power.

There are two methods for applying this range rejection in the BEAMnrc code, the first involves treating each region separately and calculating the range rejection for each particle to the edge of the region. If the particle does not have sufficient range to reach the edge of the region it is stopped there. The second method also checks if the particle has sufficient energy to reach the bottom of the last CM in the model, allowing it to stop tracking particles which will not make it to scoring planes at the end of the accelerator model.

The selection of an appropriate cutoff energy is important, as terminating histories prematurely will not allow the production of any secondary photons that would normally result from the particle. To mitigate this, a separate energy cutoff is provided, above which range rejection will not be considered.
2.2.2.2 Bremsstrahlung splitting and Russian Roulette

This method would fall under the categories described by MCNP as ‘population control’ variance reduction techniques. The bremsstrahlung splitting technique has three sub-methods, but all work by creating multiple bremsstrahlung photons each time an electron/positron goes through a bremsstrahlung event, with the weight of each photon reduced by the number produced. The electron is treated as though it had undergone only one bremsstrahlung event, but since the weights are adjusted appropriately, this does not introduce significant error in the total.

2.2.2.2.1 Uniform Splitting

The BEAMnrc allows the EGSnrc code to split each bremsstrahlung photon into multiple photons as though the event had occurred multiple times. Each photon is fully followed and the probability distribution of photon directions is uniformly distributed.

2.2.2.2.2 Selective Splitting

In selective splitting, an aperture is defined at a distance from the target in which it is desired to have low variance. The code calculates the probability of a bremsstrahlung photon being created and passing through the defined aperture, and uses this to determine the number of photons generated during splitting. This allows one to selectively ignore splitting which would result in a significant number of events leaving the area of interest of the simulation.

2.2.2.2.3 Charged Particle Russian roulette

Both of the previous splitting methods may employ a Russian roulette system, so named because the particles are terminated based on a probabilistic method. With
Russian roulette, secondary charged particles in the histories of the particles generated by bremsstrahlung splitting (via Compton events, photoelectric events, and pair production) will be given a survival threshold inversely related to the splitting number of their parent product, and a random number is assigned to the particle. If the random number is larger than the survival threshold, the particle is never followed; otherwise, the particle’s weight is increased by the survival threshold and it is tracked. This relationship effectively restores the number of charged particles tracked to the same value as if no bremsstrahlung splitting were used, while allowing for much greater bremsstrahlung photon numbers (19).

2.2.2.2.4 Directional Bremsstrahlung Splitting

Following the same idea as selective splitting, this method seeks to enhance the ratio of bremsstrahlung splitting in the direction of the area of interesting in the simulation. In contrast to the selective splitting method, this method applies Russian roulette to both photons and charged particles generated from the splitting event, but only for photons if they are not directed at the destination aperture. If a photon survives the Russian roulette, it and its descendants are termed ‘fat’ and treated as a high weight thereafter, but charged particles continue to receive roulette terminations. This results in a significant increase in photon fluence at the area of interest, but suppresses charged particles. In simulations where the lack of charged particle dose deposition would produce significant challenges to the accuracy, a charged particle splitting occurs to all ‘fat’ charged particles passing through a plane specified by the user. Below this plane any time a ‘fat’ particle interacts it is split and carried as a low weight particle not subject to Russian roulette.
As the ‘fat’ nature of a particle is not stored in the properties of a particle in the phase space file (see 4.3), additional care must be taken in handling the particles if a phase space output of the model is to be used in further simulations or analysis.

2.2.2.3 Photon Forcing

This option allows one to specify a region in which one wants to enhance the number of photon interactions, typically region which would have few interactions in reality. To maintain accuracy, when a photon is forced to interact, it is split prior to interacting. The photon forced to interact is given a percentage of the original weight proportional to the unforced probability of interaction, while the other photon is given the remaining weight, and forced not to interact for the rest of the region.

2.2.2.4 Bremsstrahlung Cross-Section Enhancement

In this method, a single medium as defined in the material file, has its bremsstrahlung cross sections uniformly scale by a supplied factor. This factor is then used to reduce the weight of any bremsstrahlung photons generated in this material to maintain accuracy. The original charged particle’s energy is randomly reduced by the energy of the bremsstrahlung photon with a probability inversely related to the scaling factor. A Russian roulette option is implemented which, when enabled, eliminates the charged particle products of interaction with a probability inversely related to the scaling factor in a similar manner as described in 2.2.2.2.3.

2.2.2.5 EGSnrc transport parameters

Additional variance reduction may be accomplished by adjusting the parameters used by BEAMnrc in EGSnrc, the most interesting of which would be the selection of
algorithm to be used for boundary crossing, and for electron stepping. Additional parameters are available for the selection among various cross section databases for bremsstrahlung interactions, Compton scattering, pair production, and elastic scattering. Many of the parameters allow for significant variance reduction if the particle energy being simulated is relatively high, and the default set of parameters selected by BEAMnrc are optimized for megavoltage beams (19).
Chapter 3

Cluster Design

In order to accurately calculate solutions to large scale complex problems in modern science fields, in a reasonable amount of time, it has been necessary to use a significant amount of computational power. One method of obtaining a large amount of computational power for relatively lower cost is to combine multiple lower power systems into a combined computing cluster. The advantage of using a cluster approach is that lower cost individual systems may be employed, and the cluster may be expanded as more computational power is required.

In this research project, two computational clusters were created for use in this project, and in future projects in the department. The first cluster system was created using computer systems removed from active use in the department and repurposed for the cluster use. As the demands of the calculation began to outgrow the available resources, a new cluster was commissioned designed to be a low cost custom built cluster able to significantly increase computation power available.

The usefulness of a cluster system depends on the ease with which it may be made to run the code required by its users. The primary calculation code bases run at our facility prior to this project were the MCNP5 and BEAMnrc codes, and in more recent
work the COMSOL physics package. In this project, much of the calculation is done using various user codes written in Python, in addition to the BEAMnrc code. The accommodation of this broad range of code bases was a key part of the design of the software for the clusters. Even so, at the moment, the COMSOL package has not been incorporated into this cluster.

3.1 Parallelization considerations

Both Monte Carlo systems discussed include functionality for running in a parallel environment. Consideration of the problem being solved is critical in order to parallelize any type of calculation. Methods of parallelizing may be as complicated as requiring information from every cell in the parallel calculation before each step, to as simple as allowing each system to operate completely independently, combining the results after the simulation has completed. In the case of simulations of concern in radiation oncology, most simulations assume that effects from the radiative source to not persist in the medium in a manner which affects particles later in the simulation. This offers us complete temporal and state independence for simulations, allowing us to consider calculations as “embarrassingly” parallel, or requiring no interdependence on the state of any other members.

In the case of MCNP5, one has several options for parallelization, with implementation depending on compiler available. If the MCNP5 code is compiled with Message Passing Interface (MPI) or Parallel Virtual Machine (PVM) enabled, it may be run with a master controlling the calculations to be performed and the nodes being given the calculation tasks (26). The MCNP5 manual also suggests operating in an “embarrassingly” parallel configuration, running the same simulation on multiple nodes
with varying random number generator seeds and combining the result tallies after completion. In the case of BEAMnrc, this is the only method of parallelization offered, though the code is designed to automatically adjust the random number generator seeds for each calculation worker.

The need for selecting a unique random number generator seed is due to a series of compromises which must be made in Monte Carlo codes regarding the “random number” portion of the Monte Carlo code. Specifically, a typical expectation of a computer calculation is one of a deterministic result for a given set of input parameters. This expectation extends to the desire to evaluate how specific changes affect the result, even in the simulation of a stochastic event such as Monte Carlo. In Monte Carlo, one may also wish to continue a previous simulation directly, without needing to consider statistical uncertainty calculations for combining distinct simulations. Additionally, the generation of truly random numbers in a computer environment is a significant concern in the field of cryptographic security (27), as a poor random number generator can result in the potential of cryptographic failure. To generate a true random number on a computer system an external source of entropy is typically necessary to provide the randomness as the primary design goal of microprocessor systems is to eliminate non-deterministic results.

As result of these considerations, both Monte Carlo codes discussed use “pseudo random number generators” which employ algorithms which provide a uniform distribution of values with an apparent lack of correlation between successive evaluations which may be calculated quickly. In the case of MCNP, these sequences of pseudo-random numbers are controlled by five parameters, the selection of which can have a
significant effect on the statistical uncertainty of the simulation (28). The BEAMnrc code supplies two different algorithms for pseudo-random number generation, the ‘RANMAR’ algorithm (29), and the ‘RANLUX’ algorithm (30); both algorithms provide two parameters for adjusting the seed of the random number generator. In BEAMnrc, when a simulation is run with identical seeds, and given the output configuration option to ‘restart’ the simulation, it will continue the simulation as though it had not stopped.

3.2 The Two Clusters

The layout of the two clusters is very similar, with one system being used in both. The initial cluster was named the “Torque Cluster” after the job scheduling service used in it, while the second cluster was named the “Blade Cluster” after the primary computing components. In both clusters, one server acts as the master for all of the sub nodes. For a comparison of the two systems, see Table 3.1, below.

For both clusters, the Debian Linux operating system, release 6.0.2 (codename: “Squeeze”), was chosen as the base due to the high availability compilers and compiled packages for this system, and due to the long term stability and support of the Debian distribution. This simplifies the administration of multiple nodes by having a long life cycle for each software version on the node.

3.2.1 Torque Cluster

The first cluster was created from available computer systems in the department which had been retired from clinical use, and two dedicated servers obtained previously. The majority of the systems used Gateway E4100 with Intel Pentium 4 HT 2.8C 2.8 GHz processors. This processor provides Intel’s Hyper-Threading Technology, which allows
the system to treat each single core processor as two virtual cores (31). Unfortunately, most of the highly serialized and optimized calculation operations performed in modern Monte Carlo codes negate this benefit, as the code is set up to minimize processor stalling (time where the processor is waiting for external interfaces, e.g. cache-missing); thus, the expected twofold increase in processing power is not often attained. It has been demonstrated using MCNP5 that jobs using greater than eight physical processors see gains from enabling Hyper-Threading outweighed by the additional communication overhead between nodes (26).

The master node was created from one of the two dedicated server systems, a Dual Xeon 3.06 system with 4GB of memory. For storage, it has a SCSI U320 host backplane, on it a 147 GB drive. Additional storage was added for storing historical calculations, a 400 GB ATA drive. This master was configured to also act as a node for calculations, with all of the data stored on the SCSI drive.

The nodes are a set of six E4100 systems and a single server similar in configuration to the master node. The process of configuring the E4100 nodes was simplified by installing Debian 6.0.2 on one of the nodes. This node was configured with the required software packages and given the hostname “beamcluster.” Finally, the SSH “host keys,” which are automatically generated cryptographic keys designed to uniquely sign each system providing a Secure SHell daemon (SSHd), were removed from the system. A mirror of the system was created using the PartImage is Not Ghost (PING) Linux distribution, and was stored on the master node. Each node was then restored from this master node image, providing identical systems for the cluster. At first boot, each
system was given a unique hostname, and automatically regenerated the necessary SSH host keys.

Two unique users were added to each system in the cluster for running user codes: the “beamuser” user, and the “mcnp” user; given default passwords of “beamnrc”, and “mcnp5” respectively. In order to simplify managing the simulation results, none of the nodes in the cluster have local home directories for these two users, and instead mount the home directory for the appropriate user from the master node using the Network File System v4 (NFS) protocol, mounting using the Linux kernel based automounter file system (autofs5) to mount only when required. While this configuration results in significant network utilization – as the home directory contains all of the data, and code, for the BEAMnrc and MCNP5 codes – it was felt it would not significantly reduce performance. The reasoning is that both Monte Carlo codes use a fairly small amount of memory, and the Linux kernel is capable of “read-ahead” which allows it to load significant portions of expected program data into main memory not currently in use (32).

<table>
<thead>
<tr>
<th>Component</th>
<th>“Torque Cluster”</th>
<th>“Blade Cluster”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processors</td>
<td>6 x Intel Pentium 4 HT 2.8 GHz&lt;br&gt;4 x Intel Xeon 3.06 GHz (Gallatin)</td>
<td>32 x Intel Xeon 1.86 GHz L5320 SLAEP</td>
</tr>
<tr>
<td>Memory</td>
<td>2 x 4 GB DDR PC-2100&lt;br&gt;6 x 1 GB DDR PC-2700</td>
<td>64 x 1GB DDR2 PC2-5300 (4/node)</td>
</tr>
<tr>
<td>Local Storage</td>
<td>2 x 400 GB ATA-133, 2 x 147 GB&lt;br&gt;6 x 40 GB ATA-133</td>
<td>None</td>
</tr>
<tr>
<td>Network</td>
<td>803.2ab</td>
<td>803.2ab</td>
</tr>
</tbody>
</table>

### 3.2.2 Blade Cluster

The second cluster was envisioned after noting the limitations of the previous cluster in terms of raw computation power in real world usage. Several large scale
simulations were required for the project involving 5 trillion histories, with an estimated run time of on the order of several weeks on the existing cluster; this was found to be infeasible due to hardware failures (specifically CPU fans) occasionally bringing nodes down, resulting in a failure of the entire simulation. To improve reliability and drastically increase the computing power available, a cluster was designed employing an HP BladeSystem c7000 enclosure and 16 HP BL460c blade systems.

The blade style system is well entrenched in modern datacenters as an efficient design for high density of computing power for rack space, as well as hardware infrastructure. The design separates the ancillary components of the system from the critical computing components, allowing each to function optimally in their disparate tasks. That is, the power supply, active cooling, system interconnectivity, and management systems are located in the BladeSystem enclosure, relieving each blade of the need to supply each of these components locally; each blade is then only responsible for mating to the power/fiber backplane and providing local system functions.

In the configuration we have designed, each BL460c server contains two Intel Xeon L5320 1.86 GHz processors, with those processors that were found capable being overclocked to 2.33 GHz (24 of 32 processors were found to be stable at 2.33 GHz under heavy load, those found to be unstable were reduced to the stock speed of 1.86 GHz). Each node is also given 4 GB of PC5300 DDR2 Fully Buffered memory, so chosen because the previous code had been compiled and verified for 32bit operations, and 4 GB is the largest addressable memory register on a 32bit system. Each system is capable of holding 64GB, and may be expanded 32GB without removing the existing memory outlay, so this compromise was not found to be disadvantageous in the long term.
To maintain a high level of control over the cluster, each system is connected via an internal IEEE 802.1ab compliant backplane to a 24 port Layer-2 Gigabit switch, which is then connected to the master node only, via a software aggregated 4 port Gigabit link (33). This configuration keeps the entire cluster directly linked to the master node via a high-availability, high-throughput link, while preventing external systems from directly accessing the nodes. The master is configured as a Dynamic Host Configuration Protocol server (DHCP) to provide unique locally administered IP addresses from the Internet Assigned Numbers Authority (IANA) reserved class C subnet 192.168.0/8 (34) to each blade system. The master is then configured to provide itself as a Network Address Translation Masquerading router for the blades, allowing each blade access to systems outside of the local blade network.

As each system was configured without any local storage for cost savings, the nodes are configured to boot using the Preboot eXecution Environment (PXE) using the syslinux pxelinux (35) utility. Each system then loads the same custom version of Debian configured using the Debian Live-Build tools. This method was chosen after experience gained with the previous cluster, as it allows one to roll out updates to the operating system, and installed packages, on all nodes very simply -- one needs only rebuild the live system, optionally test it prior to deployment, deploy by replacing the previous live build, and finally reboot the blades to activate the change.

Several custom modifications to the live-build system were required to configure the blade systems as needed for this project. The additional programs created to aide in this are shown in Appendix A. As each blade is booted, it determines its local hostname based on the DHCP optional hostname return parameter, and configures the monitoring
software. Additional configuration parameters are set by the live-boot system, and when the system completes booting, it is configured to use Network Information Service to determine acceptable usernames and for password verification for those users via the master system. As with the previous cluster, the home directories for the user codes are mounted using NFS, allowing seamless program function.

3.3 TORQUE Resource Manager

In the “Torque Cluster” configuration, a user is able to physically access each system in the cluster, and also remotely launch an X-Windows session on each node, allowing one to manually launch each calculation; however, this is a tedious process for large computation tasks. To simplify the process of submitting jobs, as well as provide the ability to track usage, it was decided that a job scheduler would be needed for the cluster. The system was chosen primarily based on what was documented as being supported in the BEAMnrc and what was available in the Debian repositories. The BEAMnrc code provides support for three batch systems (2), the first of which (‘at’) is not designed for use across multiple nodes. It was decided that the Portable Batch System (PBS) would provide the best fit for the cluster, with an actively maintained version provided as the Terascale Open-source Resource and Queue Manager (TORQUE) via Adaptive Computing (36).

The TORQUE PBS system has one master node which handles configuration of the available computing resources. This system also typically handles scheduling of jobs to be performed on the resources available to the master using a PBS scheduler – initially we used the version available as torque-scheduler in Debian. Each node in the system is configured to run a PBS Machine Oriented Miniserver (MOM) process which reports to
the master system the resources available from the node, and also handles launching tasks
locally at the request of the master system.

The TORQUE batch system was configured with MCO0033850742 as the master
node, and all systems as available compute nodes. An initial queue was created
following recommendations in the Administrators Manual (36), with the designation
“batch.” This queue was set as the default queue, and the PBS job system was configured
as the default batch submission system for the BEAMnrc system. At the time of the initial
configuration, no nodes were given higher priority than others, or any special
configuration, other than the number of virtual processors available on each node. This
allowed for simple job submission, as all three options in BEAMnrc for batch submission
would submit to the same queue.

After the creation of the Blade cluster, it was determined that the two systems
should not have jobs split between them, due to the significant difference in power per
node, so a separate queue was created for the “Blade Cluster,” the “fast” queue. To limit
queues by machine type, a more advanced PBS scheduler was required, so the torque-
scheduler was replaced by the Maui cluster scheduler, which allows one to restrict
compute nodes used in a queue based on server configuration parameters. The result
allows very fine-grained control over which resources are allocated to a particular job, as
shown in Table 3.2. Additionally, each execution system may be specified in the batch
run command for very fine-grained control; however, this will force the task to wait for
all nodes specified to be available, rather than running on the resources available at the
time of task execution.
### Table 3.2: Resources allocated by node identifier

<table>
<thead>
<tr>
<th>Resource identifier</th>
<th>Systems allocated</th>
</tr>
</thead>
<tbody>
<tr>
<td>peeeees</td>
<td>node1, node2, node3-node8</td>
</tr>
<tr>
<td>simple</td>
<td>node3-node8</td>
</tr>
<tr>
<td>noserver</td>
<td>node2-node8</td>
</tr>
<tr>
<td>server</td>
<td>node1</td>
</tr>
<tr>
<td>blades</td>
<td>blade1-blade16</td>
</tr>
</tbody>
</table>

### 3.4 Custom Code Modifications

After completing the construction of the “torque cluster,” it was necessary to compile and deploy the two Monte Carlo codes intended to be available for the cluster discussed in Chapter 2. During regression testing, an issue was identified in the BEAMnrc codebase, specifically the EGSnrc code for reading from the lockfile for parallel runs had a race condition which caused large scale relatively fast runs to deadlock and subsequently fail out. A modification to the code was implemented which solves this deadlock by eliminating a delay loop in favor of filesystem level blocking I/O commands. Several other minor modifications to the code were implemented to enhance the types of calculations used in this research, *e.g.* large flat arrays requiring modifications to user macros within the scope of the documentation for EGSnrc.
Chapter 4

Accelerator Model Creation

The accelerator model created for this research is intended to provide an analog of the design of a modern linear accelerator, while not being based on a particular existing accelerator or accelerator design. The decision to create a model which has no physical counterpart presents some difficulty in attempting to verify simulations with measurements, but the compromise was felt to be acceptable as this project is intended as a ‘proof of concept’ to demonstrate the theoretical feasibility of this algorithm while avoiding introducing the difficulties of matching an existing linear accelerator. An additional advantage provided in this approach is the relief from developing legally binding confidentiality agreements with an accelerator manufacturer to obtain specifications necessary for creating an accurate model. Several manufacturers of clinical linear accelerators sell packages with much of the information required to commission a Monte Carlo model of their linear accelerators, but these packages are typically quite expensive, and come embedded with agreements restricting the publication of the results. In order to maintain complete autonomy in this research project, it was determined that a model would be generated which is similar in construction to existing linear accelerators, using reasonable values for the physical
positions and dimensions of components, and using values available in the literature for those components of a more complex design. The accelerator model is then fully describable in this publication, with no restriction on its reproduction in print.

As discussed in Chapter 2, the BEAMnrc codebase is to be used in this project. The design of an accelerator in the BEAMnrc code involves the stacking of a sequence of CMs into a beam line. Additionally, one must specify simulation-time parameters including the source characterization and the number of histories to be simulated. Thus, the accelerator model in BEAMnrc is split into two segments: this first identifies the names, types, and order of component modules to be used in the accelerator model; and the second defines all of the parameters required for each component module. The first section is compiled into a MORTRAN macro (MORTRAN being a macro extension to the FORTRAN language written for EGSnrc), and then compiled with EGSnrc into an executable code. This program then accepts as input the materials definition file and the simulation input parameters file in order to run the code. The following sections will consider first the model definition, and subsequently the main configuration parameters.

### 4.1 Component Module Sequence

The component modules used in the definition of the accelerator head are listed in Table 4.1 along with a short description of the function each component is intended to serve in the model. As the definition of the parameters of each component is on a per model basis, discussion of the specifics of each model is left for the following section. The second simulation model, after the sequence of the accelerator head is described in Table 4.2.
<table>
<thead>
<tr>
<th>Identifier</th>
<th>CM Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIMCOL</td>
<td>CONS3R</td>
<td>The primary beam collimator, designed using a stacked series of radially symmetric cones.</td>
</tr>
<tr>
<td>FLATTENF</td>
<td>FLATFILT</td>
<td>The flattening filter, designed to produce a flat profile at depth.</td>
</tr>
<tr>
<td>CHAMBER</td>
<td>FLATFILT</td>
<td>Model of Ionization chamber as used for evaluating delivered dose.</td>
</tr>
<tr>
<td>MIRROR</td>
<td>MIRROR</td>
<td>Aluminized Mylar mirror placed in beam path to allow the projection of a light field at patient level.</td>
</tr>
<tr>
<td>SECJAWS</td>
<td>JAWS</td>
<td>A pair of aperture defining devices to limit the extent of the radiation field at per delivery level.</td>
</tr>
<tr>
<td>MLC</td>
<td>DYNVMLC</td>
<td>Multileaf Collimator component with the ability of defining time dependent dynamic positions.</td>
</tr>
<tr>
<td>AIRSLAB</td>
<td>SLABS</td>
<td>Air gap after MLC component and PMMA plane defining the end of the accelerator.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Identifier</th>
<th>CM Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHANTOM</td>
<td>SIDETUBE</td>
<td>A Cylindrical phantom intended to encompass the entire radiation field in the cylinder-axial direction.</td>
</tr>
<tr>
<td>TABLE</td>
<td>SLABS</td>
<td>The treatment couch surface supporting the phantom.</td>
</tr>
</tbody>
</table>

### 4.2 Simulation Input Parameters

To run a simulation in BEAMnrc, after one has generated an accelerator module by defining the desired sequence of component modules, one creates an ‘egsinp’ input file to define the parameters of each module. This file contains all of the physical parameters as well as the materials definitions for the simulation, allowing a great deal of flexibility in simulating an accelerator while maintaining a small code base. The two accelerator modules described in 4.1 above are described in detail below, with the corresponding ‘egsinp’ files reproduced in Appendix C.
In the accelerator model we have designed, we begin the simulation after the level of the X-ray target described in 1.1, and design the simulation to begin with a spectrum based isotropic photon source. The primary motivation behind this method is to match the photon spectra with previously published spectra \(^{37}\), and eliminate the need of implementing bremsstrahlung splitting based variance reduction techniques which would be critical if the beam were simulated beginning with an electron beam through the vacuum exit window of the acceleration column.

The spectra obtained from the exit of the accelerator has been evaluated using the BEAMdp tool provided in the BEAMnrc package is shown in Figure 4-1, and compares favorably with the published spectra of Mohan, et al.
4.2.1 Accelerator Head Model

The accelerator head model includes seven distinct component modules which are configured to simulate the entire head of a linear accelerator after the level of the target/beam-stop. For all simulations, the accelerator head may be considered as two separate sections: the first section which extends to the top of the JAWS CM is static through all treatments with the same energy; the section starting with the JAWS CM and extending to the PMMA exit window (and ending at 60 cm from the effective source) is typically varied to conform to patient specific parameters, in particular in this research the MLC component will be varied significantly. This could allow a simpler simulation
configuration whereby the first section is simulated and a phase space file generated which could then be used as a source in the simulation of the patient specific portion of the head. Initially this was the approach we took; however, due to the large number of particles required at the level of the JAWS CM to attain sufficient statistics at the level of the phantom, file size and disk access speed became a significant loss of computing power. A second option was investigated whereby the input into the second half of the accelerator head section was generated dynamically at simulation time using a set input file and the “BEAM Simulation Source” source option, but the overhead involved in transferring histories from the upper head section to the lower head section continued to prove too costly, resulting in a nearly ten-fold speed cost relative to the final design involving a full head simulation. As result, the simplest approach of simulating the full head in one run proves to be the fastest approach, and the approach used in the rest of this research.

Each component module will be described in how it is implemented, along with the materials generated by the BEAMnrc code for visualizing the component during input generation in the BEAMnrc Graphical User Interface (GUI). The simulation represented is one which would produce a 10x10 cm² field at 100 cm Source-to-Surface Distance (SSD). All of the parameters are in reproduced in C.1, and many component specifics are drawn from the BEAMnrc example files provided with the code. Any segments whose material has not been specified are set to air, and all material specifications use the 700ICRU PEGS4 input file provided with BEAMnrc (2). This material specification provides accurate electron cross section data for total energies above 700 keV, thus we must select the global electron cutoff energy to be 0.7 MeV throughout the simulation.
All component modules are configured to be associated with LATCH bit 0, which results in no LATCH bits being set on any particles except for LATCH bit 23, which is set by interactions in any medium not given a specific LATCH setting (e.g. the air between components).

4.2.1.1 PRIMCOL – Primary Collimator

This component is the primary beam collimator configured as a 7.8 cm thick cone composed of iron beginning 2.1 cm from the isotropically radiating photon source. It is configured to expose a diameter of approximately 50cm at 100 cm SSD.
4.2.1.2 FLATTENF – Flattening Filter

The flattening filter design used in this research is optimized for a 6MV photon beam and provided in the BEAMnrc package. It is simulated by the FLATFILT CM using a series of 13 layers of radially defined conic segments. Each conic segment is composed of either: air, lead, or tungsten with the final design shown in Figure 4-3.
4.2.1.3 CHAMBER – Monitor Chamber

The monitor ionization chamber is simulated using a series of 35 layers of the FLATFILT CM with twin layers of Copper sandwiching a steel/mica/steel layer. All of these materials have layers of air between them. Though this is modeled using the FLATFILT CM which is designed to have cones, each layer is instead modeled as a thin
uniform disk of radius 7.5 cm surrounded by air. Once more, this component is based on the reference design distributed with the BEAMnrc code. The primary reason for this element being modeled using the FLATFILT component, rather than a SLAB component which would initially seem ideal, is that the FLATFILT component is coded with additional specific routines to accurately handle the directional bremsstrahlung splitting code, thus one can use the monitor chamber as a reference for calibrating dose delivered to a phantom with dose delivered in the CHAMBER module to provide an absolute calibration for the model. As our research does not intend to perform absolute calculations, and DBS has not been enabled, these enhancements are not necessary, but by retaining the existing model design, we may simplify use of these features in the future. The appearance of this stack may be seen in Figure 4-4.
4.2.1.4 MIRROR – Mirror

The mirror component consists of a 50 micron thick Mylar sheet angled with respect to the X-axis of the accelerator column. The mirror component is unlikely to introduce a significant change in beam profile (38), and it is unlikely to have a significant effect on the simulation time.
4.2.1.5 SECJAWS – X/Y Secondary Collimators

The secondary collimators are the first point in the acceleration column where patient-specific parameters are expected to be involved. This component module provides a paired Tungsten slab in the X and Y direction of the accelerator, with the ability to set the upper and lower edge of each. This allows one to simulate a double-
focus machine, or a machine in which both the X and Y jaws are capable of providing an appropriately diverging blocking face; the alternative solution employed in Elekta brand linear accelerators is a single-focus machine, wherein one secondary collimator (the collimator not aligned with the MLC) is capable of matching divergence, while the second collimator uses a rounded edge and linear travel corrections to shape the field.

The second consideration with this module is whether it is desirable to simulate an accelerator which can position the secondary collimators dynamically during the treatment. For simulation simplicity, and to be consistent with the style of machine most closely resembling this model, we have opted for the fixed collimator simulation, in line with the method employed by the Varian bran linear accelerators. A graphical representation of the X and Y jaws, shown in XZ and YZ projections are shown in Figure 4-6 and Figure 4-7 respectively.
Figure 4-6: Secondary collimators shown in XZ view. The top-most collimator is the X collimator and is shown configured to create a 10x10 cm² field at 100 cm SSD.
Figure 4-7: Secondary collimators shown in YZ view. The bottom-most collimator is the Y collimator and is shown configured to create a 10x10 cm² field at 100 cm SSD.

4.2.1.6 DYNVMLC – Dynamic Multileaf Collimator

The DYNVMLC CM is used here to simulate an MLC design based on the Varian Millenium MLC. Due to a lack of publically available information on this design, the parameters defining physical leaf cross sections, including tongue and groove width, offset, overlap, etc. were based on a dissertation from N. Tang (39) where values were presented which correlate with expected positions.
The DYNVMLC module has an additional set of configuration parameters not seen in the component modules discussed to this point, the ability to specify a time sensitive position parameter file. This specification provides the ability to simulate a treatment aperture which changes dynamically during the ‘beam-on’ time by defining the position of the leaves and the percentage of the time spent at that position. The BEAMnrc code will then determine at the start of a particle history what the leaf configuration is by random selection with the probability of each configuration defined by the relative percent of the delivery defined in the configuration file. Additionally, an option exists to define ‘step-and-shoot’ vs. ‘sliding-window’ format for leave positions; if the former is selected, the simulation uses the precise positions listed in the configuration parameter; if the latter is selected, the code interpolates the position of the leaves based on the history’s time component that was randomly generated, thus allowing for the simulation of continuous motion to within the accuracy of the floating point numbers storing the position of the leaf positions.

During simulations for open field configurations, the leaves are configured to be in a static configuration and pulled out to 10 cm from the Z-axis, placing them well behind the X and Y secondary collimators. This is to simulate the behavior of the Varian Millenium MLC system which was originally designed as an upgrade option to the standard accelerator, thus by pulling the MLC out of the field it would not contribute significant additional scatter to the patient when not in use.

The representation of the MLC in the XZ and XY projections shown in Figure 4-8 and Figure 4-9 respectively demonstrate this position, while Figure 4-10 is a YZ projection at the X-axis with the leaves extended 5 cm beyond the X axis to provide a
clear image of the leaf divergence. Note the rounded edges shown in the XZ projection in Figure 4-8.
Figure 4-9: MLC CM shown in the XY plane at the Z=51 cm SSD. The leaves are configured for maximum retraction as would be expected for an un-shaped field. The outer 10 leaf pairs on each side project to 1 cm width at 100 cm SSD, while the inner 40 leaf pairs project to 0.5 cm at 100 cm SSD.
4.2.1.7 AIRSLAB – Air and Exit Window

The final component in the model, this SLAB CM provides an air gap after the MLC component, as well as a 0.3 mm PMMA window representing the end of the accelerator head. The air gap ends at precisely 60 cm SSD, whereupon all of the
simulations performed produce a phase space file to be used in further simulations. At the end of this module, it is expected that the model will have relatively little additional fluence added as all interactions in the accelerator head should reach this level or have been terminated by the variance reduction features of the simulation.

Figure 4-11: Air gap and PMMA window marking the end of the accelerator head. The phase space file output is created at the end of this CM at 60 cm SSD.
4.2.2 Cylindrical Phantom

A cylindrical phantom and treatment couch table top have been simulated using the BEAMnrc package as opposed to the more typical DOSXYZnrc package due to the desire to produce a planar fluence from the particles passing through the phantom/table. This phantom and couch top are intended to function as a patient analog providing a scatter medium to the level of the virtual EPID. The simulation parameters are identical to those of the head model in 4.2.1, with the exception of the source parameter, which is now configured to be a ‘Phase Space Source’ incident on the first CM at 60 cm SSD. This allows us to use the phase space results from a simulation run as configured in 4.2.1 directly, or after manipulating the phase space file as discussed in Chapter 6. The two CM present in this simulation are a SIDETUBE and a SLABS segment. The number of histories run for this simulation is set to the number of particles present in the phase space source file to maintain a direct correlation from the simulation described in 4.2.1 to this simulation.

4.2.2.1 PHANTOM – Cylindrical phantom oriented along X axis

The SIDETUBE CM is used to create a water phantom centered at 100 cm from the source with a radial extent of 15 cm. The phantom is currently modeled as being a solid cylinder of water; however, it may be desirable to include a small air cavity in the very center of the phantom to represent an ion-chamber for dose measurement. In this configuration, the phantom could be used for direct measurement of dose delivered.
4.2.2.2 TABLE – Carbon fiber treatment couch analog

Located directly beneath the bottom of the cylindrical water phantom, a 1 cm thick slab of C17 material is used to represent a carbon fiber tabletop as one finds in most modern linear accelerators. The bottom of this CM is located at 150 cm SSD, where a final phase space output is created at the level of the modeled virtual EPID.

4.2.3 Air Slab

A final simulation was created which consists of only a single slab of material air beginning at 60 cm SSD and extending to 150 cm SSD. The function of this simple model is to provide a simulation environment to appropriately transit the gap from the exit window described in 4.2.1.7 to the level of the virtual EPID. All other simulation parameters are identical to those of 4.2.2.

4.3 Phase space file format

The format of the phase space file is documented in the BEAMnrc manual (2); however, there are some unique aspects we use in this research which are not presented therein. The phase space file is a binary file containing a header and a representation of each particle which has reached the level of the output where the phase space file is generated. The representation of the particle contains information about the energy, charge, position, direction, and some level of interaction history of the particle present at the scoring plane. Each particle is stored in a byte aligned FORTRAN record consisting of a sequence of 4-byte values on the Linux systems discussed in Chapter 3, with seven values for MODE0 simulations, and eight for MODE2. A side effect of this definition is that the length of the header section of the phase space file, and thus the byte location of
the start of the first particle, depends on the mode setting of the file. Understanding the side effects of this method of access is critical to creating programs which can manipulate the phase space files outside of the BEAMnrc code base.

Each particle in the phase space file contains at least a 32 bit unsigned integer associated with the LATCH, and six 32-bit single precision (40) floating-point numbers. The values of interest to us in Chapter 6 are the X and Y positions, as well as the weight which will be unity for all particles, as our simulations have been run without any bremsstrahlung splitting described in 2.2.2.2.

In all of the simulations described in 4.2, the scoring plane where the phase space output is generated is located at the bottom of the last CM in the accelerator stack, so we can expect that our phase space results do not include particles for multiple crossings, thus the weight parameter for all particles will be positive. In the process of working with large simulations, it is occasionally the case that simulations may terminate prematurely, and rather than discard the partial results returned from such premature terminations, a program has been created which can recover the partially recorded phase space file and correct the header for the corrupted values. Similarly, when running a large number of simultaneous simulations (in the case of the cluster described in 3.2.2, 128 simultaneous simulations are common), it becomes cumbersome to combine the results of these simulations manually using the BEAMdp tool provided in the BEAMnrc package; to this end, a program was created to automate the combining of phase space files in a safe, as well as a destructive manner. These ancillary programs are included in Appendix D.
Chapter 5

Virtual Electronic Portal Image

Device (vEPID)

The best method of measuring the beam fluence at the exit of the patient (exit fluence) is through the use of the EPID devices attached to most modern linear accelerators described in 1.1. Previous work by Renner, et al. (1) has demonstrated success in using a deconvolution kernel based on a series of exponential functions to derive fluence from images taken by an EPID in the absence of a patient scattering medium. In order to clinically implement the algorithm developed in this project, it will be necessary in the clinical environment to deconvolve the detector function from the images gathered by the EPID in order to obtain fluence measurements which correspond with the exit fluence from the patient. The deconvolution algorithm described by Renner, et al. has the potential to provide a solution to this problem, but verification that it is applicable in this research is needed.

To this end, the creation of a virtual Electronic Portal Image Device (vEPID) in the Monte Carlo package which can be used to produce ‘images’ from the treatment
delivery is necessary. The ideal environment for this simulation is the DOSXYZnrc (41) package in BEAMnrc as it is designed for the fast calculation of a dose in a phantom, and the EPID in a medical accelerator system acts as a proxy dose detector.

The DOSXYZnrc package allows for great flexibility in the design of the phantom in which dose is deposited, as well as the regions in which dose is scored. Care must be taken in the selection of dose calculation voxels, as attempting to increase the resolution by decreasing the voxel size results in a reduction in the number of particles scored in each voxel, decreasing the statistical accuracy of the simulation. The DOSXYZnrc code calculates the uncertainty in dose for every voxel, and is configured by default to set the dose in voxels with an uncertainty higher than 50% to zero. An appropriate compromise of simulation time, uncertainty, and spatial resolution is critical in producing acceptable results from this code.

The vEPID model is simulated by the creation of a thin layer of material, bounded on all sides by air, and composed of water. The vEPID consists of one voxel layer in the Z direction of thickness 0.3 cm, and three groups of voxels symmetric in the X and Y directions: a 1024x1024 voxel inner region with sides of length 0.04 cm, and a wide outer band to simulate the EPID platform. The simulation source is selected to be a compiled library version of the model described in 4.2.1 above with the input files configured as in Appendix C.

In order to create the required coefficients for the deconvolution kernel, a set of five simulations of varying field size were created using the model presented in chapter 4.2.1 with the addition of 90 cm of air to the bottom of the acceleration column after the PMMA screen. The phase space file resulting from this simulation provides us with both
the fluence at the level of the vEPID, as well as a source which may be delivered to the vEPID described above to generate virtual treatment images. In each simulation, the number of particles present in the phase space source file was used as the basis for the number of histories to run, increasing by a factor of 400 due to the very low interaction probability from the thickness of the vEPID. The field sizes chosen were similar to those presented in Renner, et al., presented here as square sizes at 100 cm SSD: 2x2, 5x5, 10x10, 20x20, and 25x25 cm$^2$. The simulations of the head-air slab stack were configured to run a number of histories in order to obtain dose errors in the vEPID simulation on the order of a few percent. The resulting number of particles in the phase space file, and the average dose error for those pixels receiving at least 20% of the maximum dose delivered are shown in Table 5.1.

<table>
<thead>
<tr>
<th>Field size @ 100cm SSD</th>
<th># of particles in phase-space @ vEPID</th>
<th># of histories run</th>
<th>Average relative error in dose (DA &gt; 0.2 * max(DA))</th>
</tr>
</thead>
<tbody>
<tr>
<td>2x2 cm$^2$</td>
<td>3.1972 * 10$^7$</td>
<td>1.2789 * 10$^{10}$</td>
<td>0.013498</td>
</tr>
<tr>
<td>5x5 cm$^2$</td>
<td>2.2071 * 10$^8$</td>
<td>8.8284 * 10$^{10}$</td>
<td>0.017215</td>
</tr>
<tr>
<td>10x10 cm$^2$</td>
<td>7.3149 * 10$^8$</td>
<td>2.9260 * 10$^{11}$</td>
<td>0.023613</td>
</tr>
<tr>
<td>20x20 cm$^2$</td>
<td>7.0207 * 10$^8$</td>
<td>2.8083 * 10$^{11}$</td>
<td>0.056466</td>
</tr>
<tr>
<td>25x25 cm$^2$</td>
<td>8.0022 * 10$^8$</td>
<td>3.2009 * 10$^{11}$</td>
<td>0.061285</td>
</tr>
</tbody>
</table>

### 5.1 vEPID Detector Deconvolution

The simulation results for open fields are then used to produce the deconvolution kernel parameters by the means of a non-linear problem (NLP) solver and a series of Python programs written for the task. The particulars of the NLP toolkit used are discussed further in Chapter 8 below. To deconvolve the detector function of the vEPID from the image, the algorithm described in Renner, et al. (1) was reproduced in a Python...
module (shown in Appendix E.7) to work with a 3dDose matrix loaded from a completed vEPID simulation in the DOSXYZnrc package.

The data required for the deconvolution parameter fitting must be coalesced into a usable format, and must include all parameters required to renormalize the fluence and computed dose in order to calculate an accurate deconvolution parameter. To accomplish this, a utility program was written (shown in Appendix E.6) which packages the dose and fluence along with the field size and number of histories run. The results are then concatenated into a single file to be used in the parameter fitting code.

\[ K(q) = \sum_{i=1}^{n} a_i \frac{2\pi b_i}{(4\pi^2 q^2 + b_i^2)^{3/2}} \]

Equation 5-1: Point Spread Kernel used in deconvolution.

The deconvolution kernel is described in Equation 5-1, where \( K(q) \) is the value at the frequency radius \( q \) (cycles/cm) taken from the Fourier transform of the vEPID matrix, \( a_i \) and \( b_i \) are the coefficients. The implementation uses the 2D Fast Fourier Transform (FFT) and inverse FFT functions included in SciPy (42) Python library, first computing the vEPID size from the input image, then generating a Fourier space deconvolution kernel following Equation 5-1 to be used in the deconvolution. The source image is then processed according to Equation 5-2.

\[ Fl(x, y) = FFT^{-1} \left[ \frac{FFT(D(x, y))}{K(q(x, y))} \right] \]

Equation 5-2: vEPID Deconvolution function.

5.1.1 Deconvolution Parameter Fitting

In the fitting of the parameters for the deconvolution kernel, the number of exponential terms must be chosen to optimize the quality of the deconvolution weighted against the time to optimize the exponential coefficients. It has previously been shown
that the algorithm produces an acceptable level of accuracy with a five element exponential kernel (1).

\[
\text{Res} = \sum_{FS} \left[ \sum_i \left( f_{FS}^l - c_{FS}^l \right)^2 \right]^{1/2}.
\]

Equation 5-3: Function used to calculate the residual for the current deconvolution kernel parameters.

The deconvolution kernel parameter solver takes the computed vEPID 3ddose matrix, simulated fluence at the vEPID, histories run in BEAMnrc, and histories run in DOSXYZnrc for each simulated open field size as inputs and attempts to minimize the quality of the computed fluence relative to the simulated fluence as described in Equation 5-3 above. Here the initial sum is over the given field sizes \((FS)\), over each element of the images \((i)\) and for the simulated fluence \((f_l)\), and the computed deconvolved fluence \((c_{fl})\). The NLP solver manipulates the coefficients \(a_i\) and \(b_i\) described in Equation 5-1 treating the full set as a flat array of ten variables for the solver package.

The initial guess for the deconvolution parameters was set to the values published in Renner, et al. (1) which provide a set of \(a_i\) and \(b_i\) which are well distributed in magnitude. The solver imposes a constraint on the coefficients that they be >9e-19 to maintain the constraint that \(a_i\) and \(b_i\) be >0 while allowing a reasonable range of values. No constraint is imposed on the upper bounds of the parameters, or on their interrelationship. The solver was allowed to run until the fit quality changed by less than 10^{-3\%}.
The parameter fitting completed after 10651 iterations. The results of the fitting are shown in Table 5.2. With this algorithm, we are able to reconstruct a calculated fluence from an input 10x10 square field vEPID simulation and compare this deconvolved fluence to the originally simulated fluence. We find that, in the region of the image where the fluence is greater than two percent of the maximum fluence, the average difference is -1.12%. We see in Figure 5-1 that the majority of percent difference values are near zero, with a maximum outer peak around -1%, indicating a good quality deconvolution kernel. There is a bimodal distribution of percentage difference represented in the histogram shown which can be attributed to the low dose region surrounding the center intensity. The deconvolution algorithm overestimates fluence in areas outside of the central dose region by 1-2%, which explains the large number of very low dose points around 1-2%; however, the absolute dose difference in these points is very small as they are outside of the irradiation field and so receive around 2% of the central dose.

Table 5.2: Calculated Parameters from Deconvolution Parameter Solver.

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$b_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.86773276*10^7</td>
<td>8.38625193*10^3</td>
</tr>
<tr>
<td>4.16705648*10^7</td>
<td>3.16469550*10^3</td>
</tr>
<tr>
<td>5.20701791*10^7</td>
<td>1.27395524*10^9</td>
</tr>
<tr>
<td>3.39912833*10^13</td>
<td>2.55941836*10^13</td>
</tr>
<tr>
<td>4.47829145*10^14</td>
<td>8.56560594*10^13</td>
</tr>
</tbody>
</table>
Figure 5-1: Histogram of percent difference values for pixels where fluence is greater than 2% of maximum.

Additionally, Figure 5-2 demonstrates that the percent difference between the deconvolved fluence and the simulated fluence is only significant in regions where the fluence is a relatively small.
Figure 5-2: Colormap image of percent difference values for pixels where fluence is greater than 2% of maximum.
Chapter 6

Parameter Space construction

The method proposed for dose reconstruction using exit portal imaging devices is as follows. A model of the accelerator system will be created in the BEAMnrc code to accurately model the accelerator to the Polymethylmethacrylate (PMMA) exit window of the accelerator. At this point, the model will produce a phase space file containing the position, energy, weight, particle type, and direction cosines of each particle arriving at this plane. The format of the phase space file is described in the BEAMnrc documentation (2). It notes that in addition to the previously mentioned properties, each particle also has a 32 bit unsigned integer allocated for LATCH-bits. The LATCH bits are used in the BEAMnrc software to indicate regions in which the particle has interacted in the model.

As described in the documentation: bit 0 stores information about the interaction history of the particle (if set, the particle has bremsstrahlung or positron annihilation events in its history); bits 29-30 store the charge of the particle; bit 31 records if the particle has crossed this scoring plane in its history; and the remaining bits are for interaction history. Of bits 1-28, bits 1-23 are for latching on specific regions, and bits 24-28 are used to store the region in which a particle was created (as a 4bit unsigned
integer). The system associates all regions not associated by a user to bit 23, thus we should avoid its use as it may be set by the simulation. This grants us bits 1-22 for storing LATCH information, which the system typically considers as 22 distinct locations. In order to store additional information for each particle, we break these 22 bits into two 10 bit regions, with 2 extra bits which currently are reserved for future use. Bits 2-11 are used to store the X position of the particle, bits 12-21 store the Y position. If the particle overflows or underflows these 1024 regions, all bits 2-21 are set to zero and bit 1 is set to one, otherwise the X and Y bits are used to store a 10-bit integer containing the position of the particle.

By storing the position information in the LATCH bits, and using the correct simulation parameters, we can track the dose deposition from a particle origination. This allows us to create a parameter space for convolution. A program was written to read in a computed phase space and set the LATCH bits for each particle as appropriate in order to provide a grid of acceptable resolution at the level of the EPID. This software can be extended to a module for BEAMnrc to set the LATCH bits as the particle runs through the simulation.

The output of this latched phase space file may then be used in a simulation passing through the patient/table system resulting in a new phase space output at the level of the EPID. This output may then be parsed to produce an ‘image’ of the fluence at the level of the EPID. This image may be broken down into a single image per pixel defined in the LATCH bits of the fluence. This allows us to create an image for each pixel in the input fluence. Each image is a 1024x1024-pixel grid representing particle density with a 64bit floating point number, resulting in each image requiring at least 8MB of storage.
As the incoming fluence is also broken into a 1024x1024 grid, to represent the full parameter space would require approximately 8TB of storage. This would prove prohibitive even by modern computer system standards, and would limit the speed of calculation to the speed of disk access, which is commonly the slowest component in the modern computer system.

\[
2 \cdot n\text{nnz} + (n + 1) < n^2 \Rightarrow n\text{nnz} < \frac{n^2 - n - 1}{2}
\]

Equation 6-1: Inequality describing the point at which a CSR stored matrix requires less space than a square dense matrix of shape n by n, and \(n\text{nnz}\) is the number of non-zero elements.

To reduce this storage requirement, two methods have been employed. The first method is a lossless compression method using sparse matrix representation (42) to store the 1024x1024 matrix of the output image. This allows us to store only the portions of the output image containing non-zero values; as the output is well constrained to immediate vicinity of the input fluence pixel, this results in significant storage savings. As the space required to store data in the CSR format is two arrays of length equal to the number of non-zero entries, and one array equal to the number of rows plus one, if the sparse matrix meets the inequality expressed in Equation 6-1, a sparse matrix will consume less space than its dense matrix counterpart. An analysis of the fluence matrices produced in a simulation of a 10x10 field demonstrates the level of density – defined as the ratio of the number of nonzero entries to the total number of entries for the corresponding dense matrix – obtained (cf. Figure 6-1). Adapting Equation 6-1 into evaluation of matrix density, Equation 6-2, we can see that for our 1024x1024 element matrices, as long as we are able to attain a density of 49.95%, or less than 523776 elements, the CSR format will save space.
\[ S = \frac{nnz}{n^2} < \frac{1}{2} \left[ 1 - \frac{1}{n} - \frac{1}{n^2} \right] \]

Equation 6-2: Definition of density, and restatement of Equation 6-1 in terms of density.

Figure 6-1: Histogram of density of parameter space data for a resample dataset of 32x32 arrays.

The two formats used are Coördinate (COO) form, and Compressed Sparse Row (CSR) format. The coordinate form uses three arrays to store the row, column, and value for each nonempty cell in the matrix. This form is ideal for constructing sparse matrices as one may append entries to the end of the list with minimal additional calculation (the only costly operation is growing the memory allocation). The alternate form is the CSR format which also allocates three arrays: the first array contains the number of rows + 1 elements which indicate the start and end position in the following two arrays of each row, that is the data for the first row is stored in the next two arrays in index row_ptr[0] to (row_ptr[1] – 1); the next array is the column array which indicates the column of the data; and the final array stores the values. This format is significantly slower to add entries so the optimal process would be to build arrays using the COO format, and save
them using the CSR format. As conversion between COO and CSR is an O(n) operation, this is the best of both worlds.

The second method employed to reduce storage requirements of the parameter space is to reduce the resolution of the stored parameter space while maintaining full resolution of each stored exit fluence. To this end, a selected number of the lower order bits of the LATCH bit in X and Y are dropped, reducing the 1024x1024 parameter space to a more manageable size. This method is a lossy compression method, but does result in significant storage savings (a factor of 4 reduction for every bit dropped from X and Y). This reduction is in somewhat of a competition with the use of sparse matrices as condensing the parameter space will decrease the density of the resultant matrices; however, as has been shown in Figure 6-1 for a 32x32 parameter space, and in Figure 6-2 for a 128x128 parameter space, the density obtained is still sufficient for advantage in storage space.

![Figure 6-2: Histogram of array density for 128x128 grid parameter space.](image-url)
Using these parameter space files we may produce an output fluence for any given input fluence in the simulated region. This will be used in our algorithm for the reconstruction of entrance fluence from exit fluence.
Chapter 7

Fluence Calculation

The algorithm for calculating the output fluence from a given input fluence estimate involves the use of the parameter space created as described in section Chapter 6 above. Equation 7-1, below, describes how the computed fluence ($f^c_l$) is computed from a summation over the entire parameter space with weights ($K_{xy}$). As the parameter space weight array is allowed to be of a higher resolution than the parameter space, an interpolation method is needed to compute the resultant fluence.

The algorithm designed to interpolate from a higher resolution entrance fluence to a lower resolution parameter space involves treating each parameter space component as a separate mutable kernel which will be shifted to align with the entrance fluence position. The appropriate kernel is selected for each pixel in the entrance fluence by selecting the nearest parameter space array to the x and y position of the pixel to be calculated. The result is then shifted in the computed fluence plane by the ratio of pixel sizes times the offset from the center of the parameter space position of the kernel. This results in each input pixel of an arbitrary resolution entrance fluence producing an appropriate output at the exit fluence plane.
\[ f_{x'y'}^{K}(K) = \sum_{x'y'} K_{x'y'} \cdot X Y A r r_{x'y'} \bigg|_{x'y'} \]

Equation 7-1: Exit fluence is calculated from the parameter space weights.
Chapter 8

Fluence Solver

The method described here to solve for the entrance fluence given an exit fluence depends on the parameter space described in section Chapter 3 above. This parameter space is used to calculate the exit fluence from a given entrance fluence as described in section Chapter 7 above. This exit fluence is then used as the calculated fluence at the level of the EPID. The difficulty then becomes determining the appropriate weights of the input matrix to produce a calculated fluence which matches with the measured (or simulated) fluence at the level of the EPID.

To solve for the entrance fluence, we will create a function to quantify the quality of the match, and use an iterative minimization solver to find the set of coefficients that minimize the quality function. The quality function we have decided upon is the square root of the sum of the square of the residuals of the two fluence matrices (Equation 8-1, below); where \( f_l(x, y) \) is the input fluence at position \((x, y)\); and, \( f_{lc}(x, y) \) is the calculated fluence at position \((x, y)\).

\[
R(K)^2 = \sum_{x,y} \left( f_{xy} - f_{xy}^c(K_{xy}) \right)^2
\]

Equation 8-1: Residual used in the calculation of match quality.
The solver used is from a package called OpenOpt, a software package for Python (43) which has collected several solver packages together, including some packages written by the OpenOpt maintainers (44). The package selected for use in this project was the Generalized epsilon-subgradient (gsubg) algorithm, part of the OpenOpt package implementing sub-solvers to seek local minima of the function. The ‘gsubg’ solver was selected because it is one of the few large scale NLP solvers available in the OpenOpt package which allows one to specify a derivative function. The ability to supply a user defined function for defining the gradient of the problem function allows us to significantly improve on calculation speed over a blind approach provided in the OpenOpt package.

8.1 Derivative calculation function

The derivative function is a finite difference calculation function designed to accept any user defined stencil for numerical analysis of the derivative. At current, three stencils have been implemented: a Newtonian quotient (44), a central two point stencil (44), and a five point stencil (45); requiring 1, 2, and 4 additional calculations of the match quality respectively. For the highest accuracy, a default choice of the five point stencil was used, though evaluation of the need for this may be worthwhile. In order to speed calculation, the input fluence minus the previous full calculated fluence along with the current coefficients are provided to each worker process. The calculation is then performed according to Equation 8-2 below, where: $f_{Mpcfl}$ is the input fluence minus the previous full calculated fluence; and $f_{c,S_y}$ is the differential fluence for the current $x,y$ in the coefficient space with the addition of the stencil difference; $S_y$ is the stencil difference.
addition; and $S_m$ is the stencil weight, or multiplier. The advantage of this method is that we may calculate each $x, y, S_m, S_r$ position separately, allowing us to massively parallelize the calculation and sum the components at the end.

$$\frac{dR_{xy}}{dK_{xy}}(K_{xy}) = \sum_{\text{stencil}} S_m \cdot \sum_{x',y'} \left[ (f^{Mpcf}_x f^{K_y} - f^{c}_x (S_{rxy}))^2 \right]$$

Equation 8-2: Prototype function used in derivative computation.

The optimization program is then set up as shown in Appendix F.6. Upon launching, the program determines if it is the master of a series of nodes, or running alone; or if it is a node; it then branches into the code accordingly. With the appropriate configuration this allows us to use the PBS described in section 3.3 above to assign nodes to the job. The primary difference between the program flow of the master and the nodes involves the initial setup. The master node will read in the fluence, and set up the initial guess.

### 8.2 Initial Guess Calculation

To enhance convergence, an initial entrance fluence is calculated by recasting the provided exit fluence onto a grid the size of the parameter space and used as the first guess. This first guess is used to create a computed exit fluence, then computing a normalization factor for the computed exit fluence to the provided exit fluence. As the normalization factor and fluence are all positive real numbers, we may safely apply the normalization factor to the initial guess to provide a starting point for the optimizer (c.f., Equation 7-1, Equation 8-3).
\[ fl' (c \cdot K) = \sum_{x',y'} c \cdot K_{x'y'} \cdot [XY_{Arr}]_{x'y'} = c \cdot \sum_{x',y'} K_{x'y'} \cdot [XY_{Arr}]_{x'y'} = c \cdot fl' (K). \]

Equation 8-3: Derivation of appropriateness of initial guess normalization.

8.3 Fluence Solver Program Design

The program is constructed in Python and is designed to take advantage of the PBS batch system to run in parallel, or to run independently on one system. When the program is launched it looks for several environment variables set by the batch system to determine if it is being run in a batch job, and if so, what nodes are under its control. The PBS batch system sets the environment variable ‘PBS\_ENVIRONMENT’ to ‘PBS\_BATCH’, creates a unique job cookie which is stored in ‘PBS\_JOBCOOKIE’, and creates a file containing this list of nodes allocated for the job whose location is stored in ‘PBS\_NODEFILE’. If ‘PBS\_ENVIRONMENT’ is not set to ‘PBS\_BATCH’, the program determines whether it is the master or the node based on command line switches.

The program is designed with the intent of being launched once on each node, it will then run one processing thread for each processor it has been granted on the node. If there is only one processor available, it does not spawn subthreads but instead performs all calculations from within the master thread.

The general form of the solver program is similar to the form of the fluence calculation program described in Chapter 7, wherein each node involved in the calculation creates a set of queues to pass information regarding what values need to be computed, and the status of those computations. When completed, shared memory arrays are updated by each process and the master thread combines these into a single completed array of data.
This master/client thread form is mirrored in the handling of multiple calculation nodes. If multiple nodes are present, the master node will prepare a set of queues capable of transmitting requests over TCP/IP to each client. Each node performs the calculations assigned to it by the master, then packages up the calculations it has performed and relays them to the master node over a TCP/IP connection. The sessions are secured (though not encrypted) through the use of the ‘PBS_JOBCOOKIE’ obtained during master startup.

It is the function of the master node to set up the problem to be solved, decide on what requires calculation, distribute the jobs, collect the results, and decide on the next step. The same optimization package described in 5.1.1, namely OpenOpt, is used for the optimization in this program. The function to be minimized is the sum of the squares of the differences between the computed fluence grid at the level of the EPID and the deconvolved fluence grid obtained via measurement; in this case, the measurement is from the simulation described in Chapter 7.

The distributed nodes are tasked with calculating the components of the derivative functions at each point in the planned grid. The optimizer then determines which variables will be modified and proceeds through a step in the optimization. The calculation of step direction is part of the OpenOPT package which is not written for parallelization and so cannot take advantage of the cluster environment. For this reason the calculation times are largely dominated by the costly step direction calculation and limit the efficiency of the program in a large cluster.

The optimizer function in use us the ALGENCAN optimizer from the OpenOPT package, which is the only solver capable of handling the very large number of variables
used in the computation (namely, each pixel in the solved fluence is considered an independent variable for the solver).
Chapter 9

Results

To verify the functionality of the fluence solver program described in Chapter 8, two sample IMRT fields were delivered onto the vEPID phantom. The two fields were designed based on contours in the Pinnacle3 Treatment Planning System (Philips), using a machine model of a Varian iX with a Millenium120 MLC. A series of IMRT optimization sequences were run designed to produce a unique intensity map based on the contoured structures. The results of the optimization were exported to the MOSAIQ record and verify system (Elekta Oncology Systems) to convert the beam parameters into a simple RTP file format. A careful analysis of the format yielded a translation between the format found in the RTP file, and a format usable by the BEAMnrc DYNVMLC module described in 4.2.1.6. A program was written to automate this translation and simplify the process of generating sample fluence patterns.

The resulting DYNVMLC modulation files were used in the BEAMnrc simulation described in 4.2.1 to produce a phase space file at the exit window of the accelerator. This phase space was binned using a modified version of ‘bin_fluence.py’ designed to produce an appropriate histogram set for 60cm from the source. The phase space was then used as a source for the simulation described in
4.2.2 to produce a phase space at the vEPID plane. This phase space was then binned using the `bin_fluence.py` program shown in E.3 to produce a fluence at the exit level of the vEPID which will be used in the fluence convolution solver.

Each solver was allowed to run for until the change in match quality was less than 0.1%. The average run time was 14.4 minutes, using a total of 862.3 CPU minutes to complete. Some overhead in the method of calculation results in an average of 46% of the CPU run time to be dominated by a single process on the master node, resulting in the deviation from the expected total CPU time of 1843.2 minutes from 14.4 minutes of calculation time and 128 active nodes.

The results of the fluence solver were then analyzed by comparing to the binned fluence obtained from `bin_fluence_60.py` to determine the capability of the system to reconstruct a known entrance fluence from a given exit fluence. The deviation histograms in percent deviation from planned fluence are shown in Figure 9-1, and Figure 9-2 for the two IMRT fields. The criterion for evaluation of the match is based on the recommendations of the AAPM Task Group 119, which suggests that 90% of points greater than the threshold value of 10% be within 3% dose or 3mm distance (DTA of 90%,3%,3mm) (47) (48). The first IMRT field was found to pass with a simple percentage difference of 94.5% of points within 1%. The second IMRT field was found to have 97.9% of points within 1%.
Figure 9-1: Histogram of percent difference for the first IMRT field

Figure 9-2: Histogram of percent difference for the second IMRT field
A visual review of the quality of the match may be gained in the side by side comparisons shown in Figure 9-3, and Figure 9-4 where the fluence obtained during the simulation is presented on the left side and the calculated fluence is presented on the right. Finally, we present a visual representation of the percentage difference between the images in Figure 9-5, and Figure 9-6.

Figure 9-3: Visual comparison of entrance fluence for the first IMRT field. (Left: planned fluence, Right: computed fluence).

Figure 9-4: Visual comparison of entrance fluence for the second IMRT field. (Left: planned fluence, Right: computed fluence).
Figure 9-5: Plot of percentage difference between simulated and computed entrance fluence for the first IMRT field.

Figure 9-6: Plot of percentage difference between simulated and computed entrance fluence for the second IMRT field
Chapter 10

Conclusion

From these results it is clear that the software package as designed may be used to simulate a quality assurance phantom and produce a parameter space which is capable of determining a fluence exiting the machine from measured convolved EPID images. With this package the process of verifying the delivered fluence from the machine may be conducted using the EPID device to measure the exit fluence from a QA phantom.

We have demonstrated that this algorithm may reproduce an entrance fluence when given an exit fluence and no information about the design of the entrance radiation field. This has been shown to greatly exceed the quality criteria established by the AAPM TG 119 for the verification of IMRT delivery for these example cases, indicating that it can be a very sensitive test for evaluation of plan quality and deliverability. Both test patterns exhibit greater than 94% of points with fluence greater than 10% of the maximum fluence passing within 1% of the expected dose. Finally, we have demonstrated that the fluence solver may calculate an entrance fluence within a reasonable period of time and that the solution appears to converge well for highly modulated fields.
Additionally, with the use of a treatment time acquired CT of the patient, one could adapt this program to calculate entrance fluence on the patient from images collected at the time of treatment. The introduction of the patient CT into the beam path could be used to produce a set of patient and beam specific parameter spaces to be used in the fluence solver module. This would allow for post treatment verification of dose delivered to the patient for each fraction of treatment.
References


8. Physical and dosimetric aspects of a multileaf collimation system used in the dynamic mode for implementing intensity modulated radiotherapy. LoSasso, Thomas,


16. American Society For Therapeutic Radiology and Oncology (ASTRO) and American College of Radiology (ACR) Practice Guidelines for Intensity-Modulated


19. **BEAM: A Monte Carlo code to simulate radiotherapy treatment units.**


22. **Carrol, Mike and Parsai, E Ishmael.** Project.


39. *Monte Carlo dose calculations in quality assurance for IMRT of head and neck cancers.* Tang, Nin-fai Francis and 鄧年輝. s.l. : University of Hong Kong (Pokfulam Road, Hong Kong), 2008, HKU Theses Online (HKUTO).


42. *Jones, Eric, et al., et al. SciPy: Open source scientific tools for Python. 2001-.*


47. AAPM IMRT QA task group report.


Appendix A

Live-Build Customizations

The following programs and modifications were created to simplify deployment of the cluster to all nodes.

A.1 auto/build

```
#!/bin/sh
lb build noauto "$@" 2>&1 | tee binary.log
```

A.2 auto/config

```
#!/bin/sh
mkdir -p config
cp -rf auto/chroot_* config/
cp -rf auto/archives config/

lb config noauto \
    --debian-installer=false --distribution=squeeze --linux-packages=linux-image-2.6.32-5 - \n    --linux-flavours=686' \n    --mirror-bootstrap=http://mirror.steadfast.net/debian/ --language=en --cache- \n    packages='true' --cache-stages='none' \n    --archive-areas=main contrib non-free --packages-lists "blade_live" \n    --binary-indices false --bootappend-live="live-config.noautologin live- \n    config.timezone=America/New_York ipv6.disable=1 toram" \n    --binary-images='net' --net-root-filesystem=nfs --net-root-path='/export/debian_x86' --net- \n    root-server='192.168.0.1' \n    "$@"

cp -rf auto/chroot_local-includes config/
```

A.3 auto/clean

```
#!/bin/sh
```
A.4 auto/chroot_local-preseed/nis.cfg

nis nis/domain  string torqueserver

A.5 auto/chroot_local-packagelists/blade_live.lst

# Include only minimal, then add in everything but locales from standard
eject file user-setup
console-setup console-common kbd

firmware-bnx2 lm-sensors autofs nis
# libcr0 blcr-dkms blcr-util
torque-client torque-mom
v66d ntpdate screen
openssh-server libx11-6 mpi-default-dev mpi-default-bin libgfortran3
bzip2 acpid ganglia-monitor lbzip2 pigz python-openopt python-scipy python-setproctitle
mingetty less

A.6 auto/chroot_local-

includes/etc/ganglia/conf.d/hpsmcli.pyconf

```python
modules {
    module {
        name = "hpsmcli"
        language = "python"
    }
}

collection_group {
    collect_every = 10
    time_threshold = 50
    metric {
        name_match = "\.(.*)-(CPU\.*\.)_Temperature"
        title = "\2 \1 Temperature"
        value_threshold = 20
    }
    metric {
        name_match = "\.(.*)-PROCESSOR.*_Temperature"
        title = "Processor Zone \1 Temperature"
        value_threshold = 20
    }
    metric {
        name_match = "\.(.*)-\^[CP].*\._Temperature"
        title = "\2 Temperature"
        value_threshold = 20
    }
    metric {
        name_match = "Fan_.*"
        title = "Fan Zone \1"
        value_threshold = 10
    }
    metric {
        name_match = "Power_.*"
```
A.7 auto/chroot_local-

includes/etc/ganglia/conf.d/modpython.conf

def
value_threshold = 100
}
}

A.8 auto/chroot_local-includes/etc/ganglia/gmond.conf

/* This configuration is as close to 2.5.x default behavior as possible
The values closely match ./gmond/metric.h definitions in 2.5.x */
globals {
daemonize = yes
setuid = yes
user = ganglia
debug_level = 0
max_udp_msg_len = 1472
mute = no
deaf = no
host_dmax = 0 /*secs */
cleanup_threshold = 300 /*secs */
gexec = no
send_metadata_interval = 0
}

/* If a cluster attribute is specified, then all gmond hosts are wrapped inside
* of a <CLUSTER> tag. If you do not specify a cluster tag, then all <HOSTS> will
* NOT be wrapped inside of a <CLUSTER> tag. */
cluster {
name = "Blade Cluster"
owner = "UTMC"
latlong = "41.6N 83.6W"
url = "http://torqueserver/ganglia"
}

/* The host section describes attributes of the host, like the location */
host {
location = "2,1,__BLADE__"
}

/* Feel free to specify as many udp_send_channels as you like. Gmond
used to only support having a single channel */
udp_send_channel {
 mcast_join = 239.2.11.71
 port = 8649
 ttl = 1
}

/* You can specify as many udp_recv_channels as you like as well. */
udp_recv_channel {
mcast_join = 239.2.11.71
port = 8649
bind = 239.2.11.71

/* You can specify as many tcp_accept_channels as you like to share
an xml description of the state of the cluster */
tcp_accept_channel {
    port = 8649
}

/* Each metrics module that is referenced by gmond must be specified and
loaded. If the module has been statically linked with gmond, it does not
require a load path. However all dynamically loadable modules must include
a load path. */
modules {
    module {
        name = "core_metrics"
    }
    module {
        name = "cpu_module"
        path = "/usr/lib/ganglia/modcpu.so"
    }
    module {
        name = "load_module"
        path = "/usr/lib/ganglia/modload.so"
    }
    module {
        name = "mem_module"
        path = "/usr/lib/ganglia/modmem.so"
    }
    module {
        name = "net_module"
        path = "/usr/lib/ganglia/modnet.so"
    }
    module {
        name = "proc_module"
        path = "/usr/lib/ganglia/modproc.so"
    }
    module {
        name = "sys_module"
        path = "/usr/lib/ganglia/modsys.so"
    }
}
include ('/etc/ganglia/conf.d/*.conf')

/* The old internal 2.5.x metric array has been replaced by the following
collection_group directives. What follows is the default behavior for
collecting and sending metrics that is as close to 2.5.x behavior as
possible. */

collection_group {
    collect_once = yes
time_threshold = 20
metric {
    name = "heartbeat"
}
}

collection_group {
    collect_once = yes
time_threshold = 1200
metric {
    name = "cpu_num"
}
title = "CPU Count"
}
metric {
    name = "cpu_speed"
    title = "CPU Speed"
}
metric {
    name = "mem_total"
    title = "Memory Total"
}/** Should this be here? Swap can be added/removed between reboots. */
metric {
    name = "swap_total"
    title = "Swap Space Total"
}
metric {
    name = "boottime"
    title = "Last Boot Time"
}
metric {
    name = "machine_type"
    title = "Machine Type"
}
metric {
    name = "os_name"
    title = "Operating System"
}
metric {
    name = "os_release"
    title = "Operating System Release"
}
metric {
    name = "location"
    title = "Location"
}

/* This collection group will send the status of gexecd for this host every 300 secs */
/* Unlike 2.5.x the default behavior is to report gexecd OFF. */
collection_group {
    collect_once = yes
    time_threshold = 300
    metric {
        name = "gexec"
        title = "Gexec Status"
    }
}

/* This collection group will collect the CPU status info every 20 secs.
The time threshold is set to 90 seconds. In honesty, this time_threshold could be
set significantly higher to reduce unnecessary network chatter. */
collection_group {
    collect_every = 20
    time_threshold = 90
    /* CPU status */
    metric {
        name = "cpu_user"
        value_threshold = "1.0"
        title = "CPU User"
    }
    metric {
        name = "cpu_system"
        value_threshold = "1.0"
        title = "CPU System"
    }
    metric {
        name = "cpu_idle"
        value_threshold = "5.0"
        title = "CPU Idle"
    }
    metric {

name = "cpu_nice"
value_threshold = "1.0"
title = "CPU Nice"
}

metric {
name = "cpu_aidle"
value_threshold = "5.0"
title = "CPU aidle"
}

metric {
name = "cpu_wio"
value_threshold = "1.0"
title = "CPU wio"
}

/* The next two metrics are optional if you want more detail...
... since they are accounted for in cpu_system.

metric {
name = "cpu_intr"
value_threshold = "1.0"
title = "CPU intr"
}

metric {
name = "cpu_sintr"
value_threshold = "1.0"
title = "CPU sintr"
*/

collection_group {
collect_every = 20
time_threshold = 90
/* Load Averages */
metric {
name = "load_one"
value_threshold = "1.0"
title = "One Minute Load Average"
}
metric {
name = "load_five"
value_threshold = "1.0"
title = "Five Minute Load Average"
}
metric {
name = "load_fifteen"
value_threshold = "1.0"
title = "Fifteen Minute Load Average"
}
}

/* This group collects the number of running and total processes */
collection_group {
collect_every = 80
time_threshold = 950
metric {
name = "proc_run"
value_threshold = "1.0"
title = "Total Running Processes"
}
metric {
name = "proc_total"
value_threshold = "1.0"
title = "Total Processes"
}
}

/* This collection group grabs the volatile memory metrics every 40 secs and
sends them at least every 180 secs. This time_threshold can be increased
significantly to reduce unneeded network traffic. */
collection_group {
collect_every = 40
time_threshold = 180
metric {
    name = "mem_free"
    value_threshold = "1024.0"
    title = "Free Memory"
}
metric {
    name = "mem_shared"
    value_threshold = "1024.0"
    title = "Shared Memory"
}
metric {
    name = "mem_buffers"
    value_threshold = "1024.0"
    title = "Memory Buffers"
}
metric {
    name = "mem_cached"
    value_threshold = "1024.0"
    title = "Cached Memory"
}
metric {
    name = "swap_free"
    value_threshold = "1024.0"
    title = "Free Swap Space"
}
}
collection_group {
    collect_every = 40
    time_threshold = 300
    metric {
        name = "bytes_out"
        value_threshold = 4096
        title = "Bytes Sent"
    }
    metric {
        name = "bytes_in"
        value_threshold = 4096
        title = "Bytes Received"
    }
    metric {
        name = "pkts_in"
        value_threshold = 256
        title = "Packets Received"
    }
    metric {
        name = "pkts_out"
        value_threshold = 256
        title = "Packets Sent"
    }
}

A.9 auto/chroot_local-includes/etc/init.d/nfsswap

#!/bin/sh
### BEGIN INIT INFO
# Provides:          nfsswap
# Required-Start:    $remote_fs $local_fs $named $network $time
# Required-Stop:     $remote_fs $local_fs $named $network
# Default-Start:     1 2 3 4 5
# Default-Stop:      0 6
# Short-Description: Start the NFS SWAP
# Description:       Creates a swap on an NFS mount
### END INIT INFO

PATH=/usr/sbin:/usr/bin:/sbin:/bin
NAME=nfsswap
DESC="NFS Swap"
PIDFILE=/var/spool/torque/server_priv/server.lock

test -x /etc/init.d/nfsswap || exit 0

# Include torque defaults if available
if [ -f /etc/default/nfsswap ]; then
  . /etc/default/nfsswap
fi

# Load lsb functions
./lib/lsb/init-functions

case "$1" in
start)
  if [ -f /tmp/nfsswap.dev ]; then
    restart
  else
    log_daemon_msg "Starting $DESC"
    log_progress_msg "$NAME"
    LOOPDEV=$(losetup -f --show ${SWAPFILE:-/nfsswap/nfsswap}) || log_end_msg $?
    echo "$LOOPDEV" >> /tmp/nfsswap.dev
    swapon "$LOOPDEV"
    log_end_msg $?
    fi
  ;;
stop)
  log_daemon_msg "Stopping $DESC" "$NAME"
  if [ -f /tmp/nfsswap.dev ]; then
    LOOPDEV=`cat /tmp/nfsswap.dev`
    swapoff "$LOOPDEV"
    losetup -d "$LOOPDEV"
    log_end_msg $?
  else
    log_progress_msg "$NAME not running"
    log_end_msg 1
  ;;
restart)
  # stop && sleep 2 && # start
  ;;
  *)
N=/etc/init.d/`basename $0`
  echo "Usage: $N {start|stop|restart|reload|force-reload}" >&2
  exit 1
  ;;
esac
exit 0

A.10 auto/chroot_local-includes/lib/live/config/001-hostname

#!/bin/sh
## live-config(7) - System Configuration Scripts
## Copyright (C) 2006-2011 Daniel Baumann <daniel@debian.org>
## live-config comes with ABSOLUTELY NO WARRANTY; for details see COPYING.
## This is free software, and you are welcome to redistribute it
## under certain conditions; see COPYING for details.
Hostname ()
{
    # Checking if package is installed or already configured
    if [ -z "${LIVE_HOSTNAME}" ] || \
       [ -e /var/lib/live/config/hostname ]
    then
        return
    fi

echo -n " hostname"

Configure_hostname
}

Configure_hostname ()
{
    # Change hostname only if it is not set
    if [ ! -e /etc/hostname ] || grep -qs localhost\.localdomain /etc/hostname
    then
        LIVE_HOSTNAME=$(awk '/host/ { print $3 }' /var/log/netboot.config)
        LIVE_ROOTSERV=$(awk '/rootserver/ { print $2 }' /var/log/netboot.config)
        echo "${LIVE_HOSTNAME}" > /etc/hostname
        else
        LIVE_HOSTNAME="$(cat /etc/hostname)"
    fi

    # Create /etc/hosts only if it is not present or empty
    if [ ! -s /etc/hosts ]
    then
        cat > /etc/hosts
        echo "127.0.0.1       localhost ${LIVE_HOSTNAME}" >> /etc/hosts
        echo "::1           localhost ip6-localhost ip6-loopback" >> /etc/hosts
        echo "fe00::0        ip6-localnet" >> /etc/hosts
        echo "ff00::0        ip6-mcastprefix" >> /etc/hosts
        echo "ff02::1        ip6-allnodes" >> /etc/hosts
        echo "ff02::2        ip6-allrouters" >> /etc/hosts
        echo "${LIVE_ROOTSERV} torqueserver" >> /etc/hosts
        echo "localhost" >> /etc/hosts
        fi

    hostname "$(LIVE_HOSTNAME)"

    sed -i "s/__BLADE__/$(hostname | cut -b 6-)/" /etc/ganglia/gmond.conf

date --set="$(ssh torqueserver date)"

    # Creating state file
    touch /var/lib/live/config/hostname
}

A.11 auto/chroot_local-

includes/usr/lib/ganglia/python_modules/hpsmcli.py
## Example Output of "show temp"

temp_example_output =
```
Sensor   Location              Temp       Threshold
------   --------              ----       ---------
#1        SYSTEM_BD            45C/113F   80C/176F
#2        PROCESSOR_ZONE       38C/100F   65C/149F
#3        CPU#1                37C/98F    95C/203F
#4        CPU#1                37C/98F    95C/203F
#5        PROCESSOR_ZONE       35C/95F    70C/158F
#6        CPU#2                37C/98F    95C/203F
#7        CPU#2                37C/98F    95C/203F
#8        MEMORY_BD            53C/127F   85C/185F
#9        AMBIENT              23C/73F    38C/100F
```

## Example Output of show fans:
fans_example_output =
```
Fan  Location        Present Speed  of max  Redundant  Partner  Hot-pluggable
---  --------        ------- -----  ------  ---------  -------  -------------
#1   VIRTUAL         Yes     NORMAL  25%     N/A        N/A      No
```

## Example Output of show power:
powr_example_output =
```
Power Meter #1
Power Reading : 128
```

```python
from time import time
from subprocess import Popen, PIPE

descriptors = list()

last_run_fans = 0
last_run_temp = 0
last_run_powr = 0

fans_list = dict()
temp_list = dict()
powr_list = dict()
time_max = 10
```
fans_cmd = "show fans"
temp_cmd = "show temp"
powr_cmd = "show powermeter"
sudo_cmd = "/usr/bin/sudo"
hpasmcli_cmd = "/sbin/hpasmcli"

def metric_handle_fans(name):
    global last_run_fans
    global fans_list
    global time_max

    the_time = time()

    # If it has been more than time_max seconds since we last
    # collected sensor data, do it again, otherwise use stale data.
    if (abs(the_time - last_run_fans) > time_max):
        last_run_fans = the_time
        try:
            # Run sensors and pull out the right data (with 'fan')
            output = Popen([sudo_cmd, hpasmcli_cmd, '-s', fans_cmd], stdout=PIPE).communicate()[0]

            for line in output.splitlines():
                if '#' in line:
                    num=line.split()[0]
                    sensor_id='Fan_'+num
                    temp=float(line.split()[4][:-1])
                    fans_list[sensor_id.replace('#','')] = temp

        except OSError as e:
            print "Some error occurred collecting the sensor data."

    # At this point we should have the sensor data. Return just the value we need.
    if name in fans_list:
        return (fans_list[name])
    else:
        return None

def metric_handle_temp(name):
    global last_run_temp
    global temp_list
    global time_max

    the_time = time()

    # If it has been more than time_max seconds since we last
    # collected sensor data, do it again, otherwise use stale data.
    if (abs(the_time - last_run_temp) > time_max):
        last_run_temp = the_time
        try:
            # Run sensors and pull out the right data (with 'temp')
            output = Popen([sudo_cmd, hpasmcli_cmd, '-s', temp_cmd], stdout=PIPE).communicate()[0]

            temps = list()
            for line in output.splitlines():
                if '#' in line:
                    split_line=line.split()
                    sensor_num=split_line[0]
                    sensor_location=split_line[1]
                    sensor_id=sensor_num+'-'+sensor_location+' Temperature'
                    temp=float(split_line[2].split('/')[0][:-1])
                    temp_list[sensor_id.replace('#','')] = temp

            except OSError as e:
                print "Some error occurred collecting the sensor data."
# At this point we should have the sensor data. Return just the value we need.
if name in temp_list:
    return (temp_list[name])
else:
    return None

def metric_handle_powr(name):
    global last_run_powr
    global powr_list
    global time_max
    the_time = time()
    # If it has been more than time_max seconds since we last
    # collected sensor data, do it again, otherwise use stale data.
    if (abs(the_time - last_run_powr) > time_max):
        last_run_powr = the_time
    try:
        # Run sensors and pull out the right data (with 'powr')
        output = Popen([sudo_cmd, hpasmcli_cmd, '-s', powr_cmd], stdout=PIPE).communicate()[0]
        power_number = ""
        for line in output.splitlines():
            if '#' in line:
                continue
            if power_number:
                sensor_id = 'Power_' + power_number
                powr = float(line.split()[3])
                powr_list[sensor_id.replace('#','')] = powr
                power_number = ""
        except OSError as e:
            print "Some error occured collecting the sensor data."
    except Exception as e:
        print "Some error occured collecting the sensor data."
    # At this point we should have the sensor data. Return just the value we need.
    if name in powr_list:
        return (powr_list[name])
    else:
        return None

def get_sensors(time_max):
    """Gets the full set of sensors for input into descriptors."""
    global descriptors
    # Run the hpasmcli command, and if it fails, use the default output values.
    # This is needed because, when the system first starts, the gmond finishes launching
    # before the drivers needed to read this information.
    fans_running = Popen([sudo_cmd, hpasmcli_cmd, '-s', fans_cmd], stdout=PIPE)
    fans_output = fans_running.communicate()[0]
    if fans_running.returncode != 0:
        fans_output = fans_example_output
    temp_running = Popen([sudo_cmd, hpasmcli_cmd, '-s', temp_cmd], stdout=PIPE)
    temp_output = temp_running.communicate()[0]
    if temp_running.returncode != 0:
        temp_output = temp_example_output
    powr_running = Popen([sudo_cmd, hpasmcli_cmd, '-s', powr_cmd], stdout=PIPE)
    powr_output = powr_running.communicate()[0]
    if powr_running.returncode != 0:
        powr_output = powr_example_output
    for line in fans_output.splitlines():
        if '#' in line:
            num = line.split()[0]
            sensor_id = 'Fan_' + num
for line in temp_output.splitlines():
    if '#' in line:
        split_line = line.split()
        sensor_num = split_line[0]
        sensor_location = split_line[1]

        if 'CPU' in sensor_location:
            desc_groups = 'temperature, cpu'
        else:
            desc_groups = 'temperature'

        sensor_id = sensor_num + ' Temperature'
        descriptors.append({
            'name': sensor_id.replace('#', ''),
            'call_back': metric_handle_temp,
            'time_max': time_max,
            'value_type': 'float',
            'units': 'C',
            'slope': 'both',
            'format': '%f',
            'description': sensor_location + '.',
            'groups': desc_groups})

for line in powr_output.splitlines():
    if '#' in line:
        power_number = line.split()[2]
        sensor_id = 'Power_' + power_number
        descriptors.append({
            'name': sensor_id.replace('#', ''),
            'call_back': metric_handle_powr,
            'time_max': time_max,
            'value_type': 'float',
            'units': 'W',
            'slope': 'both',
            'format': '%f',
            'description': 'Power usage.',
            'groups': 'power'})

def metric_init(params):
    '''Initialize the random number generator and create the
    metric definition dictionary object for each metric.'''
    global descriptors
    global time_max

    if 'time_max' in params:
        time_max = params['time_max']

    get_sensors(time_max)

    return descriptors

def metric_cleanup():
    '''Clean up the metric module.''
    pass

#This code is for debugging and unit testing
if __name__ == '__main__':
    params = {}
    metric_init(params)
    for d in descriptors:
A.12 auto/chroot_local-hooks/blcr-dkms.chroot

```bash
BLCR_VERSION=$(dpkg -s blcr-dkms 2>/dev/null | grep "^Version" | cut -d' ' -f2 | cut -d- -f1 | cut -d: -f2)
KVER=$(dpkg -l | grep "linux-image" | grep "^ii" | tail -n1 | cut -d' ' -f3 | cut -d- -f3-)
if [ ! -z "$BLCR_VERSION" ]; then
dkms build -m blcr -v $BLCR_VERSION -k $KVER
dkms install -m blcr -v $BLCR_VERSION -k $KVER
fi
```

A.13 auto/chroot_local-hooks/nfsswap.chroot

```bash
if [ -x "/etc/init.d/nfsswap" ]; then
  insserv -f nfsswap >/dev/null
fi
```

A.14 auto/chroot_apt/preferences

```
Package: xauth
Pin: version *
Pin-Priority: -1

#Package: openmpi-checkpoint
#Pin: version *
#Pin-Priority: -1

Package: g++
Pin: version *
Pin-Priority: -1

Package: g++-4.4
Pin: version *
Pin-Priority: -1

Package: python-matplotlib
Pin: version *
Pin-Priority: -1

Package: python-openopt
Pin: release n=siid
Pin-Priority: 600

Package: *
Pin: release n=siid
Pin-Priority: -1
```
Appendix B

EGSnrc & BEAMnrc Modifications

This appendix provides a set of difference files indicating the major changes needed to improve the stability of the BEAMnrc package on the cluster when running very large simulations. These modifications are against the distribution version of BEAMnrc and EGSnrc V4.2.3.2. The differences are listed in the unified diff format to allow simple patching. The original files are the secondary comparison.

B.1 EGSnrc unified diff

diff -ubr egsnrc/cutils/egs_c_utils.c /home/beamuser/HEN_HOUSE/cutils/egs_c Utils.c
--- egsnrc/cutils/egs_c_utils.c 2004-10-19 17:35:42.000000000 -0400
+++ /home/beamuser/HEN_HOUSE/cutils/egs_c_utils.c 2012-03-16 10:53:58.000000000 -0400
@@ -200,22 +200,32 @@
#endif
if( __is_locked == 1 ) { *status = 0; return; }
if( __my_fd < 0 ) { *status = -1; return; }
+ /*
  for(i1=0; i1<5; i1++) {
    for(i2=0; i2<12; i2++) {
-      *status = fcntl(__my_fd,F_SETLK,&fl_write);
+      *status = fcntl(__my_fd,F_SETLKW,&fl_write);
if( *status == 0 ) { __is_locked = 1; return; }
sleep(1);
    }
  }
-  printf("egsLockControlFile: failed to lock file for 12 seconds...\n");
+  printf("egsLockControlFile: failed to lock file while blocking...\n");
  }
- /*
- *status = fcntl( _my_fd,F_SETLK,&fl_write);
- if( *status == 0 ) __is_locked = 1;
+ /*
+ while(__is_locked == 0) {
+   *status = fcntl( _my_fd,F_SETLKW,&fl_write);
+   if( *status == 0 ) {
+     __is_locked = 1;
+     return;
+     */
+     */
```c
+ } else {
+      perror("egsLockControlFile: failed to lock file. Retrying.\nError was: ");
+      sleep (0.1);
+  }  
+}
+
+
+#elif DEBUG
+  if ( *status != 0 ) perror("error was");
+#endif
+
+void egsUnlockControlFile(int *status) {
+  perror("egsLockControlFile: failed to lock file. Should not get here.\nError was: ");
+
+  perror("egsLockControlFile: failed to lock file after 1 minute wait!
Error was: ");
+
  if ( *status == 0 ) perror("error was");

    printf("egsLockControlFile: failed to lock file after 1 minute wait!
Error was: ");

void egsUnlockControlFile(int *status) {

diff -ubr egsnrc/egs++/egs_run_control.cpp /home/beamuser/HEN_HOUSE/egs++/egs_run_control.cpp
--- egsnrc/egs++/egs_run_control.cpp 2007-06-13 13:54:51.000000000 -0400
+++ /home/beamuser/HEN_HOUSE/egs++/egs_run_control.cpp 2012-03-13 09:51:47.000000000 -0400
@@ -302,7 +302,7 @@
  
  #endif
  for(int i1=0; i1<5; i1++) {
-                int res = fcntl(fd,F_SETLK,&fl_write);
+                int res = fcntl(fd,F_SETLKW,&fl_write);
          if( !res ) { is_locked = true; return true; }  
          WAIT_FOR_FILE;
  }

Only in egsnrc/pegs4/inputs: .air.pegs4inp.swp
diff -ubr egsnrc/pieces/egs_c_utils_unix.c /home/beamuser/HEN_HOUSE/pieces/egs_c_utils_unix.c
--- egsnrc/pieces/egs_c_utils_unix.c 2004-09-29 18:52:57.000000000 -0400
+++ /home/beamuser/HEN_HOUSE/pieces/egs_c_utils_unix.c 2012-03-16 10:54:11.000000000 -0400
@@ -86,22 +86,32 @@
  
    #endif
  if( __is_locked == 1 ) { *status = 0; return; }  
+    if( __my_fd < 0 ) { *status = -1; return; }  
+  
+    for(i1=0; i1<5; i1++) {
+      for(i2=0; i2<12; i2++) {
+        *status = fcntl(__my_fd,F_SETLK,&fl_write);
+        if( *status == 0 ) { __is_locked = 1; return; }  
+        sleep(1);  
+      }
+      printf("egsLockControlFile: failed to lock file for 12 seconds...
Error was: ");
+    }
+    
+  */
+  *status = fcntl(__my_fd,F_SETLK,&fl_write);
+  if( *status == 0 ) { __is_locked = 1;  
+    */
+  while(__is_locked == 0) {
+    *status = fcntl(__my_fd,F_SETLK,&fl_write);
+    if( *status == 0 ) {
+      __is_locked = 1;  
+      return;  
+    }  
+    else {
+      perror("egsLockControlFile: failed to lock file. Retrying.\nError was: ");
+      sleep (0.1);
+    }  
+  }
+  
+  #ifdef DEBUG
  if ( *status != 0 ) perror("error was");
+  
+  printf("egsLockControlFile: failed to lock file after 1 minute wait!
Error was: ");
+  perror("egsLockControlFile: failed to lock file. Should not get here.\nError was: ");
}  

void egsUnlockControlFile(int *status) {

diff -ubr egsnrc/scripts/batch_options.pbs /home/beamuser/HEN_HOUSE/scripts/batch_options.pbs
--- egsnrc/scripts/batch_options.pbs 2011-05-09 19:10:01.000000000 -0400
+++ /home/beamuser/HEN_HOUSE/scripts/batch_options.pbs 2011-05-09 19:10:01.000000000 -0400
```

# Generic (job name independent) batch options
#
-generics_bo="-j eo -m ae -M ${USER}@irs.phy.nrc.ca"
+generics_bo="-j eo -m ae -M ${USER}@localhost"
# In the above:
# -j eo means to join standard error and standard output
# -m ae means to send e-mail at job completion.
@@ -74,6 +74,7 @@
# The 2 PBS queues that we have here at NRC.
# Change these if you have defined different queues.
#
-short_queue="-q oshort"
-long_queue="-q onlong"
+short_queue="-q batch"
+long_queue="-q fast"
+medium_queue="-q batch"

diff -ubr egsnrc/scripts/batch_options.pbsdsh
/home/beamuser/HEN_HOUSE/scripts/batch_options.pbsdsh
--- egsnrc/scripts/batch_options.pbsdsh 2008-08-21 15:41:25.000000000 -0400
+++ /home/beamuser/HEN_HOUSE/scripts/batch_options.pbsdsh 2012-03-24 16:23:38.000000000 -0400
@@ -31,7 +31,7 @@
# Generic (job name independent) batch options
#
-generics_bo="-j eo -m ae -M ${USER}@irs.phy.nrc.ca"
+generics_bo="-j eo -m ae -M ${USER}@localhost"
# In the above:
# -j eo means to join standard error and standard output
# -m ae means to send e-mail at job completion.
@@ -74,6 +74,6 @@
# The 2 PBS queues that we have here at NRC.
# Change these if you have defined different queues.
#
-short_queue="-q oshort"
-long_queue="-q onlong"
+
+short_queue="-q batch"
+long_queue="-q fast"
+medium_queue="-q batch"

diff -ubr egsnrc/scripts/batch_options.pbsdsh
/home/beamuser/HEN_HOUSE/scripts/batch_options.pbsdsh
--- egsnrc/scripts/batch_options.pbsdsh 2008-08-21 15:41:25.000000000 -0400
+++ /home/beamuser/HEN_HOUSE/scripts/batch_options.pbsdsh 2012-03-24 16:23:38.000000000 -0400
@@ -31,7 +31,7 @@
# Generic (job name independent) batch options
#
-generics_bo="-j eo -m ae -M ${USER}@irs.phy.nrc.ca"
+generics_bo="-j eo -m ae -M ${USER}@localhost"
# In the above:
# -j eo means to join standard error and standard output
# -m ae means to send e-mail at job completion.
@@ -74,6 +74,6 @@
# The 2 PBS queues that we have here at NRC.
# Change these if you have defined different queues.
#
-short_queue="-q oshort"
-long_queue="-q onlong"
+
+short_queue="-q batch"
+long_queue="-q fast"
+medium_queue="-q batch"

diff -ubr egsnrc/scripts/compile_user_code /home/beamuser/HEN_HOUSE/scripts/compile_user_code
--- egsnrc/scripts/compile_user_code 2003-11-03 16:24:08.000000000 -0500
+++ /home/beamuser/HEN_HOUSE/scripts/compile_user_code 2011-07-25 12:50:00.000000000 -0400
@@ -1,4 +1,4 @@
-#!/bin/sh
+#!/bin/bash
#
#******************************************************************************
# $Id: compile_user_code,v 1.3 2003/11/03 19:45:34 iwan Exp $
#\ndiff -ubr egsnrc/scripts/egsnrc_bashrc_additions
/home/beamuser/HEN_HOUSE/scripts/egsnrc_bashrc_additions
@@ -77,3 +77,6 @@
# Source this file if available
#
+export EGS_BATCH_SYSTEM=pbsdsh
+export EGS_BATCH_SYSTEM=pbs

diff -ubr egsnrc/scripts/egui /home/beamuser/HEN_HOUSE/scripts/egui
@@ -12,6 +12,6 @@
#
EGS_HOME=$HOME/egsnrc_mp/
```bash
EGS_CONFIG=/home/course/HEN_HOUSE/specs/g77.conf
-EGS_BATCH_SYSTEM=nqs
+EGS_BATCH_SYSTEM=pbsdsh
export EGS_HOME EGS_CONFIG EGS_BATCH_SYSTEM
/home/course/HEN_HOUSE/bin/g77/egs_gui

diff -ubr egsnrc/scripts/run_pbsdsh_task /home/beamuser/HEN_HOUSE/scripts/run_pbsdsh_task
--- egsnrc/scripts/run_pbsdsh_task 2011-05-09 19:10:01.000000000 -0400
@@ -24,10 +24,10 @@
done
    # space all jobs by at least 0.5 seconds, to avoid race condition on lock file
    - usleep $(($job*500000))
    + #sleep $(($job*500000))
    # extra user-specified sleep seconds between each job
    - sleep $(($job*$dshsleep))
    + #sleep $(($job*$dshsleep))
fi

# launch the job
diff -ubr egsnrc/scripts/run_user_code_batch /home/beamuser/HEN_HOUSE/scripts/run_user_code_batch
@@ -117,7 +117,7 @@
  dshsleep=`echo $1 | sed 's/sleep=//'`
  start=*)  start_job=`echo $1 | sed 's/start=//'`
  stop=*)  stop_job=`echo $1 | sed 's/stop=//'`
-      fresh=no)  fresh=no
+       fresh=*)  fresh=`echo $1 | sed 's/fresh=//'`
  *)
fi

# submit the following stdin script to qsub, which will invoke the run_pbsdsh_task script
+ if test "x$dshnodes" = x; then
    $batch_command $batch_options $queue $other_args <<EOF
    #!/bin/sh
    #PBS -l dshnodes,EGS_HOME,EGS_CONFIG
    #PBS -v HEN_HOUSE,EGS_HOME,EGS_CONFIG
    #PBS -l nodes=$n_parallel
    pbsdsh \$HEN_HOUSE/scripts/run_pbsdsh_task "$command" "$basename" "$dshsleep"
    EOF
+    else
+      $batch_command $batch_options $queue $other_args <<EOF
+    #!/bin/sh
+    #PBS -l dshnodes,procs=$n_parallel
+    #PBS -v HEN_HOUSE,EGS_HOME,EGS_CONFIG
+    pbsdsh \$HEN_HOUSE/scripts/run_pbsdsh_task "$command" "$basename" "$dshsleep"
+    EOF
+    fi
+    exit 0
fi
```

### 111
IMPLICIT NONE;

PARAMETER (MAX = 1024);  
"---MAX used in defining the maximum number of---"

REAL X(MAX),Y(MAX),ERRY(MAX),YMIN,HISTXMIN,ERRYOLD,SMALLESTX,SMALLESTY,FUDGE;
INTEGER NPTS,NPTS1,CURVENUM, COUNT,UNITNUM,TYPE,AXISTYPE;

B.2 BEAMnrc unified diff

+++ /home/beamuser/HEN_HOUSE//omega/beamnrc/beamnrc.mortran 2012-03-17 12:39:59.000000000 -0400
@@ -5593,8 +5593,9 @@
"IARG 3 means user has asked for a discard not at a scoring plane"
"IARG 5 means we are at scoring plane"
OUTPUT IARG;(/' ********IN AUSGAB:  IRL = 1 with IARG=',I5,/
-       'This should not happen'//);
-       $CALL(EXIT(1);
+       'This should not happen, fixed'//);
+       "$CALL(EXIT(1)
+       RETURN;

IF(IARG = 3) [RETURN;} "Outside the geometry, HOWFAR will discard"
"If IARG = 5, it will return after writing phase space"

--- hen_house//omega/progs/beamdp/beamdp.mortran 2008-03-31 10:21:47.000000000 -0400
@@ -93,7 +93,7 @@

"note: this number should be consistent with
"that used in the simulation code (=-$MXSMFS in BEAM)"

--- hen_house//omega/progs/dosxyz_show/dosxyz_show.c 2007-10-03 02:59:00.000000000 -0400
+++ /home/beamuser/HEN_HOUSE//omega/progs/dosxyz_show/dosxyz_show.c 2011-10-27 11:27:03.000000000 -0400
@@ -1198,12 +1198,10 @@

"note: this number should be consistent with
"that used in the simulation code"
+ { 
    strcat(dm_file,".egsphant");
    printf(" Density matrix: %s\n",dm_file);
    fp = fopen(dm_file,"r");
    @ -1248,26 +1247,29 @
    printf(" Error opening density matrix file %s\n",dm_file);
    exit(1);
    }

+ if(argc >= 3) {
    dose_name = argv[2];
    strcpy(dose_file,dose_name);
+ } else {
+   strncpy(dose_file,dm_file,strlen(dm_file)-9);
+   if ( dose_file[strlen(dose_file)-1] == '.' ){
+     strcat(dose_file,"3ddose");
+   } else {
+     strcat(dose_file,".3ddose");
+   }
+   printf(" No dose matrix specified! Using %s\n", dose_file);
+   }

+ len = strlen(dose_file);
+ if( len > 6 ) { 
+   if( strcmp(&dose_file[len-7],".3ddose") != 0 )
+     strcat(dose_file,".3ddose");
+   }
+ else strcat(dose_file,".3ddose");
+   home = getenv("HOME");
+   strcpy(dose_dir,home);
+   ind1 = strlen(home);
+   if( strcmp(&dose_dir[ind1-1],"/")){strcpy(&dose_dir[ind1],"/"); ++ind1;}
+   strcpy(&dose_dir[ind1],"egs4/dosxyz/"); ind1 = strlen(dose_dir);
+   /* strcpy(&dose_dir[ind1],"."); ind1 = strlen(dose_dir); */
+   strcat(dose_dir,dose_file); ind1 = strlen(dose_dir);
+   printf(" Dose matrix: %s\n",dose_dir);
+   /* printf("    Dose matrix: %s\n",dose_dir); */
+   } else printf(" No dose matrix specified!\n");
+   printf("    Dose matrix: %s\n",dose_dir);
/* allocate density matrix data */
density_data = malloc( sizeof(Data3D) );
@@ -1318,7 +1320,7 @@
   #endif
   /* skip material information */
   for(k=0; k<density_data->VoxelNumber.z; k++) {
-     for(i=0; i<density_data->VoxelNumber.y; i++) fscanf(fp,'%s\n',buf);
+     for(i=0; i<density_data->VoxelNumber.y; i++) fscanf(fp,'%*\n');
     fscanf(fp,'%s\n');
   }
   i_max = density_data->VoxelNumber.x;
@@ -1398,9 +1400,9 @@
   printf(" Can not allocate %d bytes for dose matrix!\n",dm_size);
   exit(2);
} 
*/
printf(" %d bytes allocated for dose matrix\n",dm_size);
+ printf(" %d bytes allocated for dose matrix\n",dm_size);
+
+ if( argc >= 3) {
+   fp = fopen(dose_file,"r");
+   if( fp == NULL ) {
+     printf(" %s does not exist,\n will try %s\n",dose_file,dose_dir);
@@ -1468,7 +1470,6 @@
   }
}
}
diff -ubr /home/beamuser/HEN_HOUSE//omega/progs/gui/beamnrc/run_beamnrc.tcl
 /home/beamuser/HEN_HOUSE//omega/progs/gui/beamnrc/run_beamnrc.tcl
--- /home/beamuser/HEN_HOUSE//omega/progs/gui/beamnrc/run_beamnrc.tcl
+++ /home/beamuser/HEN_HOUSE//omega/progs/gui/beamnrc/run_beamnrc.tcl
14:40:44.000000000 -0400
@@ -342,7 +342,7 @@
     cd $inp_file_dir
     .bot.frm.dialg insert end "\n\n\n"
     if $batch==0 {
-           set rpipe [open "|BEAM_${mod_base}$opt p $pegs4base i $inp_base" "r"]
+           set rpipe [open "|BEAM_${mod_base}$opt p $pegs4base i $inp_base |& cat" "r"]
     } else {
           set arg "$speed"
     } if [catch {set egs_batch_system $env(EGS_BATCH_SYSTEM)}] {
@@ -354,7 +354,7 @@
     } }
     puts $arg
     set batch_script [file join $hen_house scripts run_user_code_batch]
-           set rpipe [open "|$batch_script BEAM_${mod_base}$opt $inp_base $pegs4base $arg 2>&1" "r"]
+           set rpipe [open "|$batch_script BEAM_${mod_base}$opt $inp_base $pegs4base $arg |& cat" "r"]
     } fconfigure $rpipe -blocking 0
     fileevent $rpipe readable [list My_Reader $rpipe]

diff -ubr /home/beamuser/HEN_HOUSE//omega/progs/gui/dosxyznrc/run_dosxyznrc.tcl
 /home/beamuser/HEN_HOUSE//omega/progs/gui/dosxyznrc/run_dosxyznrc.tcl
--- /home/beamuser/HEN_HOUSE//omega/progs/gui/dosxyznrc/run_dosxyznrc.tcl
+++ /home/beamuser/HEN_HOUSE//omega/progs/gui/dosxyznrc/run_dosxyznrc.tcl
14:41:30.000000000 -0400
@@ -259,7 +259,7 @@
     cd $inp_file_dir
     .bot.frm.dialg insert end "\n\n\n"
     if $batch==0 {
-          set rpipe [open "|dosxyznrc -p $pegs4base -i $inp_base" "r"]
+          set rpipe [open "|dosxyznrc -p $pegs4base -i $inp_base |& cat" "r"]
     } else {
           set arg "$speed"
     } if [catch {set egs_batch_system $env(EGS_BATCH_SYSTEM)}] {
@@ -270,7 +270,7 @@
     } set arg "$arg p=$nparallel"
     } set batch_script [file join $hen_house scripts run_user_code_batch]
-         set rpipe [open "|$batch_script dosxyznrc $inp_base $pegs4base $arg 2>&1" "r"]
+         set rpipe [open "|$batch_script dosxyznrc $inp_base $pegs4base $arg |& cat" "r"]
     } fconfigure $rpipe -blocking 0
     fileevent $rpipe readable [list My_Reader $rpipe]

diff -ubr /home/beamuser/HEN_HOUSE//omega/progs/statdose/statdose.mortran
 /home/beamuser/HEN_HOUSE//omega/progs/statdose/statdose.mortran
--- /home/beamuser/HEN_HOUSE//omega/progs/statdose/statdose.mortran
+++ /home/beamuser/HEN_HOUSE//omega/progs/statdose/statdose.mortran
08:56:30.000000000 -0500
@@ -172,7 +172,7 @@
 " - note that 5 is about the largest number of curves able to be stored
 " without sending the array size out of this world
-REPLACE {NDIST1_MAX} WITH {15};  "Was 5  DR "
+REPLACE {NDIST1_MAX} WITH {5};  "Was 5  DR "
 " - maximum number of curves which can be plotted
 " this doesn't have to correspond to NDIST1_MAX because
@@ -180,13 +180,13 @@
-REPLACE {MAXCURVE} WITH {400};
+REPLACE {MAXCURVE} WITH {1024};
 " - maximum number of points in one curve I think which can be plotted
-REPLACE {MAXPOINTS} WITH {400};
+REPLACE {MAXPOINTS} WITH {1024};
- maximum number of voxels in each direction which can be used
- replace \$MAXVOXX WITH (28);
- replace \$MAXVOXY WITH (28);
- replace \$MAXVOXZ WITH (400);
- replace \$MAXVOXEL WITH (400);  
"Note: MAXVOLXEL must be the greatest of "
+ replace \$MAXVOXX WITH (256);
+ replace \$MAXVOXY WITH (256);
+ replace \$MAXVOXZ WITH (700);
+ replace \$MAXVOXEL WITH (700);  
"Note: MAXVOLXEL must be the greatest of "
"the above 3 values!!"
"Note the main arrays are dimensioned $NDIST1_MAX*$MAXVOXX*$MAXVOXY*$MAXVOXZ"
"2 x 128 x 128 x 64 compiles under linux at NRC, but NOT SGI with 80MByte"

diff -ubr
/home/beamuser/HEN_HOUSE/user_codes/dosxyznrc/dosxyznrc_user_macros.mortran
/home/beamuser/HEN_HOUSE/user_codes/dosxyznrc/dosxyznrc_user_macros.mortran
--- hen_house/user_codes/dosxyznrc/dosxyznrc_user_macros.mortran 2011-05-09
19:10:09.000000000 -0400
+++ /home/beamuser/HEN_HOUSE/user_codes/dosxyznrc/dosxyznrc_user_macros.mortran 2012-03-13
10:08:55.000000000 -0400
@@ -67,9 +67,9 @@
"It works with $STAT = 1 but gives no stats"
 REPLACE \$MXMED WITH (7) "Maximum number of media
 REPLACE \$MXSTACK WITH (10000) "Maximum particle stack size
- replace \$IMAX WITH (128) "Maximum number of x cells
- replace \$JMAX WITH (128) "Maximum number of y cells
- replace \$KMAX WITH (128) "Maximum number of z cells
+ replace \$IMAX WITH (1026) "Maximum number of x cells
+ replace \$JMAX WITH (1026) "Maximum number of y cells
+ replace \$KMAX WITH (4) "Maximum number of z cells
 REPLACE \$MAXDOSE WITH \{COMPUTE \$IMAX*\$JMAX*\$KMAX+1\}
 "Number of dose regions, can be set to < $IMAX*$JMAX*$KMAX if "
 "necessary to reduce memory requirement, +1 for outside region"
@@ -79,7 +79,7 @@
 "Following macro set to 1 to zero doses with error>50% in the .3ddose file
 "Note that dose errors do not get zeroed
 "Set to any other number to not zero these doses
- replace \$DOSEZERO WITH (1);
+ replace \$DOSEZERO WITH (0);
 "this is the maximum number of user-selected theta-phi combinations"
 "in isource=7,8"
Appendix C

Accelerator Model Input Files

C.1 6MVmohan_tomylar_10x10.egsinp

```
Epid 5x5
2
H2O700ICRU
AIR700ICRU
0.7, 0.1, 0, 0, 0
-3, -3, 1, 1
-75
54.52, 1
0.04, 1024
54.52, 1
-75
54.52, 1
.04, 1024
54.52, 1
0
.1
0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0
2, 2, 0, 0, 0, 0, 0
2, 0, 2, 20, 1, 20, 0, 100

4000000000, 0, 24, 33, 97, 40, 1, 0, 2, 1, 0.1, 0, 0, 0, 1, 0

:Start MC Transport Parameter:

Global ECUT= 0.7
Global PCUT= 0.1
Global SMAX= 5
ESTEPE= 0.25
XIMAX= 0.5
Boundary crossing algorithm= PRESTA-I
Skin depth for BCA= 0
Electron-step algorithm= PRESTA-II
Spin effects= On
Brems angular sampling= Simple
Brems cross sections= BH
Bound Compton scattering= Off
Compton cross sections= default
Pair angular sampling= Simple
Pair cross sections= BH
Photoelectron angular sampling= Off
Rayleigh scattering= Off
```
### C.2 cylinder_imrt.egsnp

**6MV Mohen IMRT simulation to bottom of phantom epid.**

```
AIR700ICRU
0, 0, 0, 0, 1, 2, 1, IWATCH ETC.
7874520, 6133, 1521, 900.0, 0, 0, 0, NCASE ETC.
0, 21, 1, 1, 1, 0, 0, 0, 0, IQIN, ISOURCE + OPTIONS
/home/beamuser/egsnrc_mp/BEAM_cylinder/inp_IMRT.egsphsp1
0, 0, 0.7, 0.01, 0, 1, 0.3, 0, ECUT,PCUT,IREJCT,ESAVE
0, 1, 1, 7, 7, PHOTON FORCING
1, 2, SCORING INPUT
1, 1, 1,
0, DOSE COMPONENTS
60, Z TO FRONT FACE
*********** start of CM SIDETUBE with identifier PHANTOM ************
35, RMAX
Phantom
85, ZMIN
30, ZTHICK
100, ZCYL
-35, 35, XMIN, XMAX
1, # CYLINDERS
15,
0.7, 0.01, 0, 0,
H2O700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
*********** start of CM SLABS with identifier TABLE *************
40, RMAX
Tabletop
2, NSLABS
115, ZMIN
1, 0.7, 0.01, 0, 0
170C700ICRU
34, 0.7, 0.01, 0, 0
AIR700ICRU
************** end of all CMs***************
```

### Global Parameters

- **Global ECUT= 0.7**
- **Global PCUT= 0.01**
- **Global SMAX= 5**
- **ESTEPE= 0.25**
- **XIMAX= 0.5**

- **Boundary crossing algorithm= EXACT**
- **Skin depth for BCA= 0**
- **Electron-step algorithm= PRESTA-II**
- **Spin effects= On**
- **Brems angular sampling= Simple**
- **Brems cross sections= BH**
- **Bound Compton scattering= Off**
- **Compton cross sections= default**
- **Pair angular sampling= Simple**
- **Pair cross sections= BH**
- **Photoelectron angular sampling= Off**
- **Rayleigh scattering= Off**
- **Atomic relaxations= Off**
- **Electron impact ionization= Off**
Photon cross sections= si
Photon cross-sections output= Off
:Stop MC Transport Parameter:
##################################################
Appendix D

Ancillary Phase Space Tools

The programs listed below were created with the goal of manipulating the phase space results from BEAMnrc simulations described in Chapter 4. Each program is written with the goal of being relatively user friendly; however, great care must still be taken as the programs directly manipulate simulation results in a way that is not always recoverable. Programs written in C such as D.1 are written with large file support enabled to support files larger than 4 GB; however, many file systems (notably FAT32) cannot support large files, so portability may be compromised.

D.1 phsp_fix.c

This program functions to correct errors in the phase space file if a run has terminated incorrectly (often due to the disk filling prior to completion of a write). It scans through the phase space file and accounts for each particle in the totals. It cannot correct for the number of histories in the original simulation as this information must come from the simulation when writing; however, the deviation from the true value will be at maximum overestimating by one write cycle.

#pragma pack (1)
#define _LARGEFILE_SOURCE
```c
#define _LARGEFILE64_SOURCE
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <math.h>
#include <stdarg.h>
#include <string.h>
#include <stdio_ext.h>
#include <errno.h>
#include <inttypes.h>

#define HEADER_FORMAT_STRING "\
  Mode:\t\t%.5s
  Number of Particles:\t%d.
  Number of Photons:\t%d.
  Maximum Energy:\t%e MeV.
  Minimum Energy:\t%e MeV.
  Number of Incoming:\t%e"
#define PHSP_FORMAT_STRING "%.8f %d %8.3f %8.3f %8.3f %8.3f %8.2e 
#define TRUE 0x01
#define BAR_LENGTH 78
#define BAR_CHARACTER '+
#define PHSP_FIX_NAME "Phase space file fixer."
#define PHSP_FIX_VERSION "1.0"

typedef struct {
  /* Phase Space file Mode descriptor (5 bytes) */
  char MODE_RW[5];
  /* Number of particles in file (4 bytes), Number of Photons (4 bytes) */
  int NPPHSP, NPHOTPHSP;
  /* Maximum KE in file, Minimum e- KE in file, Number of incoming phase space entries from source */
  float EKMAXPHSP, EKMINPHSP, NINCPHSP;
}
  } PhaseSpaceHeaderType;

size_t PhaseSpaceHeaderSize = sizeof(PhaseSpaceHeaderType);

typedef struct {
  /* Latch bits which determine particles origin, and type */
  unsigned int LATCH;
  /* Energy (KE + Rest), X,Y position, U,V direction cosines, weight, and ZLAST if present 
   * Note for ZLAST: if the file is of MODE 0 or MODE 1, this will not be recorded. 
   * The program should never try to access it as it will have an undefined value. 
   * It is declared here because C will not allow for dynamic typedefs. */
  float ESHORT, X, Y, U, V, WT, ZLAST;
}
  } PhaseSpaceEntryType;

typedef enum { False=0, True=1, Error=-1 } tri_state;

int read_entry(FILE *, PhaseSpaceEntryType *, off64_t);
int print_header(PhaseSpaceHeaderType const *);
int print_entry(PhaseSpaceEntryType const *, tri_state);
tri_state parse_yes_no(tri_state, const char *, ...);

int main(int argc, char * argv[])
{
  struct stat64 info;
```
FILE *datafile;
PhaseSpaceHeaderType header;
PhaseSpaceEntryType entry;
tri_state Assume_Yes, truncate_file, process_file;
char *DatafileName;
off64_t new_length=0;

if(argc<2 || !(strcmp(argv[1],"--help") || !(strcmp(argv[1],"-h"))))
{
    fprintf(stderr, "%s (%s) by Nicholas Sperling

" "Usage: %s [-y] filename.
" " - y: Causes the program to assume yes to all questions.
" " file.egsphsp#: The phase space file which you would like to correct.
" PHSP_FIX_NAME, PHSP_FIX_VERSION, argv[0]);
    exit(1);
}
if (!strcmp(argv[1],"-y"))
{
    Assume_Yes=True;
    DatafileName=argv[2];
} else
{
    Assume_Yes=False;
    DatafileName=argv[1];
}

datafile = fopen64(DatafileName, "r+");
if(datafile == NULL)
{
    switch (errno)
    {
    /* Encountered some error in opening file, try to deal with it. */
    case EACCES: case EINVAL: /* Access error, first try opening read only */
        fprintf(stderr, "Unable to open file "%s" for writing due to permissions. Attempting read only access.
", DatafileName);
        datafile = fopen64(DatafileName, "r");
        if (datafile == NULL) { exit(errno); }
        break;
    case EISDIR: /* File is a directory, we do not handle this. */
        fprintf(stderr, "Error: "%s" is a directory, cannot be opened.
", DatafileName);
        exit(errno);
    case ENOENT: case ENOTDIR:
        fprintf(stderr, "Error: "%s" does not exist, aborting.
", DatafileName);
        exit(errno);
    default:
        fprintf(stderr, "Error opening file "%s", please ensure the file listed is correct.
", DatafileName);
    }
}
if (fstat64(fileno(datafile), &info))
{
    fprintf(stderr, "stat() error, exit forced.
");
    exit(1);
}

printf("Size of file "%s" is: %" PRIId64 " \n", DatafileName, info.st_size);

int bytes_read = fread(&header, PhaseSpaceHeaderSize, 1, datafile) * PhaseSpaceHeaderSize;
if (bytes_read != PhaseSpaceHeaderSize) {
fprintf(stderr, "Could not read file correctly, %s\bytes_read: %i\n", DatafileName, bytes_read);
    exit(1);
}

print_header(&header);

off64_t ExpectedSize, PhaseSpaceEntrySize;

    /* We do not have a ZLAST in the entry. All datasets are of size:
       off64_t (sizeof(PhaseSpaceEntryType) - 4)
       This is because ZLAST takes 4 bytes to store and is not present */
    PhaseSpaceEntrySize = sizeof(PhaseSpaceEntryType) - 4;
    PhaseSpaceEntrySize = sizeof(PhaseSpaceEntryType);
} else {
    fprintf(stderr, "Error, MODE_RW \"%s\" was not expected. Terminating.\n", header.MODE_RW);
    exit(1);
}

/* Header Size is determined by MODE_RW. Set to match PhaseSpaceEntrySize. */
ExpectedSize = (off64_t) header.NPPHSP + 1) * PhaseSpaceEntrySize;

off64_t extra_bytes = info.st_size % PhaseSpaceEntrySize;

if (extra_bytes != 0) {
    new_length = info.st_size - extra_bytes;
    if (Assume_Yes == True) {
        truncate_file = True;
    } else {
        truncate_file = parse_yes_no(True, "Would you like to truncate the extra %d bytes in the file?", extra_bytes);
    } else if (ExpectedSize != info.st_size) {
        printf("The file does not need to be truncated; however, the header indicates that it has %d entries,\n" " whereas the size of the file indicates %d complete entries.\n" "You are highly encouraged to run the full scan to correct this issue.\n", header.NPPHSP, (int) (info.st_size / PhaseSpaceEntrySize) - 1));
    } else {
        printf("No truncation needed!\n");
    }

    if (new_length != 0 && truncate_file == True) {
        /* The user has chosen to truncate the file, so we will do that now,
           then scan the file to fix header errors */
        if (!ftruncate64 (fileno(datafile), new_length)) {
            fstat64(fileno(datafile), &info);
            printf("The file was truncated successfully, it is now %" PRId64 " bytes.\n", info.st_size);
        } else {
            printf("Error! We could not truncate the file.\n");
            printf("It is possible that another process has a write lock on it, ");
            printf("or that we do not have permission to modify the file.\n ABORTING.\n");
            exit(1);
        }
    }

    // process_file=Assume_Yes==True ? Assume_Yes : parse_yes_no(True, "Would you like to scan the file and correct the header information?");
    if (Assume_Yes == True)
process_file=True;
else
    process_file=parse_yes_no(True, "Would you like to scan the file and correct the header information? ");
if (process_file == Error)
    exit();
else if (process_file == True)
    { // We will now scan the file to determine the number of particles of each type.
        printf("We will now scan the file to determine the number of particles of each type.\n ");
        int particles_read=0, num_electrons=0, num_positrons=0;
        float max_energy=1, min_e_energy=1;
        /* Move to the first entry. Located after the header at a EntrySize boundary. */
        fseeko64(datafile, (off64_t) PhaseSpaceEntrySize, SEEK_SET);
        tri_state is_electron;
        printf("Progress:%72s\n[", "100% ");
        int n_entries_from_size = (info.st_size / PhaseSpaceEntrySize);
        int entries_per_bar_unit = n_entries_from_size / BAR_LENGTH;
        while( ftello64(datafile) < info.st_size )
        {
            is_electron=False;
            if ( read_entry(datafile, &entry, PhaseSpaceEntrySize) < PhaseSpaceEntrySize )
            {
                /* This should not happen! We truncated the file to remove short Entries. Error out and do not write anything. */
                fprintf(stderr, "ERROR, the last entry read was not complete. This is a serious error.\n ");
                fprintf(stderr, "If the phase space file is the correct size, please contact the author.\n ");
                fprintf(stderr, "The error occured at offset %" PRIx64 "\.\n", ftello64(datafile));
                fprintf(stderr, "You may choose to write the current header totals out; however,\n ");
                fprintf(stderr, "this is not recommended as the cause of this error has not been determined.\n ");
                tri_state quit=parse_yes_no(True, "Would you like to quit now (changes will not be saved)? ");
                if ( quit == True || Assume_Yes == True )
                    exit (1);
                else break; /* Drop out of the loop and don't count this particle */
            }
            particles_read++;
            if ( entry.LATCH>>30 & TRUE )
            { /* This particle is an electron */
                num_electrons++;
                is_electron=True;
                entry.ESHORT = fabsf(entry.ESHORT) - (float) 0.511; /* subtract rest mass energy */
            }
            else if ( entry.LATCH>>29 & TRUE )
            { /* This particle is a positron */
                num_positrons++;
                entry.ESHORT = fabsf(entry.ESHORT) - (float) 0.511; /* subtract rest mass energy */
            }
            if ( ( num_electrons==1 || entry.ESHORT < min_e_energy ) && is_electron==True )
            { /* We don't know what value to start with for the minimum energy, so we will just use the energy of the first particle. */
                min_e_energy=entry.ESHORT;
                if ( fabsf(entry.ESHORT) > max_energy)
                    max_energy=fabsf(entry.ESHORT);
        
    }
if ( particles_read % entries_per_bar_unit == 0 )
{
    /* Every time this is zero, we want to display another BAR_CHARACTER */
    printf("%c", BAR_CHARACTER);
    fflush(stdout);
}

header.NPPHS = particles_read;
header.NPHOTPHSP = particles_read - (num_electrons + num_positrons);
header.EKMAXPHSP = max_energy;
header.EKMINPHSPE = min_e_energy;

printf("Complete! Read %d particles. About to save the following header to the file %s Clin ",
      particles_read, DatafileName);
print_header(&header);

/* Now we wish to actually write the new header to the file.
  Note that we did _NOT_ recalculate NINCPHSP because it is not possible
to do from without the simulation. */
fseeko64(datafile, 0L, SEEK_SET);
if (fwrite(&header, PhaseSpaceHeaderSize, 1, datafile) != 1)
    fprintf(stderr, "Error, could not save the header information. You should run
this program again to verify that it has not failed."
   );
else
    printf("Complete!\n\n");

fclose(datafile);
ext(0);

int read_entry(FILE *datafile, PhaseSpaceEntryType *entry, off64_t PhaseSpaceEntrySize)
{
    int bytes_read = fread(entry, sizeof(char), PhaseSpaceEntrySize, datafile);
    if ( bytes_read < PhaseSpaceEntrySize )
        fprintf(stderr, "\nERROR, did not read all of set."\n    );
    return bytes_read;
}

int print_header(PhaseSpaceHeaderType const *header)
{
    return (printf(HEADER_FORMAT_STRING, header->MODE_RW, header->NPPHSP,
                   header->NPHOTPHSP, header->EKMAXPHSP,
                   header->EKMINPHSPE, header->NINCPHSP));
}

int print_entry(PhaseSpaceEntryType const *entry, tri_state print_latch)
{
    int charge;
    if ( (entry->LATCH>>30 & TRUE) ) /* bit 30 set indicates that this is an e- */
        charge=-1;
    else if ( (entry->LATCH>>29 & TRUE) ) /* bit 29 set indicates that this is an e+ */
        charge=1;
    else /* neither bit set indicates that this is a photon */
        charge=0;

    int num_written = printf(PHSP_FORMAT_STRING, entry->ESHORT, charge, entry->X, entry->Y,
                             entry->U, entry->V, entry->WT);
    if (print_latch && num_written > 0)
    {

int looptmp, latchsize = (sizeof(entry->LATCH))*8;
for ( looptmp=0; looptmp<latchsize; looptmp++) {
    printf("%s", TRUE & (entry->LATCH>>looptmp)? "1" : "0");
}
printf("\n");
return num_written;
}

tri_state parse_yes_no(tri_state def_value, const char * fmt, ...)
{
    va_list arg;
    va_start(arg, fmt);
    vprintf(fmt, arg);
    va_end(arg);
    tri_state answer=Error;
    char response;
    switch (def_value) {
        /* We wish to explain to the user what the defaults are. */
        case Error: printf(" (Yes/No): "); break;
        case False: printf(" (Yes/[N]o): "); break;
        default: break; /* Shouldn't get here */
    }
    while ( answer==Error )
    {
        __fpurge(stdin); /* We want to ignore any characters that were spuriously entered */
        if ( scanf("%c", &response) != 1 ) return Error;
        switch ( response ) {
            case 'n': return def_value; break;
            case 'y': case 'Y': return True; break;
            case 'n': case 'N': return False; break;
            default: break;
        }
        printf("Please answer Yes or No: ");
    }
    return Error; /* We should never get here */
}

D.2 set_latch.py

This program will iterate through a phase space file of either format and set the latch bits for each particle to a value depending on the location of the particle. The program operates in a master worker pattern, where the master thread will read blocks of data from the phase space file as a buffer and store the block in a shared queue. Each worker process will pull from the queue and use numpy array calculation tools to set the latch value for each particle. The worker then sends the data to be written to file to a
separate process which either writes directly, or writes a compressed file to disk. The choice of read buffer size was optimized using an iterative technique and a sample 10,000 kparticle phase space source file. This threaded process was designed to speed up the processing of the file; however, it was decided that a lower level language may yield significant improvements beyond this threading.

#!/usr/bin/python2.6
import os
import sys
import numpy
import matplotlib.pyplot as plt
import cPickle
import gzip
import struct
from signal import signal, SIGINT, SIG_IGN
from time import time
from multiprocessing import Process, JoinableQueue, Queue, cpu_count, active_children
import Queue as stdQueue
from cStringIO import StringIO
import subprocess

# MODE0 PHSP (no Z-LAST)
MODE0_dt = numpy.dtype([('names': ['LATCH', 'E', 'X', 'Y', 'U', 'V', 'WT'],
                        'formats': ['u4', 'f4', 'f4', 'f4', 'f4', 'f4', 'f4']})
MODE2_dt = numpy.dtype([('names': ['LATCH', 'E', 'X', 'Y', 'U', 'V', 'WT', 'ZL'],
                        'formats': ['u4', 'f4', 'f4', 'f4', 'f4', 'f4', 'f4', 'f4']})

# This def lets us read in only small chunks of the file as we need it, rather than loading the whole phase space into memory (could be gigs...)
def buf_data_from_file(infile, buf_size):
    while True:
        data = infile.read(buf_size)
        if not data:
            break
        yield data

def data_writer_raw(fn_outfile, header, master_queue, write_queue):
    """ Write the data out to the file, then let the master know we are done """
    signal(SIGINT, SIG_IGN)
    try:
        f_outfile = open(fn_outfile, 'wb')
    except IOError as wr_e:
        sys.stderr.write("Could not open %s for writing, do you have permission? (Error: %s)" % (fn_outfile, wr_e.strerror))
        sys.exit(2)
    f_outfile.write(header)
    master_queue.get()
    for data in iter(write_queue.get, 'DONE'):
        f_outfile.write(data)
        try:
            master_queue.put_nowait('.')
        except stdQueue.Full:
            pass
        write_queue.task_done()
write_queue.task_done()

f_outfile.close()
master_queue.put('DONE')
master_queue.task_done()

```python
def data_writer_compress(fn_outfile, header, master_queue, write_queue):
    """ Write the data out to the file, then let the master know we are done """
    signal(SIGINT, SIG_IGN)
    try:
        f_outfile = open(fn_outfile, 'wb')
    except IOError as wr_e:
        sys.stderr.write("Could not open %s for writing, do you have permission? (Error: %s)" % (fn_outfile, wr_e.strerror))
        sys.exit(2)
    #process = subprocess.Popen(["gzip", "-f"], stdout=f_outfile, stdin=subprocess.PIPE)
    process = subprocess.Popen(["pigz", "-f"], stdout=f_outfile, stdin=subprocess.PIPE)
    process.stdin.write(header)
    master_queue.get()

    for data in iter(write_queue.get, 'DONE'):
        process.stdin.write(data)
        try:
            master_queue.put_nowait('.')
        except stdQueue.Full:
            pass
    else:
        write_queue.task_done()

    process.stdin.close()
    process.wait()
    f_outfile.close()
    master_queue.put('DONE')
    master_queue.task_done()
```

```python
def data_worker(dtype, pixel_size, master_queue, read_queue, write_queue):
    """ Take a series of bytes in on the read_queue, process, then dump to the write_queue. """
    # Don't know how many particles we are getting, but we do have the data type.
    signal(SIGINT, SIG_IGN)
    working_array = None
    master_queue.get()

    for data in iter(read_queue.get, 'DONE'):
        if working_array is None or len(data) != working_array.nbytes:
            # We either haven't set up working_array, or we are at the tail end of the data
            # (giving 3024 values) of the latch, and Y on bits 12-21. If we are
            # outside of the range (-512 - +512 pixels), we will only set bit 1.
            # We Will also want to remove any existing bits, this means using the
            # magic number \xFF000001 to set all bits we are using to zero.

            working_array = numpy.fromstring(data, dtype=dtype)
            Xbits = numpy.empty((len(working_array),), dtype=int)
            Ybits = numpy.empty((len(working_array),), dtype=int)
            Ybits = numpy.empty((len(working_array),), dtype=numpy.uint32)
        else:
            # The working array is setup, and we have data in it. We will just
            # replace the data string with the new one.
            working_array.data[:] = data

    # (Latch bits are zero indexed here, i.e. bits 0-31)
    # Work with the data. We want to attach a latch for X on bits 2-11
    # (giving 1024 values) of the latch, and Y on bits 12-21. If we are
    # outside of the range (-512 - +512 pixels), we will only set bit 1.
    # We Will also want to remove any existing bits, this means using the
    # magic number \xFF000001 to set all bits we are using to zero.

    # This makes the equation fairly simple:
    Xbits = (floor( (X / pixel_size) ) + 512)
    Ybits = (floor( (X / pixel_size) ) + 512)
    if 0 <= Xbits <= 1023 or 0 <= Ybits <= 1023:
        NEW_BITS = (Xbits<<2 | Ybits<<12)
MAGIC = int('0xFF000001', 16)
Xbits[:] = (numpy.floor((working_array['X'] / pixel_size)) + 512)
Ybits[:] = (numpy.floor((working_array['Y'] / pixel_size)) + 512)

Xbits[Xbits < 0] = 0
Ybits[Xbits < 0] = 0
Xbits[Ybits < 0] = 0
Ybits[Ybits < 0] = 0
Xbits[Xbits > 1023] = 0
Ybits[Xbits > 1023] = 0
Xbits[Ybits > 1023] = 0
Ybits[Ybits > 1023] = 0

NEW_BITS = (Xbits << 2 | Ybits << 12)

working_array['LATCH'] = (working_array['LATCH'] & MAGIC) | NEW_BITS

# A quick function to generate 4 digit precis. SI suffixes

def si(value):
    value_str = '%4.4g' % value
    if 'e' in value_str:
        mod = int(value_str.split('e')[1]) % 3
        exp = SI[int(value_str.split('e')[1]) - mod]
        o_str = '%s %s' % ('%.1f' % (float(value_str.split('e')[0]) * 10 ** mod), exp)
    else:
        o_str = '%s %s' % value_str
    return o_str.rjust(7)

if __name__ == '__main__':
    try:
        NCPU = cpu_count()
        except NotImplementedError as e:
            NCPU = 4
        display = False
        benchmark = True
        outputcheck = False
        MAXQ = 400
        data_writer = data_writer_raw
        def_bins = 1024
        def_min = -20.48
        def_max = 20.48
        def_range = (def_min, def_max)
        def_points = numpy.linspace(def_min, def_max, def_bins)
        # def_pixel_size = 0.04
        # Pixel size at 150cm = 0.04 cm
        # Scaled to 60cm for entrance fluence:
        def_pixel_size = 0.016
        def_readblock = 50000
        # Optimized for speed/memory/disk access
        # Apparently, if MODE0, header size is 28 bytes (all w/ 3 bytes pad) if
        # MODE2, header size is 32 bytes (all w/ 7 bytes pad). This seems to be set
# to line up with the size of each block.
# PHSP header format: char MODE_RW[5]; int NPPHSP, NPHOTPHSP; float EMAXPHSP, EKMINPHSP; pad[3]

fmt_PHSPhead = '=5s 2I 3f'
f_readfile = None
f_writefile = None

if len(sys.argv) < 2:
    sys.stderr.write("Usage: %s input_points [output_hist]." % sys.argv[0])
    sys.exit(1)

fn_readfile = sys.argv[1]

# Param 2 is output filename, or we assign it based on input filename.
if len(sys.argv) == 3:
    fn_writefile = sys.argv[2]
else:
    fn_readfile_basename = os.path.basename(fn_readfile)
    # Strip off ".gz" from the basename, if it is there.
    if fn_readfile_basename.endswith(('.gz'),-3):
        fn_readfile_basename = fn_readfile_basename.rpartition(('.))[0]
    # Strip off ".egsphsp?" extention, if there.
    if fn_readfile_basename.endswith(('.egsphsp'),0,-1):
        fn_writefile_basename = fn_readfile_basename.rpartition(('.')[0]
    else:
        fn_writefile_basename = fn_readfile_basename
    fn_writefile = os.path.dirname(fn_readfile) + fn_writefile_basename + "\tagged.egsphsp1"

# Try to open the input file for reading and abort if we cannot.
# Also checks to see if the file looks like a gzip file. If it seems like it is
# we load it as such, and read the first 5 characters to test this. If it is a
# false positive, the data was corrupt anyway (first 2 characters should be ASCII 'MO')
mode_string = ""
try:
    f_readfile = open(fn_readfile,'rb')
    if f_readfile.read(2) == '\x1f\x8b':
        f_readfile = gzip.GzipFile(fileobj = f_readfile)
        f_readfile.rewind()
        mode_string = f_readfile.read(5)
        f_readfile.rewind()
    else:
        f_readfile.seek(0)
        mode_string = f_readfile.read(5)
        f_readfile.seek(0)
except IOError as e:
    sys.stderr.write("Could not open %s, it may not exist. (Error: %s)\n" % ( fn_readfile, e.strerror ))
    sys.exit(1)

if outputcheck:
    # Try to open the output file for reading, if we can, it already exists and we abort.
    # Otherwise, so we should be safe to open for writing.
    try:
        open(fn_writefile,'r')
    except IOError as e:
        if e.errno == 2: # File not found error, we can continue.
            pass
        else:
            raise
    else:
        sys.stderr.write("Error: %s already exists, aborting.\n" % fn_writefile)
        sys.exit(2)

mode_dtype = None
mode_size = None
if mode_string[4] == '0': # MODE0 file, use MODE0_dt (UINT LATCH; float ESHORT,X,Y,U,V,WT)
    7 * 4 Bytes
    mode_dtype = MODE0_dt
mode_size = 28

elif mode_string[4] == '2':  # MODE2 file, use MIDE2_dt (UINT LATCH; float
ESHORT,X,Y,U,V,WT,ZLAST) = 8 * 4 Bytes
    mode_dtype = MIDE2_dt
    mode_size = 32
else:
    sys.stderr.write('Header format corrupted, or unusable mode (%s). Aborting.
', mode_string)
    sys.exit(1)

header = f_readfile.read(mode_size)
if len(header) <> mode_size:
    sys.stderr.write("Error: input file %s does not have a complete header.
" % fn_readfile)
    sys.exit(1)
else:
    # Add the appropriate padding to get up to mode_size length
    fmt_PHSPhead = fmt_PHSPhead + ' ' + str(mode_size - struct.calcsize(fmt_PHSPhead)) + 'x'

    #MODE_RW,NPHSP,NPHOTPHSP,EKMAXPHSP,EKMINPHSP,NINCPHSP = struct.unpack(fmt_PHSPhead,header)
    # cleaner way of doing above, all in a single list
    header_unpacked = struct.unpack(fmt_PHSPhead, header)
    print( 'MODE: %s, np: %i, np_phot: %i, ekmax: %f, ekminE: %f, ninc: %.0f' % header_unpacked)

    # Set the block size.
    readblock_size = def_readblock * mode_size

    master_queue = JoinableQueue(4)
    read_queue = JoinableQueue(MAXQ)
    write_queue = JoinableQueue(MAXQ)

    # Start the writer processes.
    writer = Process(target=data_writer, args=(fn_writefile, header, master_queue, write_queue))
    writer.start()
    master_queue.put('writer started')

    # The first thing to write is the header. Check that it wrote, then hand
    # over control to the data_worker process.
    worker = Process(target=data_worker, args=(mode_dtype, def_pixel_size, master_queue, read_queue, write_queue))
    worker.start()
    master_queue.put('worker started')

    points = 0
    start = time()
    rate = []
    try:
        for data in buf_data_from_file(f_readfile, readblock_size):
            points += len(data) / mode_size

            sys.stdout.write( '%i / %i (%.1f%%) (%spts/s), Q: R-%4i W-%4i
', points, header_unpacked[1],
                              (points*100/header_unpacked[1]), si(points/(time() - start)), read_queue.qsize(), write_queue.qsize())
            sys.stdout.flush()
            read_queue.put(data)

        else:  # EOF reached!
            read_queue.put('DONE')
            read_queue.join()
            read_queue.close()

            for data in iter(master_queue.get, 'DONE'):
                sys.stdout.write( '%i / %i (%.1f%%) (%spts/s), Q: R-%4i W-%4i Read done!
', points, header_unpacked[1],
                              (points*100/header_unpacked[1]), si(points/(time() - start)), read_queue.qsize(), write_queue.qsize())
                sys.stdout.flush()
            master_queue.task_done()
```python
else:
    master_queue.task_done()

write_queue.join()
write_queue.close()
print 'Done!'
except KeyboardInterrupt:
    sys.stderr.write('
Aborting!')
sys.stderr.flush()
    for child in active_children():
        child.terminate()
sys.exit(3)

if benchmark: end = time()
f_readfile.close() # Close file

if benchmark: print 'total time: %fs, average speed: %spts/s' % ((end-start), si(points / (end - start)))
```

**D.3 phsp_set_latch.c**

This program is built by extending phsp_fix.c (D.1) and set_latch.py (D.2) to add latch bits to the phase space file based on each particles position. It is a rewritten version of the python version with the intent of greatly reducing the time to run. It was found that this version reduced run times for a sample phase space file with 220,710,218 particles from 404.7s (average 545.3 kparticles/s) to 369.5s (597.3 kparticles/s).

```c
#pragma pack (1)
#define _FILE_OFFSET_BITS 64
#define _LARGEFILE_SOURCE
#define _LARGEFILE64_SOURCE
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <math.h>
#include <stdarg.h>
#include <string.h>
#include <stdio_ext.h>
#include <errno.h>
#include <inttypes.h>
#include <sys/mman.h>
#include <fcntl.h>

#define HEADER_FORMAT_STRING "
Mode:	%.5s
Number of Particles:	%d
Number of Photons:	%d
Maximum Energy:	%e MeV
Minimum Energy:	%e MeV
Number of Incoming:	%d"
```
```c
#define PHSP_FORMAT_STRING "%.3f %3d %.3f %.3f %.3f %.3f %.2e 
#define TRUE 0x01
#define BAR_LENGTH 78
#define BAR_CHARACTER '+
#define PHSP_FIX_NAME "Phase space file fixer."
#define PHSP_FIX_VERSION "1.0"
#define PIXEL_SIZE 0.016
#define BIN_COUNT 1024

typedef struct {
    /* Phase Space file Mode descriptor (5 bytes) */
    char mode_rwlock[5];
    /* Number of particles in file (4 bytes), Number of Photons (4 bytes) */
    int npfhsp, nphophsp;
    /* Maximum KE in file, Minimum e- KE in file, Number of incoming phase space entries from source */
    float ekmaxphsp, ekminphsp, ninphsp;
    /* Padding to fill remaining bytes is determined by mode_rwlock.
    * If MODE0, header size=28 bytes.
    * If MODE2, header size=32 bytes.
    * To accomodate this, we will seek to MODE_SIZE after reading the header */
} PhaseSpaceHeaderType;

size_t PhaseSpaceHeaderSize = sizeof(PhaseSpaceHeaderType);

typedef struct {
    /* Latch bits which determine particles origin, and type */
    unsigned int latch;
    /* Energy (KE + Rest), X,Y position, U,V direction cosines, and weight */
    float eshort, x, y, u, v, wt;
} sMODE0;

typedef struct {
    /* Latch bits which determine particles origin, and type */
    unsigned int latch;
    /* Energy (KE + Rest), X,Y position, U,V direction cosines, weight, and ZLAST */
    float eshort, x, y, u, v, wt, zlast;
} sMODE2;

typedef sMODE2 PhaseSpaceEntryType;

typedef enum { false=0, true=1, Error=-1 } tri_state;

int read_entry(FILE *, PhaseSpaceEntryType *, off_t);
int print_header(PhaseSpaceHeaderType const *);
int print_entry(PhaseSpaceEntryType const *, tri_state);
tri_state parse_yes_no(tri_state, const char *, ...);
unsigned int compute_latch(PhaseSpaceEntryType *, float, int);

/* Rather than reading the phsp file into memory, we will use memory mapped io
 * to chunk the file into a set of data. The first step though will be to
 * determine the size of the data blocks. */
int main(int argc, char * argv[])
{
    struct stat64 info;
    FILE *datafile;
    PhaseSpaceHeaderType header;
    tri_state Assume_Yes, truncate_file, process_file;
```
char *DatafileName;

off_t new_length = 0;

if (argc<2 || !(strcmp(argv[1], "--help")) || (!strcmp(argv[1], "-h")))
{
    fprintf(stderr,
            "%s (%s) by Nicholas Sperling\n\n" "Usage: %s [-y] filename.\n" "-y: Causes the program to assume yes to all questions.\n" file.egsphsp#: The phase space file which you would like to correct.\n", PHSP_FIX_NAME, PHSP_FIX_VERSION, argv[0]);
    exit(1);
}

if (!strcmp(argv[1], "-y"))
{
    Assume_Yes = true;
    DatafileName = argv[2];
}
else
{
    Assume_Yes = false;
    DatafileName = argv[1];
}

datafile = fopen(DatafileName, "r+");
if (datafile == NULL)
{
    switch (errno)
    {
    /* Encountered some error in opening file, try to deal with it. */
        case EACCES: case EINVAL: /* Access error, first try opening read only */
            fprintf(stderr, "Unable to open file ""%s"" for writing due to permissions. Attempting read only access.\n", DatafileName);
            datafile = fopen(DatafileName, "r");
            if (datafile == NULL) { exit(errno); }
            break;
        case EISDIR: /* File is a directory, we do not handle this. */
            fprintf(stderr, "Error: ""%s"" is a directory, cannot be opened.\n", DatafileName);
            exit(errno);
        case ENOENT: case ENOTDIR:
            fprintf(stderr, "Error: ""%s"" does not exist, aborting.\n", DatafileName);
            exit(errno);
        default:
            fprintf(stderr, "Error opening file ""%s"", please ensure the file listed is correct.\n", DatafileName);
    }
}

if (fstat64(fileno(datafile), &info) == -1)
{
    perror("stat() error, exit forced: ");
    exit(1);
}

printf("Size of file ""%s"" is: %" PRId64 " \n", DatafileName, info.st_size);

int bytes_read = fread(&header, PhaseSpaceHeaderSize, 1, datafile) * PhaseSpaceHeaderSize;
if (bytes_read != PhaseSpaceHeaderSize) {
fprintf(stderr, "Could not read file correctly, %s\nbytes_read: %i\n", DatafileName, bytes_read);
    exit(1);
}
print_header(&header);

off_t ExpectedSize, PhaseSpaceEntrySize;

    /* We do not have a ZLAST in the entry.  All datasets are of size:
     * off_t (sizeof(PhaseSpaceEntryType) - 4)
     * This is because ZLAST takes 4 bytes to store and is not present */
    PhaseSpaceEntrySize = sizeof(sMODE0);
    PhaseSpaceEntrySize = sizeof(sMODE2);
} else {
    fprintf(stderr, "Error, MODE_RW \"%s\" was not expected. Terminating.\n", header.MODE_RW);
    exit(1);
}

/* Header Size is determined by MODE_RW. Set to match PhaseSpaceEntrySize. */
ExpectedSize = (off_t) header.NPHSP + 1) * PhaseSpaceEntrySize;

off_t extra_bytes = info.st_size % PhaseSpaceEntrySize;

if (extra_bytes != 0 ) {
    new_length = info.st_size - extra_bytes;
    if (Assume,Yes == true )
        truncate_file = true;
    else
        truncate_file = parse_yes_no(true, "Would you like to truncate the extra %d bytes in the file?", extra_bytes);
} else if (ExpectedSize != info.st_size)
    printf("The file does not need to be truncated; however, the header indicates that is has %d entries,\n" whereas the size of the file indicates %d complete entries.\n"You are highly encouraged to run the full scan to correct this issue.\n", header.NPHSP, (int) ( (info.st_size / PhaseSpaceEntrySize) - 1 ));
else
    printf("No truncation needed!\n");

if (new_length != 0 & truncate_file == true )
    /* The user has chosen to truncate the file, so we will do that now, 
    * then scan the file to fix header errors */
    if (!ftruncate (fileno(datafile), new_length ) )
        { 
            fstat64(fileno(datafile), &info);
            printf("The file was truncated successfully, it is now %" PRId64 " bytes.\n", info.st_size);
        } else
            { 
                printf("Error! We could not truncate the file.\n");
                printf("It is possible that another process has a write lock on it, ");
                printf("or that we do not have permission to modify the file.\n ABORTING.\n");
                exit(1);
            }
    if (Assume,Yes == true)
        process_file=true;
else
process_file = parse_yes_no(true, "Would you like to scan the file and correct the header information?");

if (process_file == Error)
    exit(1);
else if (process_file == true)
{
    printf("We will now scan the file to determine the number of particles of each type.\n");

    int particles_read = 0, num_electrons = 0, num_positrons = 0;
    float max_energy = 0, min_e_energy = 0;

    /* Move to the first entry. Located after the header at a EntrySize boundary. */
    fseeko(datafile, (off_t) PhaseSpaceEntrySize, SEEK_SET);
    tri_state is_electron;

    printf("Progress: %72s | %79s |\t\n", "100", " ");
    fflush(stdout);

    int n_entries_from_size = (info.st_size / PhaseSpaceEntrySize) - 1;
    float entries_per_bar_unit = (float)n_entries_from_size / BAR_LENGTH;
    int chars_printed = 0;

    /* Start processing the file. We need to work based on the size of the particle structure, so we will have some code repetition. Condition based on the mode. */

    /* The rest of the file will be read using memory mapped io */
    size_t page_size = (size_t) sysconf(_SC_PAGESIZE);
    size_t avp_pages = (size_t) sysconf(_SC_AVPHYS_PAGES);
    size_t read_size = info.st_size;
    tri_state sliding_window = false;
    if (read_size > page_size * avp_pages)
    {
        // We don't have enough memory to hold the whole file, so we will hold just a part of it.
        read_size = page_size * avp_pages;

        if (read_size % PhaseSpaceEntrySize != 0)
            {
            /* We have data on a page boundary, we have to handle this. */
            /* The simplest solution is to choose smaller read sizes to align on both page_size and PhaseSpaceEntrySize boundaries */
            size_t block_size = page_size * PhaseSpaceEntrySize;
            // This will get the next smallest block that will fit.
            size_t gcd = (size_t) floor(PhaseSpaceEntrySize / 2);
            for (; gcd>=1; gcd--)
                {
                printf("%i", gcd);
                if (page_size % gcd == 0 && PhaseSpaceEntrySize % gcd == 0)
                    break;
                }
            block_size = block_size / gcd;
            read_size = (size_t) floor(read_size / block_size) * block_size;
            if (read_size > page_size * avp_pages)
                {
                /* Not enough space remains to allocate along page size and PhaseSpaceEntrySize boundaries. For now, we will not handle this case. */
                // TODO: Handle case of phasespace and page size boundaries not aligning.
                fprintf(stderr, "Cannot allocate enough memory to align on page size boundaries. Aborting.\n");
                exit(1);
            }
        }

        sliding_window = true;
    }

    int MAP_FLAGS = PROT_READ | PROT_WRITE;
    size_t sliding_position = 0;
float *mmap_file = (float*)mmap64(NULL, read_size, MMAP_FLAGS, MAP_SHARED, fileno(datafile), sliding_position);
    if (mmap_file == MAP_FAILED) {
        fclose(datafile);
        perror("mmap failed...");
        exit(1);
    }
    int ret = madvise(mmap_file, read_size, MADV_SEQUENTIAL);
    if (ret != 0) {
        fclose(datafile);
        perror("madvise failed...");
        exit(1);
    }

    /* We will use PhaseSpaceEntryType which is too large for the MODE0 files. */
    int newLATCH = 0;
    for (; particles_read < n_entries_from_size; entries++)
    {
        if (PhaseSpaceEntrySize != sizeof(PhaseSpaceEntryType))
        {
            /* We are in a MODE0 file, we don't have ZLAST so we */
            /* should subtract off one float from the current pointer. */
            entries = ((PhaseSpaceEntryType*)entries - 1);
        }
        if (sliding_window == true && (char*)entries == (char*)mmap_file + read_size)
        {
            /* Time to slide the window, unmap the memory, and map a new set */
            /* pointed at mmap_file, then setup entries. */
            entries = NULL;
            if (munmap(mmap_file, read_size) == -1)
            {
                perror("munmap failed: ");
                exit(1);
            }
            sliding_position++;
            mmap_file = (float*)mmap64(NULL, read_size, MMAP_FLAGS, MAP_SHARED, fileno(datafile), read_size*sliding_position);
            if (mmap_file == MAP_FAILED) {
                fclose(datafile);
                perror("mmap failed...");
                exit(1);
            }
            ret = madvise(mmap_file, read_size, MADV_SEQUENTIAL);
            if (ret != 0)
            {
                fclose(datafile);
                perror("madvise failed...");
                exit(1);
            }
            entries = (PhaseSpaceEntryType *)mmap_file;
        }
        is_electron = false;
        particles_read++;
        E = fabsf(entries->ESHORT);
        if (entries->LATCH>>30 & TRUE)
        {
            /* This particle is an electron */
            num_electrons++;
            is_electron = true;
            E -= (float) 0.511; /* subtract rest mass energy */
        }
        else if (entries->LATCH>>29 & TRUE)
`/* This particle is a positron */
num_positrons++;
E = E - (float) 0.511; /* subtract rest mass energy */`

if ( (num_electrons==1 || E < min_e_energy) && is_electron == true )
    /* We don't know what value to start with for the minimum energy,
     * so we will just use the energy of the first particle. */
    min_e_energy=E;

if ( E > max_energy)
    max_energy=E;
newLATCH = compute_latch(entries, PIXEL_SIZE, BIN_COUNT);
if ( entries->LATCH != newLATCH ){
    fprintf(stderr, "\nError, latch didn't set. Aborting.\n");
    exit(1);
}

if ( particles_read / entries_per_bar_unit > chars_printed )
    /* Every time this is zero, we want to display another BAR_CHARACTER */
    chars_printed++;
    printf("\%c", BAR_CHARACTER);
    fflush(stdout);
}

printf("]\n\n");
header.NPPHSP=particles_read;
header.NPHOTPHSP=particles_read - (num_electrons + num_positrons);
header.EMAXPHSP=max_energy;
header.EMINPHSP=.min_e_energy;
printf("Complete!  Read %d particles.  About to save the following header to the file %s.\n",
    particles_read, DatafileName);
print_header(&header);

/* Now we wish to actually write the new header to the file.
   Note that we did _NOT_ recalculate NINCPHSP because it is not possible
   to do from without the simulation. */
fseeko(datafile, 0L, SEEK_SET);
if ( fwrite(&header, PhaseSpaceHeaderSize, 1, datafile) != 1 )
    fprintf(stderr, "\nError, could not save the header information.\n"
    "You should run this program again to verify that it has not failed.\n");
else
    printf("Complete!\n\n");

fclose(datafile);
exit (0);
}

int read_entry(FILE *datafile, PhaseSpaceEntryType *entry, off_t PhaseSpaceEntrySize) {
    int bytes_read = fread(entry, sizeof(char), PhaseSpaceEntrySize, datafile);
    if ( bytes_read < PhaseSpaceEntrySize )
        fprintf(stderr, "\nERROR, did not read all of set.\n");
    return bytes_read;
}

int print_header(PhaseSpaceHeaderType const *header) {

return (printf(HEADER_FORMAT_STRING, header->MODE_RW, header->NPHSPHSP, header->NPHOTPHSP, header->EHMAXPHSP, header->EHMINPHSPE, header->HINCPHSP));

int print_entry(PhaseSpaceEntryType const *entry, tri_state print_latch)
{
    int charge;
    if ( (entry->LATCH)>>30 & TRUE ) /* bit 30 set indicates that this is an e- */
        charge=-1;
    else if ( (entry->LATCH)>>29 & TRUE) /* bit 29 set indicates that this is an e+ */
        charge=1;
    else /* neither bit set indicates that this is a photon */
        charge=0;

    int num_written = printf(PHSP_FORMAT_STRING, entry->ESHORT, charge, entry->X, entry->Y, entry->U, entry->V, entry->WT);

    if (print_latch && num_written > 0)
    {
        int bit=28;
        for (; bit>=0; bit--)
        {
            if (bit==23 || bit==28) printf(" ");
            printf("%s", (entry->LATCH>>bit) & 0x01? "1" : "0");
        }
        printf("\n");

        return num_written;
    }

    tri_state parse_yes_no(tri_state def_value, const char * fmt, ...)
    {
        va_list arg;
        va_start(arg, fmt);
        vprintf(fmt, arg);
        va_end(arg);

        tri_state answer=Error;
        char response;

        switch (def_value)
        {
            /* We wish to explain to the user what the defaults are. */
            case Error: printf(" (Yes/No): "); break;
            case true: printf(" ([Y]es/No): "); break;
            case false: printf(" (Yes/[N]o): "); break;
            default: break; /* Shouldn't get here */
        }

        while ( answer==Error )
        {
            __fpurge(stdin); /* We want to ignore any characters that were spuriously entered */
            if (scanf("%c", &response) != 1) return Error;

            switch ( response )
            {
                case '\n': return def_value; break;
                case 'y': case 'Y': return true; break;
                case 'n': case 'N': return false; break;
                default: break;
            }

            printf("Please answer Yes or No: ");
        }
        return Error; /* We should never get here */
    }

unsigned int compute_latch(PhaseSpaceEntryType *entry, float pixel_size, int bin_count)
{
    /* Calculates the latch for this particle, sets it, and returns it's value. */


* (Latch bits are zero indexed here, i.e. bits 0-31)
* Work with the data. We want to attach a latch for X on bits 2-11
* (giving 1024 values) of the latch, and Y on bits 12-21. If we are
* outside of the range (-512 - +512 pixels), we will only set bit 1.
* We will also want to remove any existing bits, this means using the
* magic number \xFF000001 to set all bits we are using to zero.
* This makes the equation fairly simple:
* Xbits = (floor( (X / pixel_size) ) + 512)
* Ybits = (floor( (Y / pixel_size) ) + 512)
* if 0 <= Xbits <= 1023 or 0 <= Ybits <= 1023:
*   NEW_BITS = (Xbits<<2 | Ybits<<12)
* else:
*   NEW_BITS = 1<<1
* NEW_LATCH = (LATCH & MAGIC) | NEW_BITS */

unsigned int MAGIC = 0xFF000001;
int Xbits = (int) (floor(entry->X / pixel_size) + 512);
int Ybits = (int) (floor(entry->Y / pixel_size) + 512);
unsigned int NEW_BITS = 2;
if ( Xbits > 0 && Xbits < 1023 &&
     Ybits > 0 && Ybits < 1023 ) {
    /* The value is within the range of 0-1023, we can set the bits */
    NEW_BITS = (unsigned int) (Xbits<<2 | Ybits<<12);
}
return (entry->LATCH = (entry->LATCH & MAGIC) | NEW_BITS);}
Appendix E

Virtual EPID Characterization

The characterization of the virtual EPID is accomplished through several files. Example input files to the BEAMnrc package are presented below, as well as several programs designed to manipulate the results. Finally, the deconvolution parameter solver and deconvolution package are presented below.

E.1 BEAM_6MVmohan_tomylar_20x20_Epid.egsnp

This is an example egsnrc input file used to create the fluence at the level of the EPID.

```plaintext
6MV Mohen spectrum, photon source to top of jaws. LARGE PHSP
AIR?000CRU
0, 0, 0, 0, 1, 3, 1, IWATCH ETC.
10000000000, 22987, 8487, 900.0, 0, 0, 0, NCASE ETC.
0, 1, 0.4, 0.6, 0, 1.0, 0, 0, IQIN, ISOURCE + OPTIONS
1, SPECTRUM
/home/beamuser/HEN_HOUSE/spectra/mohan6.spectrum
1
0, 0, 0.7, 0.01, 0, 1, 0.3, 0 , ECUT,PCUT,IREJCT,ESAVE
0, 1, 1, 7, 7, PHOTON FORCING
1, 7, SCORING INPUT
1, 1
1,
0, DOSE COMPONENTS
0.0, Z TO FRONT FACE
*********** start of CM CONS3R with identifier PRIMCOLL ***********
10, RMAX
Primary Collimator
2.1, ZMIN
7.83856, ZTHICK
2, NUM_NODE
2.1, 0.52359, 9.93856, 2.97796, 0.7, 0.01, 0, 0, 0,
```
************ start of CM FLATFILT with identifier FLATTENF ************

10.0, RMAX
Flattening Filter
9.93857, ZMIN
13, NUMBER OF LAYERS
3, 0.0356, # CONES, ZTHICK OF LAYER 1
0, 2.478, 10,
0.127, 2.487, 10,
3, 0.066, # CONES, ZTHICK OF LAYER 2
0.127, 2.487, 10,
0.254, 2.503, 10,
3, 0.1651, # CONES, ZTHICK OF LAYER 3
0.254, 2.503, 10,
0.508, 2.544, 10,
3, 0.1676, # CONES, ZTHICK OF LAYER 4
0.508, 2.544, 10,
0.762, 2.586, 10,
3, 0.1549, # CONES, ZTHICK OF LAYER 5
0.762, 2.586, 10,
1.016, 2.625, 10,
3, 0.1524, # CONES, ZTHICK OF LAYER 6
1.016, 2.625, 10,
1.27, 2.663, 10,
3, 0.1422, # CONES, ZTHICK OF LAYER 7
1.27, 2.663, 10,
1.524, 2.698, 10,
3, 0.1194, # CONES, ZTHICK OF LAYER 8
1.524, 2.698, 10,
1.778, 2.728, 10,
3, 0.0331, # CONES, ZTHICK OF LAYER 9
1.778, 2.728, 10,
1.855, 2.736, 10,
4, 0.071, # CONES, ZTHICK OF LAYER 10
1.855, 3.175, 6.033, 10,
2.032, 2.993, 6.033, 10,
4, 0.1194, # CONES, ZTHICK OF LAYER 11
2.032, 2.993, 6.033, 10,
2.286, 2.697, 6.033, 10,
4, 0.0203, # CONES, ZTHICK OF LAYER 12
2.286, 2.697, 6.033, 10,
2.438, 2.642, 6.033, 10,
2, 0.1143, # CONES, ZTHICK OF LAYER 13
6.033, 10,
6.033, 10,
0.7, 0.01, 0, 0,
PB700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
W700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
PB700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
P700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
P700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PB700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>W700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PB700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>W700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PB700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>W700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PB700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>W700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PB700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>W700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PB700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>W700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PB700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>W700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PB700ICRU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
AIR700ICRU 0.7, 0.01, 0, 0, PB700ICRU 0.7, 0.01, 0, 0, W700ICRU 0.7, 0.01, 0, 0, AIR700ICRU 0.7, 0.01, 0, 0, PB700ICRU 0.7, 0.01, 0, 0, W700ICRU 0.7, 0.01, 0, 0, AIR700ICRU

*********** start of CM FLATFILT with identifier CHAMBER ***********

7.75, RMAX
Ion chamber
11.35, ZMIN
35, NUMBER OF LAYERS
1, 0.001, # CONES, ZTHICK OF LAYER 1
7.75,
7.75,
1, 0.04318, # CONES, ZTHICK OF LAYER 2
7.75,
7.75,
1, 0.001, # CONES, ZTHICK OF LAYER 3
7.75,
7.75,
1, 0.46124, # CONES, ZTHICK OF LAYER 4
7.75,
7.75,
1, 0.00254, # CONES, ZTHICK OF LAYER 5
7.75,
7.75,
1, 0.00254, # CONES, ZTHICK OF LAYER 6
7.75,
7.75,
1, 0.00254, # CONES, ZTHICK OF LAYER 7
7.75,
7.75,
1, 0.2, # CONES, ZTHICK OF LAYER 8
7.75,
7.75,
1, 0.00254, # CONES, ZTHICK OF LAYER 9
7.75,
7.75,
1, 0.00254, # CONES, ZTHICK OF LAYER 10
7.75,
7.75,
1, 0.00254, # CONES, ZTHICK OF LAYER 11
7.75,
7.75,
1, 0.2, # CONES, ZTHICK OF LAYER 12
7.75,
7.75,
1, 0.00254, # CONES, ZTHICK OF LAYER 13
7.75,
7.75,
1, 0.00254, # CONES, ZTHICK OF LAYER 14
7.75,
7.75,
1, 0.00254, # CONES, ZTHICK OF LAYER 15
7.75,
7.75,
1, 0.46124, # CONES, ZTHICK OF LAYER 16
7.75,
7.75,
1, 0.001, # CONES, ZTHICK OF LAYER 17
7.75,
7.75,
1, 0.0381, # CONES, ZTHICK OF LAYER 18
7.75,
<table>
<thead>
<tr>
<th>Material</th>
<th>Thickness</th>
<th>Density</th>
<th>Density/Thickness Ratio</th>
<th>Porosity</th>
<th>Homogeneity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>MICA700</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AU700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>MICA700</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AU700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>W700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>MICA700</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AU700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CU700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>STEEL700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CU700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>MICA700</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AIR700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AU700ICRU</td>
<td>0.7</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AU700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AU700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
MICA700
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AU700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU

*********** start of CM MIRROR with identifier MIRROR ***********
7.0, RMAX
Mylar mirror
17.585, 8.03, ZMIN, ZTHICK
5.734, -5.734, XFMIN, XBMIN
1, # LAYERS
0.00508, thickness of layer 1
0.0, 0.0, 0, 0,
MYLAR700ICRU
0.0, 0.0, 0, 0,
AIR700ICRU
0.0, 0.0, 0, 0,
AIR700ICRU

*********** start of CM JAWS with identifier SECJAWS ***********
10, RMAX
Secondary collimators set to produce a 10x10 field at 100 cm SSD
2, # PAIRED BARS OR JAWS
X
29.0, 38.0, 2.900, 3.800, -2.900, -3.800,
Y
38.5, 47.3, 3.8500, 4.7300, -3.8500, -4.7300,
*********** start of CM DYNVMLC with identifier MLC ***********
20, RMAX
Varian Millenium MLC
1, 3, 0, ORIENT, NGROUP, MODE
47.67, ZMIN
6.7, ZTHICK
0.47, 0.04, 0.1354, 0.3673, 0.1396, 47.713, 47.996, 50.987, 51.195, 52.42855, 52.75605, 1.65, 54.0104, 54.275,
0.231467, 0.04, 0.04, 0.1054, 0.1454, 0.1454, 47.765, 48.0296, 48.99895, 49.20395, 1.65, 51.045, 51.047, 54.047, 54.166,
0.233, 0.04, 0.04, 0.0754, 0.1405, 0.1405, 47.875, 47.994, 51.094, 51.195, 52.84355, 53.04105, 1.65, 54.0104, 54.275,
10, 1
40, 2
10, 1
-9.533, START
0.0057, LEAFGAP
0, ENDTYPE
8, ZFOCUS or RADIUS of leaf ends
0, ZFOCUS of leaf sides
-10, 10, 60
0.7, 0.01, 0, 0,
AIR700ICRU
0.7, 0.01, 0, 0, 0,
W700ICRU
0.7, 0.01, 0, 0,
AIR700ICRU
*********** start of CM SLABS with identifier AIRSLAB ***********
30, RMAX
PMMA reticle + end of accel at 60cm from source
2, NSLABS
59, ZMIN
0.03, 0.7, 0.01, 0, 0, 0
PMMA700ICRU
0.97, 0.7, 0.01, 1, 0, 0
AIR700ICRU
*********************end of all CMs**********************************
:Start MC Transport Parameter:
Global ECUT= 0.7
Global PCUT= 0.01
Global SMAX= 5
ESTEPE= 0.25
XIMAX= 0.5
Boundary crossing algorithm= EXACT
Skin depth for BCA= 0
Electron-step algorithm= PRESTA-II
Spin effects= On
Brems angular sampling= Simple
Brems cross sections= BH
Bound Compton scattering= Off
Compton cross sections= default
Pair angular sampling= Simple
Pair cross sections= BH
Photoelectron angular sampling= Off
Rayleigh scattering= Off
Atomic relaxations= Off
Electron impact ionization= Off
Photon cross sections= si
Photon cross-sections output= Off
:Stop MC Transport Parameter:

*******************************************************************************

147
E.2 EPID_20x20.egsinp

This is an example DOSXYZnrc input file which uses the phase-space file results from Appendix E.1 to produce a dose array at the vEPID.

```
Epid 20x20, 1024x1024 0.04cm pixels at 150cm ssd  #!GUI1.0
2
H2O700ICRU
AIR700ICRU
0.7, 0.1, 0, 0, 0
-3, -3, 1, 1
-75
54.52, 1
0.04, 1024
54.52, 1
-75
54.52, 1
0.04, 1024
54.52, 1
0
0.1
0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0
2, 2, 0, 0, 0, 180, 0, 90, 0, 0, 0, 0, 0, 0
2, 0, 2, 20, 1, 20, 0, 100
/home/beamuser/Epid_char/20x20/6MVmohen_Epid_20x20.egsphsp1
4000000000, 0, 24, 4948, 18984, 40, 1, 0, 2, 1, 0.1, 0, 0, 0, 1, 0
#**************************************************************************
:Start MC Transport Parameter:
Global ECUT= 0.7
Global PCUT= 0.1
Global SMAX= 5
ESTEPE= 0.25
XIMAX= 0.5
Boundary crossing algorithm= PRESTA-I
Skin depth for BCA= 0
Electron-step algorithm= PRESTA-II
Spin effects= On
Brems angular sampling= Simple
Brems cross sections= BH
Bound Compton scattering= Off
Compton cross sections= default
Pair angular sampling= Simple
Pair cross sections= BH
Photoelectron angular sampling= Off
Rayleigh scattering= Off
Atomic relaxations= Off
Electron impact ionization= Off
Photon cross sections= si
Photon cross-sections output= Off
:Stop MC Transport Parameter:
#**************************************************************************
```
E.3 bin_fluence.py

This program creates a 2d array of fluence intensity in number of particles versus the position of the particle over the usable range of the vEPID. These are stored as a cPickled container of the X and Y points and the array of intensities. This calculation is performed based on a SPD of 150cm and SAD of 100cm.

```python
#!/usr/bin/python
import os
import sys
import numpy
import cPickle
import gzip
import struct
from signal import signal, SIGINT, SIG_IGN
from time import time
from multiprocessing import Process, JoinableQueue, Queue, cpu_count, active_children
import Queue as stdQueue

# MODE0 PHSP (no Z-LAST)
MODE0_dt = numpy.dtype({'names': ['LATCH', 'E', 'X', 'Y', 'U', 'V', 'WT'],
   'formats': ['u4', 'f4', 'f4', 'f4', 'f4', 'f4', 'f4']})

# This def lets us read in only small chunks of the file as we need it, rather than loading the whole phase space into memory (could be gigs...)
def buf_data_from_file(infile, buf_size):
    while True:
        data = infile.read(buf_size)
        if not data:
            break
        yield data

def parse_queue(def_bins, def_range, dtype, inqueue, outqueue):
    signal(SIGINT, SIG_IGN)
    hist = numpy.zeros((def_bins, def_bins))
    particle_array = None
    for data in iter(inqueue.get, 'DONE'):
        if particle_array is None or len(data) != particle_array.nbytes:
            # We either haven't set up particle_array, or we are at the tail end of the data
            # particle_array should be a list of floats, each 4 bytes long, so len(data)/4 floats.
            particle_array = numpy.fromstring(data, dtype=dtype)
        else:
            # The particle array is setup, and we have data in it. We will just replace the data string with the new one.
            particle_array.data[:] = data

        # Breakdown: unpack the data to a list of floats, then create array from that list
        # then reshape the array into an array of <n_cols> columns. Use column 3 and 4 to create a histogram which is then put into the out queue. This saves tons of memory
        # over iterating through the list.
        not_all_zero = ((particle_array['X'] != 0) | (particle_array['Y'] != 0))
        particle_array_view = particle_array[not_all_zero]
        hist += numpy.histogram2d(particle_array_view['X'], particle_array_view['Y'], def_bins, def_range)[0]
```

149
#hist += numpy.histogram2d(particle_array['X'], particle_array['Y'], def_bins, def_range)[0]

    inqueue.task_done()
else:
    inqueue.task_done()

outqueue.put(hist)

# A quick function to generate 4 digit precis. SI suffixes
def si(value):
    value_str = '%4.4g' % value
    if 'e' in value_str:
        SI = dict({3:'k', 6:'M', 9:'G', 12:'T', 15:'P'})
        exp = int(value_str.split('e')[1]) % 3
        o_str = '%s %s' % (float(value_str.split('e')[0])*10**mod, exp)
    else:
        o_str = '%s %s' % value_str
    return o_str.rjust(7)

if __name__ == '__main__':
    try:
        NCPU = cpu_count()
    except NotImplementedError as e:
        NCPU = 4

display=False
benchmark=True
output=True

def_bins=1024
def_min=20.48
def_max=20.48
def_points=numpy.linspace(def_min-def_min/def_bins,def_max-def_max/(def_bins),def_bins)

def_readblock=100000

# PHSP header format: char MODE_PW[5]; int NPPHSP, NPHOTPHSP; float EMAXPHSP, EKMINPHSPE, NINCPHSP; pad[3]

fmt_PHSPhead='=5s 2I 3f '}

f_readfile = None
f_writefile = None

if len(sys.argv) < 2:
    sys.stderr.write( "Usage: %s input_points [output_hist].\n" % sys.argv[0])
    sys.exit(1)

fn_readfile = sys.argv[1]

# Param 2 is output filename, or we assign it based on input filename.
if len(sys.argv) == 3:
    fn_writefile = sys.argv[2]
else:
    fn_readfile_basename = os.path.basename(fn_readfile)

    # Strip off ".gz" from the basename, if it is there.
    if fn_readfilebasename.endwith(".gz",-3):
        fn_readfilebasename = fn_readfilebasename.rpartition(".")[0]

    # Strip off ".egphsp?" extension, if there.
    if fn_readfilebasename.endwith(".egphsp",0,-1):
        fn_writefilebasename = fn_readfilebasename.rpartition(".")[0]
    else:
        fn_writefilebasename = fn_readfilebasename

    fn_writefile = os.path.dirname(fn_readfile) + fn_writefilebasename + ".egshist.gz"
# Try to open the input file for reading and abort if we cannot.
++ Also checks to see if the file looks like a gzip file. If it seems like it is
++ we load it as such, and read the first 5 characters to test this. If it is a
++ false positive, the data was corrupt anyway (first 2 characters should be ASCII 'MO')
mode_string = ""
try:
    f_readfile = open(fn_readfile, 'rb')
    if f_readfile.read(2) == '\x1f\x8b':
        f_readfile = gzip.GzipFile(fileobj=f_readfile)
        mode_string = f_readfile.read(5)
        f_readfile.rewind()
    else:
        f_readfile.seek(0)
        mode_string = f_readfile.read(5)
        f_readfile.seek(0)
except IOError as e:
    sys.stderr.write("Could not open %s, it may not exist. (Error: %s)\n" % (fn_readfile, e.strerror ))
    sys.exit(1)

if output:
    # Try to open the output file for reading, if we can, it already exists and we abort.
    # Otherwise, so we should be safe to open for writing.
    try:
        open(fn_writefile, 'r')
    except IOError as e:
        if e.errno == 2: # File not found error, we can continue.
            try:
                f_writefile = gzip.open(fn_writefile, 'wb')
            except IOErrror as wr_e:
                sys.stderr.write("Could not open %s for writing, do you have permission?\n" % (fn_writefile, wr_e.strerror ))
                sys.exit(2)
        else:
            raise
    else:
        raise
else:
    sys.stderr.write("Error: %s already exists, aborting.\n" % fn_writefile)
    sys.exit(2)

mode_size = None
if mode_string[4] == '0': # MODE0 file, use MODE0_dt (UINT LATCH; float ESHORT,X,Y,U,V,WT)
    mode_dtype = MODE0_dt
    mode_size = 28
elif mode_string[4] == '2': # MODE2 file, use MIDE2_dt (UINT LATCH; float ESHORT,X,Y,U,V,WT,ZLAST) - 8 * 4 Bytes
    mode_dtype = MIDE2_dt
    mode_size = 32
else:
    sys.stderr.write('Header format corrupted, or unusable mode (%s). Aborting.\n','
mode_string)
    sys.exit(1)

header = f_readfile.read(mode_size)
if len(header) <> mode_size:
    sys.stderr.write("Error: input file %s does not have a complete header.\n" % fn_readfile)
    sys.exit(1)
else:
    # Add the appropriate padding to get up to mode_size length
    fmt_PHSPhead = fmt_PHSPhead + str(mode_size - struct.calcsize(fmt_PHSPhead)) + 'x'

    # clean way of doing above, all in a single list
    header_unpacked = struct.unpack(fmt_PHSPhead, header)

    print ('MODE: %s, np: %i, np_phot: %i, ekmax: %f, ekmin: %f, ninc: %.0f' % header_unpacked)

readblock_size = def_readblock * mode_size

151
hist = numpy.zeros((def_bins, def_bins))

task_queue = JoinableQueue(NCPU)
results_queue = Queue(NCPU)

# Start the worker processes.
for i in range(NCPU):
    Process(target=parses_queue, args=(def_bins, def_range, mode_dtype, task_queue, results_queue)).start()

points = 0
start = time()
rate = []
task_queue.put('DONE')

try:
    for data in buf_data_from_file(f_readfile, readblock_size):
        points += len(data) / mode_size

        sys.stdout.write(’%i / %i (%.1f%%) (%spts/s)’ % (points, header_unpacked[1],
            (points*100/header_unpacked[1]), si(points/(time() - start))))
        task_queue.put(data)

    sys.stdout.flush()

except KeyboardInterrupt:
    sys.stderr.write(’
Aborting!
’)
    sys.stderr.flush()

    for child in active_children():
        child.terminate()
    sys.exit(3)

if benchmark:
    end = time()
    f_readfile.close()  # Close file

    if benchmark:
        print(’total time: %f s, average speed: %spts/s’ % ((end-start), si(points / (end - start))))

        print(’Average value above 4: %.2f. Std Dev: %.2f’ % (numpy.average(hist[hist > 4]),
            numpy.std(hist[hist > 4])))

        if display:
            import matplotlib.pyplot as plt
            hist[hist < 4] = 0
            plt.imshow(hist,
                extent=[def_min, def_max, def_min, def_max], interpolation=’nearest’)
            plt.colorbar()
            plt.show()

        if output:
            cPickle.dump((def_points, def_points, hist), f_writefile,
                cPickle.HIGHEST_PROTOCOL)
    f_writefile.close()
E.4 bin_fluence_at60.py

This program creates a 2d array of fluence intensity in number of particles versus the position of the particle over the usable range of the vEPID. These are stored as a cPickled container of the X and Y points and the array of intensities. This calculation is performed based on a SPD of 60cm and SAD of 100cm.

```python
#!/usr/bin/python
import os
import sys
import numpy
import cPickle
import gzip
import struct
from signal import signal, SIGINT, SIG_IGN
from time import time
from multiprocessing import Process, JoinableQueue, Queue, cpu_count, active_children
import Queue as stdQueue

# MODE0 PHSP (no Z-LAST)
MODE0_dt = numpy.dtype({'names': ['LATCH', 'E', 'X', 'Y', 'U', 'V', 'WT'],
                       'formats': ['u4', 'f4', 'f4', 'f4', 'f4', 'f4', 'f4']})
MODEL2_dt = numpy.dtype({'names': ['LATCH', 'E', 'X', 'Y', 'U', 'V', 'WT', 'ZL'],
                         'formats': ['u4', 'f4', 'f4', 'f4', 'f4', 'f4', 'f4', 'f4']})

# This def lets us read in only small chunks of the file as we need it, rather than loading the whole phase space into memory (could be gigs...)
def buf_data_from_file(infile, buf_size):
    while True:
        data = infile.read(buf_size)
        if not data:
            break
        yield data

def parse_queue(def_bins, def_range, dtype, inqueue, outqueue):
    signal(SIGINT, SIG_IGN)
    hist = numpy.zeros((def_bins, def_bins))
    particle_array = None
    for data in iter(inqueue.get, 'DONE'):
        if particle_array is None or len(data) != particle_array.nbytes:
            # We either haven't set up particle_array, or we are at the tail end of the data
            # particle_array should be a list of floats, each 4 bytes long, so len(data)/4 floats.
            particle_array = numpy.fromstring(data, dtype = dtype)
        else:
            # The particle array is setup, and we have data in it. We will just replace the data string with the new one.
            particle_array.data[:] = data

    # Breakdown: unpack the data to a list of floats, then create array from that list
    # then reshape the array into an array of <n_cols> columns. Use column 3 and 4 to create a histogram which is then put into the out queue. This saves tons of memory
    # over iterating through the list.
```
not_all_zero = ((particle_array['X'] != 0) | (particle_array['Y'] != 0))

particle_array_view = particle_array[not_all_zero]

hist += numpy.histogram2d(particle_array_view['X'], particle_array_view['Y'], def_bins, def_range)[0]

inqueue.task_done()
else:
inqueue.task_done()

outqueue.put(hist)

# A quick function to generate 4 digit precis. SI suffixes

def si_value(value):
    value_str = '%4.4g' % value
    if 'e' in value_str:
        SI = dict({3:'k', 6:'M', 9:'G', 12:'T', 15:'P'})
        mod = int(value_str.split('e')[1]) % 3
        exp = SI[int(value_str.split('e')[1]) - mod]
        o_str = '%s %s' % ('%.1f' % (float(value_str.split('e')[0]) * 10 ** mod), exp)
    else:
        o_str = '%s ' % value_str
    return o_str.rjust(7)

if __name__ == '__main__':
    try:
        NCPU = cpu_count()
    except NotImplementedError as e:
        NCPU = 4

display = False
benchmark = True
output = True

def_bins = 1024
def_min = -20.48
def_max = 20.48
def_scale = (0.6 / 1.5)
def_min *= def_scale

def_range = ([def_min, def_max], [def_min, def_max])
def_points = numpy.linspace(def_min - def_min / (def_bins), def_max - def_max / (def_bins), def_bins)

def_readblock = 100000

# PHSP header format: char MODE_RW[5]; int NPPHSP, NPHOTPHSP; float EMAXPHSP, EKMINPHSPE, NINCPHSP; pad[3]

fmt_PHSPhead = '-5s 2I 3f '

f_readfile = None
f_writefile = None

if len(sys.argv) < 2:
sys.stderr.write("Usage: %s input_points [output_hist].\n" % sys.argv[0])
sys.exit(1)

fn_readfile = sys.argv[1]

# Param 2 is output filename, or we assign it based on input filename.
if len(sys.argv) == 3:
    fn_writefile = sys.argv[2]
else:
    fn_readfile_basename = os.path.basename(fn_readfile)
    # Strip off ".gz" from the basename, if it is there.
    if fn_readfile_basename.endswith(".gz", -3):
        fn_readfile_basename = fn_readfile_basename.rpartition(".")[0]
    # Strip off ".egsphsp?" extension, if there.
    if fn_readfile_basename.endswith(".egsphsp", 0, -1):
fn_writefile_basename = fn_readfile_basename.rpartition(".")[0]
else:
    fn_writefile_basename = fn_readfile_basename

fn_writefile = os.path.dirname(fn_readfile) + fn_writefile_basename + ".egahist_60.gz"

# Try to open the input file for reading and abort if we cannot.
# Also checks to see if the file looks like a gzip file. If it seems like it is
# we load it as such, and read the first 5 characters to test this. If it is a
# false positive, the data was corrupt anyway (first 2 characters should be ASCII 'MO')
mode_string = ""
try:
    f_readfile = open(fn_readfile, 'rb')
    if f_readfile.read(2) == '\x1f\x8b':
        f_readfile = gzip.GzipFile(fileobj = f_readfile)
        mode_string = f_readfile.read(5)
        f_readfile.rewind()
    else:
        f_readfile.seek(0)
        mode_string = f_readfile.read(5)
        f_readfile.seek(0)
except IOError as e:
    sys.stderr.write("Could not open %s, it may not exist. (Error: %s)
                          ( fn_readfile, e.strerror )
                          )
    sys.exit(1)
if output:
    # Try to open the output file for writing, if we can, it already exists and we abort.
    # Otherwise, so we should be safe to open for writing.
    try:
        open(fn_writefile, 'w')
    except IOError as e:
        if e.errno == 2: # File not found error, we can continue.
            try:
                f_writefile = gzip.open(fn_writefile, 'wb')
            except IOError as wr_e:
                sys.stderr.write("Could not open %s for writing, do you have permission?
                            (Error: %s)
                          % ( fn_writefile, wr_e.strerror ))
                sys.exit(2)
        else:
            raise
    else:
else:
    sys.stderr.write("Error: %s already exists, aborting.
                          
                          )
    sys.exit(2)

mode_size = None
if mode_string[4] == '0': # MODE0 file, use MODE0_dt (UINT LATCH; float ESHORT,X,Y,U,V,WT)
    mode_dtype = MODE0_dt
    mode_size = 28
elif mode_string[4] == '2': # MODE2 file, use MIDE2_dt (UINT LATCH; float
    mode_dtype = MODE2_dt
    mode_size = 32
else:
    sys.stderr.write('Header format corrupted, or unusable mode (%s). Aborting.
                          
                          )
    sys.exit(1)

header = f_readfile.read(mode_size)
if len(header) <> mode_size:
    sys.stderr.write("Error: input file %s does not have a complete header.
                          
                          )
    sys.exit(1)
else:
    # Add the appropriate padding to get up to mode_size length
    fmt_PHSPhead = fmt_PHSPhead + str(mode_size - struct.calcsize(fmt_PHSPhead)) + 'x'
    #MODE_RW,NPPHSP,NPHOTPHSP,ERMXPHP,EkMINPHSP,EkNCPHSP = struct.unpack(fmt_PHSPhead,header)
# cleaner way of doing above, all in a single list
header_unpacked = struct.unpack(fmt_PHSPhead, header)
print ("MODE: %s, np: %i, np_photo: %i, ekmax: %f, ekminE: %f, ninc: %.0f" % header_unpacked)
readblock_size = def_readblock * mode_size

hist = numpy.zeros((def_bins, def_bins))
task_queue = JoinableQueue(NCPU)
results_queue = Queue(NCPU)

# Start the worker processes.
for i in range(NCPU):
    Process(target = parse_queue, args = (def_bins, def_range, mode_dtype, task_queue, results_queue)).start()

points = 0
start = time()
rate = []
try:
    data = buf_data_from_file(f_readfile, readblock_size):
        points += len(data) / mode_size
        sys.stdout.write(\r%i / %i (%.1f%%) (%spts/s)' %
            (points, header_unpacked[1], (points*100 header_unpacked[1]), si(
            points/(time() - start)))
        sys.stdout.flush()
        task_queue.put(data)

    else: # EOF reached!
        sys.stdout.write(\r%i / %i (%.1f%%) (%spts/s) Done! Flushing Queues...' %
            (points, header_unpacked[1], (points*100 header_unpacked[1]), si(
            points/(time() - start)))
        sys.stdout.flush()
        for i in range(NCPU):
            task_queue.put(\'DONE\'
            sys.stdout.write(\'\%i\' % (i+1))
            sys.stdout.flush()

        task_queue.join()
        task_queue.close()

    for i in range(NCPU):
        hist += results_queue.get()
        sys.stdout.write(\'.\'
        sys.stdout.flush()

        results_queue.close()
        print \'Done!\'
except KeyboardInterrupt:
    sys.stderr.write(\nAborting!\n)

    sys.stderr.flush()
    for child in active_children():
        child.terminate()
    sys.exit(3)

if benchmark: end = time()
f_readfile.close() # Close file

if benchmark:
    print \'total time: %f s, average speed: %spts/s \((end-start), si(points / (end - start)))\'
    print \'Average value above 4: %.2f. Std Dev: %.2f\' % (numpy.average(hist[hist > 4]),
        numpy.std(hist[hist > 4]))

    if display:
        import matplotlib.pyplot as plt
E.5 bin_3ddose.py

This program converts the defined 2d array of dose into the same format as used in the bin_fluence.py program. These are stored as a cPickled container of the X and Y points and the array of intensities. This calculation is performed based on a SPD of 150cm and SAD of 100cm.

```python
#!/usr/bin/python
import os
import sys
import numpy
import matplotlib.pyplot as plt
import cPickle
import gzip
import struct
from time import time

# 3ddose file format: all data in space seperated ascii
# Line #: Data
#      1: nx ny nz
#      2: x voxel boundaries (nx+1 values)
#      3: y voxel boundaries (ny+1 values)
#      4: z voxel boundaries (nx+1 values)
#      5: dose values array ([0,0,0] [1,0,0] ... [nx,0,0] [0,1,0] [1,1,0] ... [nx,nz,nz])
#      6: error values array (same format as dose values array)

# In this program, we will require nz = 1
# Data will be read using numpy.fromstring. Line 1 must be integers, the rest #+ are allowed to be float.

if __name__ == '__main__':

display=False
benchmark=True
output=True

f_readfile = None
f_writefile = None

if len(sys.argv) < 2:
    sys.stderr.write("Usage: %s input_3ddose [output_hist].\n" % sys.argv[0])
    sys.exit(1)

fn_readfile = sys.argv[1]
```

```python
hist[hist < 4] = 0
plt.imshow(hist, extent = [def_min,def_max,def_min,def_max], interpolation = 'nearest')
plt.colorbar()
plt.show()

if output:
cPickle.dump((def_points,def_points,hist),f_writefile,cPickle.HIGHEST_PROTOCOL)
f_writefile.close()
```
# Lines 2-4 are the x,y,z boundaries.
X_edges = numpy.fromstring(f_readfile.readline(), sep=' ', dtype=int)
if nz < 1:
sys.stderr.write("Error: input file %s does not have the expected number of z slices (%i).\n" % (fn_readfile, nz))
sys.exit(1)
except ValueError as e:
sys.stderr.write("First line has some formatting error, aborting. (Error: %s)\n" % e.stderr)
sys.exit(1)

# Param 2 is output filename, or we assign it based on input filename.
if len(sys.argv) == 3:
    fn_writefile = sys.argv[2]
else:
    fn_readfile_basename = os.path.basename(fn_readfile)
    # Strip off ".gz" from the basename, if it is there.
    if fn_readfile_basename.endswith(".gz", -3):
        fn_readfile_basename = fn_readfile.basename.rpartition(".")[0]
    # Strip off ".3ddose" extention, if there.
    if fn_readfile_basename.endswith(".3ddose"):
        fn_writefile_basename = fn_readfile.basename.rpartition(".")[0]
    else:
        fn_writefile_basename = fn_readfile.basename
    fn_writefile = os.path.dirname(fn_readfile) + fn_writefile_basename + ".dose_hist.gz"

try: # Try to open the input file for reading and abort if we cannot.
    # Also checks to see if the file looks like a gzip file. If it seems like it is
    # we load it as such, and read the first 5 characters to test this. If it is a
    # false positive, the data was corrupt anyway (first 2 characters should be normal text)
    f_readfile = open(fn_readfile, 'rb')
    if (f_readfile.read(2) == '\x1f\x8b'):
        f_readfile = gzip.GzipFile(fileobj=f_readfile)
        f_readfile.rewind()
        f_readfile.read(5)
        f_readfile.rewind()
    else:
        f_readfile.seek(0)
except IOError as e:
    sys.stderr.write("Could not open %s, it may not exist. (Error: %s)\n" % (fn_readfile, e.strerror))
sys.exit(1)

if output:
    # Try to open the output file for reading, if we can, it already exists and we abort.
    # Otherwise, so we should be safe to open for writing.
    try:
        open(fn_writefile, 'r')
    except IOError as e:
        if e.errno == 2: # File not found error, we can continue.
            try:
                f_writefile = gzip.open(fn_writefile, 'wb')
            except IOError as wr_e:
                sys.stderr.write("Could not open %s for writing, do you have permission? (Error: %s)\n" % (fn_writefile, wr_e.strerror))
                sys.exit(2)
        else:
            raise
    else:
        sys.stderr.write("Error: %s already exists, aborting.\n" % fn_writefile)
        sys.exit(2)

try:
    [nx, ny, nz] = numpy.fromstring(f_readfile.readline(), sep=' ', dtype=int)
    if nz < 1:
        sys.stderr.write("Error: input file %s does not have the expected number of z slices (%i).\n" % (fn_readfile, nz))
        sys.exit(1)
except ValueError as e:
    sys.stderr.write("First line has some formatting error, aborting. (Error: %s)\n" % e.stderr)
    sys.exit(1)
E.6 combine_hist.py

This program condenses a dose histogram and fluence histogram into a format which may be read by the deconvolution parameter solver program.

```python
#!/usr/bin/python
import os
import sys
import numpy
import matplotlib.pyplot as plt
import cPickle
import gzip
from time import time

# Line 5 is the dose, 6 is the dose error. Both are shaped as [nx, ny, nz], but nz must be 1.
Dose_Array = numpy.fromstring(f_readfile.readline(), sep=' ', dtype=float)
Dose_Array.shape = [nx, ny]

Dose_Error_Array = numpy.fromstring(f_readfile.readline(), sep=' ', dtype=float)
Dose_Error_Array.shape = [nx, ny]

# Finally, we want the centers of the bins, not the edges. (average of the edges)
X_Points = (X_edges[1:] + X_edges[:-1]) / 2
Y_Points = (Y_edges[1:] + Y_edges[:-1]) / 2

f_readfile.close()  # Close file

if output:
    print "Outputing to: %s" % fn_writefile
    cPickle.dump((X_Points[1:-1], Y_Points[1:-1], Dose_Array[1:-1, 1:-1], Dose_Error_Array[1:-1, 1:-1]), f_writefile, cPickle.HIGHEST_PROTOCOL)
    f_writefile.close()

if display:
    extents=[min(X_Points[1:-1]), max(X_Points[1:-1]), min(Y_Points[1:-1]), max(Y_Points[1:-1])]

    Dose_Error_Array[Dose_Array < (0.9* max(Dose_Array.flatten()))] = 0
    print "%s: Average error value in region > 20% max dose: %f" % (fn_readfile, numpy.average(Dose_Error_Array[Dose_Array > (0.2* max(Dose_Array.flatten()))]))

    Dose_Error_Array[Dose_Array < (0.8* max(Dose_Array)) = 0
    print "%s: Average error value in region > 80% max dose: %f" % (fn_readfile, numpy.average(Dose_Error_Array[Dose_Array > (0.05* max(Dose_Array))]))

plt.figure(1)
plt.title("Dose Array")
plt.imshow(Dose_Array[1:-1, 1:-1], extent=extents, interpolation='nearest')
plt.colorbar()
plt.figure(2)
plt.title("Dose Error Array")
plt.imshow(Dose_Error_Array[1:-1, 1:-1], extent=extents, interpolation='nearest')
plt.colorbar()
plt.show()
```
if __name__ == '__main__':

    display=False
    output=True

    f_fluence = None
    f_writefile = None

    if len(sys.argv) < 4:
        sys.stderr.write("Usage: %s label input_fluence_hist input_dose_hist [outfile]\n" % sys.argv[0])
        sys.exit(1)

        label = sys.argv[1]
        fn_fluence = sys.argv[2]
        fn_dose = sys.argv[3]

        # Param 2 is output filename, or we assign it based on input filename.
        if len(sys.argv) == 5:
            fn_writefile = sys.argv[4]
        else:
            fn_writefile = label + ".fl_dose_hist.gz"

        # Try to open the input file for reading and abort if we cannot.
        # Also checks to see if the file looks like a gzip file.  If it seems like it is
        # we load it as such, and read the first 5 characters to test this.  If it is a
        # false positive, the data was corrupt anyway (first 2 characters should be normal text)
        try:
            f_fluence = open(fn_fluence, 'rb')
            if (f_fluence.read(2) == '\x1f\x8b'):
                f_fluence = gzip.GzipFile(fileobj = f_fluence)
                f_fluence.rewind()
                f_fluence.read(5)
            else:
                f_fluence.seek(0)
            f_dose = open(fn_dose, 'rb')
            if (f_dose.read(2) == '\x1f\x8b'):
                f_dose = gzip.GzipFile(fileobj = f_dose)
                f_dose.rewind()
                f_dose.read(5)
            else:
                f_dose.seek(0)
        except IOError as e:
            sys.stderr.write("Could not open %s or %s, it may not exist. (Error: %s)\n" % (fn_fluence, fn_dose, e.strerror))
            sys.exit(2)
        else:
            raise
    else:
        sys.stderr.write("Error: %s already exists, aborting.\n" % fn_writefile)
        sys.exit(2)

if output:
    # Try to open the output file for reading, if we can, it already exists and we abort.
    # Otherwise, so we should be safe to open for writing.
    try:
        open(fn_writefile, 'r')
    except IOError as e:
        if e.errno == 2:  # File not found error, we can continue.
            try:
                f_writefile = gzip.open(fn_writefile, 'wb')
            except IOError as wr_e:
                sys.stderr.write("Could not open %s for writing, do you have permission? (Error: %s)\n" % (fn_writefile, wr_e.strerror))
                sys.exit(2)
            else:
                raise
        else:
            sys.stderr.write("Error: %s already exists, aborting.\n" % fn_writefile)
            sys.exit(2)
try:
    # Fluence histogram file format: ( X_Points, Y_Points, Fluence_Array )
    fluence = cPickle.load(f_fluence)
    # Dose histogram file format: ( X_Points, Y_Points, Dose_Array, Dose_Error_Array )
    dose = cPickle.load(f_dose)
except:
    # Remove the writefile on error in reading data in.
    sys.stderr.write("Data error of some kind. Aborting.")
    f_dose.close()
    f_fluence.close()
    f_writefile.close()
raise
finally:
    f_dose.close()
    f_fluence.close()

if output:
    print "Outputing to: %s" % fn_writefile
    cPickle.dump(((label, fluence[2], dose[2]),), f_writefile, cPickle.HIGHEST_PROTOCOL)
    f_writefile.close()

if display:
    fl_extents = [min(fluence[0]), max(fluence[0]), min(fluence[1]), max(fluence[1])]
    dose_extents = [min(dose[0]), max(dose[0]), min(dose[1]), max(dose[1])]
    dose[3][dose[3] < (0.2* max(dose[2]).flatten())] = 0
    print "Average error value in region > 20% max dose: %f" % numpy.average(dose[3][dose[3] > 0])
    plt.figure(1)
    plt.title('Fluence Array')
    plt.imshow(fluence[2], extent=fl_extents, interpolation='nearest')
    plt.colorbar()
    plt.figure(2)
    plt.title('Dose Array')
    plt.imshow(dose[2], extent=dose_extents, interpolation='nearest')
    plt.colorbar()
    plt.figure(3)
    plt.title('Dose Error Array')
    plt.imshow(dose[3], extent=dose_extents, interpolation='nearest')
    plt.colorbar()
    plt.show()

E.7 hist_deconvolution.py

This program uses the exponential deconvolution algorithm to deconvolve fluence from EPID detected dose. It is designed to operate as a library as well as a stand alone system for use in the parameter solver.

#!/usr/bin/python

""
FFT Deconvolution using a set list of exponentials.
""
import numpy
from copy import deepcopy
# Deconvolve using a set of 5 exponentials.
# First we will FFT, fun our exponentials, then invFFT.

# The function will be an array generated from the radial fourier distance (cycles/cm) and an
# Array of the Ai and Bi values of the form deconv_coefs = [ [A1, B1], ..., [A5, B5] ]
# We use the pixel spacing of the array (self.pixel_size; assumed to be 0.04 cm)
# radial distance to each point in an array. Then we apply the appropriate function
# to this array using the Deconv_param array and the simple equation.

# PSF defined in Renner, et. al.
#    n  {                (2*pi)*bi     
# K(q) = SUM { ai * ------------------------------ 
#    i  {       
#          [(2*pi*q)**2 + bi**2]**(3/2)

def psf_kernel_func(Q, deconv_coefs):
    
    Return the PSF value at frequency domain radial position q for given deconv_coefs.

def set_pixel_size(in_pixel_size, recompute=True):
    
    The physical size of each pixel in the image. Default: PIXEL_SIZE

class ExpDeconvolution:
    
    Class for deconvolution using the psf_kernel_func.

    Attributes:
    deconv_coefs -- The deconvolution kernel coefficient pairs Ai, Bi as list of 2-tuples: [ [A0, B0], ... [An, Bn] ]

    Functions:
    _init_(in_deconv_coefs=deconv_coefs, in_pixel_size=PIXEL_SIZE, in_array_size=ARRAY_SIZE)
    set_deconv_coefs(in_deconv_coefs, recompute=True)
    set_array_size(in_array_size, recompute=True)
    compute_array(force_recompute=False)
    deconvolve(input_data)
    read_deconv_coefs_file(params_file='deconv_coefs.conf')

    PIXEL_SIZE = 0.04
    ARRAY_SIZE = (1024, 1024)

    deconv_coefs = []
    
    array_computed = False

    def _init_(self, in_deconv_coefs=None, in_pixel_size=PIXEL_SIZE, in_array_size=ARRAY_SIZE):
        Initialize the class, setting the parameters if passed.
Keyword arguments:
- `in_deconv_coefs` -- The deconvolution kernel coefficient pairs Aᵢ, Bᵢ as list of 2-tuples: \([A₀, B₀], ..., [Aₙ, Bₙ]\).
- `in_pixel_size` -- The physical size of each pixel in the image. Default: PIXEL_SIZE.
- `in_array_size` -- A 2-tuple containing the X and Y size in pixels expected. Default: ARRAY_SIZE.

If the deconv_coefs are set, compute the array as well.

```python
self.pixel_size = in_pixel_size
"""The physical size of each pixel."""
self.array_size = in_array_size
"""The dimensions of the image in pixels."""

if in_deconv_coefs:
    # Create the initial deconv array.
    self.set_deconv_coefs(in_deconv_coefs)
```

```python
def set_deconv_coefs(self, in_deconv_coefs, recompute=True):
    """Set the coefficients of the deconvolution kernel.
    If the passed values are different, it will recompute the deconvolution array by default.
    To override this behavior set 'recompute' to False
    Keyword arguments:
    in_deconv_coefs -- See __init__, required.
    recompute -- bool: Recompute the array if values have changed.
    \""\n    if self.deconv_coefs != in_deconv_coefs:
        self.deconv_coefs = deepcopy(in_deconv_coefs)  # Make sure we make a copy of the coefs.
        self.array_computed = False
    # Only recompute if 'recompute' is True
    if recompute: self.compute_array()
```

```python
def set_array_size(self, in_array_size, recompute=True):
    """Set the expected size of the input array.
    The deconvolution array is recomputed by default.
    To override this behavior set 'recompute' to False
    Keyword arguments:
    in_array_size -- See __init__, required.
    recompute -- bool: Recompute the array if values have changed.
    \""\n    if self.array_size != in_array_size:
        self.array_size = in_array_size
        self.array_computed = False
    if self.deconv_coefs and recompute:
        self.compute_array()
```

```python
def set_pixel_size(self, in_pixel_size, recompute=True):
    """Set the physical size of each pixel.
    The deconvolution array is recomputed by default.
    To override this behavior set 'recompute' to False
    Keyword arguments:
    in_pixel_size -- See __init__, required.
    recompute -- bool: Recompute the array if values have changed.
    \""\n    if self.pixel_size != in_pixel_size:
        self.pixel_size = in_pixel_size
        self.array_computed = False
    if self.deconv_coefs and recompute:
        self.compute_array()
```

```python
def compute_array(self, force_recompute=False):
    """
Build the array for deconvolution.

Keyword arguments:
force_recompute -- Recompute even when array_computed is True.

Raises ValueError if self.deconv_coefs is not set.

# First check to see if the array is already computed (self.array_computed),
# if it is and we havent been passed force_recompute=True, skip the rest.
if self.array_computed and not force_recompute:
    return

try:
    temp = deepcopy(self.deconv_array)
except AttributeError as e:
    temp = 0

# self.pixel_size, and self.array_size are guaranteed to be initialized, self.deconv_coefs is
# If self.deconv_coefs is not set, we want to raise an exception
if not self.deconv_coefs:
    raise ValueError('deconv_coefs not set.')

# Create an array of radial distances based on the input FFT frequencies.
self.frequencies = (numpy.fft.fftfreq(self.array_size[0], d=self.pixel_size), # X frequencies
                    numpy.fft.fftfreq(self.array_size[1], d=self.pixel_size)) # Y frequencies

Q = map(lambda XY: numpy.sqrt(XY[0]**2 + XY[1]**2), numpy.meshgrid(self.frequencies[0], self.frequencies[1]))
X,Y = numpy.meshgrid(self.frequencies[0], self.frequencies[1])
Q = numpy.sqrt(X**2 + Y**2)

# Using the array of radial frequencies, create the deconv_array.
self.deconv_array = psf_kernel_func(Q, self.deconv_coefs)

self.array_computed = True

def deconvolve(self, input_data):
    """
    Use the configured deconvolution on the input array, returning the array deconvolved.

    input_data -- The incoming grid we will deconvolve, this should be a 2-axis numpy array.
    """

    # Check to see if we have to create a new deconv array.
    # (Is the array_size the same)
    input_array_size = numpy.shape(input_data)
    self.set_array_size(input_array_size)

    return numpy.abs(numpy.fft.ifft2(numpy.divide(numpy.fft.fft2(input_data), self.deconv_array)))

def read_deconv_coefs_file(self, params_file='deconv_coefs.conf'):
    """
    Reads the deconvolution parameters from a file.
    Configuration file format:
    Lines beginning with '#' are comments and are ignored
    keyword '=' value
    Valid keywords are: deconv_coefs, pixel_size, array_size
    Values may contain any characters, leading whitespace is ignored.
    keyword 'deconv_coefs' expects a whitespace seperated list of paired numbers, to be coerced
to floats.
    these are expected to be of the form 'A0 B0 A1 B1 A2 B2' ...
    keyword 'pixel_size' expects a single float.
    keyword 'array_size' expects a pair of integers whitespace seperated
    """
For simplicity, any line that doesn't begin with an alpha is treated as a comment.

```python
# We will read everything into a dict, then work with the dict to set values
local_dict = dict()
with open(params_file,'r') as input_file:
    for line in input_file:
        if line and not line[0].isalpha(): continue  # Any lines starting with '#' are comments.
        definition = line.partition('=')
        # If there is no '=' to split on, and this isn't a comment line, close the file and error out.
        if definition[1] != '=':
            raise IOError(1, 'Configuration file malformed', params_file)
        local_dict[definition[0]] = definition[2]

# Handle the parameters. The parameters are set to not recompute
# to save time. If the values are changed, the self.array_computed flag will be unset.
if 'pixel_size' in local_dict:
    self.set_pixel_size(local_dict['pixel_size'], recompute=False)
if 'array_size' in local_dict:
    self.set_array_size(local_dict['array_size'], recompute=False)
if 'deconv_coefs' in local_dict:
    deconv_coefs_array = numpy.array(local_dict['deconv_coefs'].split(), dtype=float)
    self.set_deconv_coefs(deconv_coefs_array.reshape(2,-1).tolist(), recompute=False)

# Finally, we will recompute the array if deconv_coefs is set.
if self.deconv_coefs:
    self.compute_array()
```

```python
def main(argv=None):
    display=True
    benchmark=True
    output=False

    f_readfile = None
    f_writefile = None

    if len(sys.argv) < 2:
        sys.stderr.write("Usage: %s input_image [output_image].\n" % sys.argv[0])
        return(1)
    fn_readfile = sys.argv[1]

    # Param 2 is output filename, or we assign it based on input filename.
    if len(sys.argv) == 3:
        fn_writefile = sys.argv[2]
    else:
        fn_readfile_basename = os.path.basename(fn_readfile)
        # Strip off ".gz" from the basename, if it is there.
        if fn_readfile_basename.endswith(".gz",-3):
            fn_readfile_basename = fn_readfile_basename.rpartition(".")[0]

        # Strip off ".3ddose" extention, if there.
        if fn_readfile_basename.endswith(".dose_hist"):
            fn_writefile_basename = fn_readfile_basename.rpartition(".")[0]
        else:
            fn_writefile_basename = fn_readfile_basename

        fn_writefile = os.path.dirname(fn_readfile) + fn_writefile_basename + ".deconv_hist.gz"

    # Try to open the input file for reading and abort if we cannot.
    # Also checks to see if the file looks like a gzip file. If it seems like it is
    # we load it as such, and read the first 5 characters to test this. If it is a
    # false positive, the data was corrupt anyway (first 2 characters should be normal text)
    try:
```
with open(fn_readfile, 'rb') as f_readfile:
    if (f_readfile.read(2) == '\x1f\x8b'):
        f_readfile = gzip.GzipFile(fileobj = f_readfile)
    f_readfile.read(5)
    f_readfile.rewind()
    try:
        (X_Points,Y_Points,Dose_Array,Dose_Error_Array) = cPickle.load(f_readfile)
    except cPickle.UnpicklingError as e:
        sys.stderr.write("Error unpickling data, aborting. (Error: %s)\n" % e.strerror)
        return(1)
    except IOError as e:
        sys.stderr.write("Could not open %s, it may not exist. (Error: %s)\n" % (fn_readfile, e.strerror))
        return(1)
if output:
    # Try to open the output file for reading, if we can, it already exists and we abort.
    try:
        f_writefile = open(fn_writefile, 'r')
    except IOError as e:
        if e.errno == 2: # File not found error, we can continue.
            try:
                f_writefile = gzip.open(fn_writefile, 'wb')
            except IOError as wr_e:
                sys.stderr.write("Could not open %s for writing, do you have permission? (Error: %s)\n" % (fn_writefile, wr_e.strerror))
                return(2)
        else:
            raise
    except:
        f_writefile.close()
        sys.stderr.write("Error: %s already exists, aborting.\n" % fn_writefile)
        return(2)
    try:
        # At this point, we have the writefile open, if anything fails, we should close it and remove it.
        # Create the deconvolution class, start with [A0,B0] = [1,1]
        deconv_coefs = [[1,1]]
        array_size = (X_Points.size, Y_Points.size)
        deconvolver = ExpDeconvolution(in_deconv_coefs=deconv_coefs, in_pixel_size=0.04, in_array_size=array_size)
        output_image = deconvolver.deconvolve(Dose_Array)
    except Exception as e:
        if output:
            f_writefile.close()
            sys.stderr.write("Error (%s), removing \"%s\"\n" % (e, fn_writefile))
            os.remove(fn_writefile)
            raise # Reraise the exception
        if display:
            extents = [min(X_Points), max(X_Points), min(Y_Points), max(Y_Points)]
            Dose_Error_Array[output_image < (0.2 * max(output_image.flatten()))] = 0
            print "Average error value in region > 20% max dose: %f" % numpy.average(Dose_Error_Array)
        plt.figure(1)
        plt.title("Dose Array")
        plt.imshow(Dose_Array, extent=extents, interpolation='nearest')
        plt.colorbar()
This program solves for the optimal parameters for the deconvolution program using an iterative approach and the input files produced from previous simulations.
Quality of the match will be determined by normalizing each pair to the average value of all pixels above a threshold of the maximum pixel value: AVG_THRESHOLD=20% by default. The two arrays will then be subtracted and the average deviation percentage in the thresholded region (in absolute) will be used as the quality term. This allows for a minimization iterative solver to find the optimal value. The quality terms for each input fluence/dose pair will be added in quadrature for the final term. This allows values that are significantly off to have a large effect on the final outcome.

```python
def run_deconv_diff(deconv_func, fluence, dose):
    # Deconvolve and return the average of the square of the difference.
    dose_deconv_norm = deconv_func(dose)
    # Comparison (fluence - dose) / fluence =>
    #   1 - dose / fluence)
    diff_array = 1 - (dose_deconv_norm / fluence)
    # Evaluation
    return numpy.mean(diff_array**2)

def run_deconv_res(deconv_func, fluence, dose):
    # Deconvolve and return the square sum of the residuals.
    dose_deconv_norm = deconv_func(dose)
    # Comparison (fluence - dose)
    residual_array = numpy.abs(fluence - dose_deconv_norm)
    # Evaluation
    return numpy.sum((numpy.abs(residual_array, 2)))

def parallel_deconv_fun(dose):
    global dca
    return numpy.abs(numpy.fft.ifft2(numpy.divide(numpy.fft.fft2(dose), dca)))

def parallel_deconv_worker(deconv_array, inqueue, outqueue, deconv_fn=run_deconv_res):
    # Worker function for deconvolution/comparison.
    global dca
dca = deconv_array
    for fluence, dose in iter(inqueue.get, 'DONE'):
        result = deconv_fn(parallel_deconv_fun, fluence, dose)
        outqueue.put(result)
    inqueue.task_done()
else:
    inqueue.task_done()

class Coef_Optimizer:
    """
    Class for optimizing the deconvolution coefficients.
    """
    Attributes:
    deconv_order -- A parameter defining the order of the devonvolution.
    Currently defaults to 5th order (DECONV_ORDER=5).
    deconv_coefs -- The deconvolution kernel coefficient pairs Ai, Bi as list of 2-tuples: [ [A0, B0], ..., [An, Bn] ]
    fluence_arrays -- A list of the input fluence arrays. Assumed to be matched one-to-one with the dose_arrays.
    dose_arrays -- A list of the input dose arrays. Assumed to be matched one-to-one with the dose_arrays.
    array_size -- a 2-tuple containing the X and Y size in pixels.
    pixel_size -- The physical size of each pixel in the image.

    Functions:
    __init__(self, in_deconv_coefs=None, in_deconv_order=None,
              in_fluence_arrays=None, in_dose_arrays=None):
    set_deconv_coefs(self, in_deconv_coefs=None, in_deconv_order=DECONV_ORDER):
    set_arrays(self, in_fluence_array, in_dose_array):
    append_array_pair(self, in_fluence_array, In_dose_array):
    compute_quality(self): returns quality from deconvolution.
```
optimize_coefs(self, )

```
__name__ = "Coef_Optimizer"
```

DECONV_ORDER = 5

```
""" Default order of the deconvolution function, per Rener, et. al. """
```

deconv_order = None

```
""" A parameter defining the order of the devonvolution. 
Currently defaults to 5th order (DECONV_ORDER=5). """
```

deconv_coefs = []

```
"""A list of 2-tuples used as the coefficients in the psf kernel."""
```

fluence_dose_arrays = []

```
""" A list of the input fluence and dose arrays. 
Takes the form [(Fluence_1, Dose_1), ..., (Fluence_n, Dose_n)]""
```

PIXEL_SIZE = 0.04

```
""" The size of each pixel in cm ""
```

array_size = None

```
""" The size of the arrays being passed in. 
All arrays must be of the same size. ""
```

deconvolver = None

```
""" This will hold the deconvolver to be used later. ""
```

workers = []

```
""" A set of workers for the multiproc. optimizer ""
```

def __init__(self, in_deconv_coefs=None, in_fluence=None, in_dose=None,
in_pixel_size=PIXEL_SIZE, in_deconv_order=DECONV_ORDER):

```
Initialize the class, setting the parameters if passed.
```

Keyword arguments:
in_deconv_coefs -- The deconvolution kernel coefficient pairs Ai, Bi as 
list of 2-tuples: [ [A0, B0], ..., [An, Bn] ]
in_fluence_arrays -- A list of the input fluence arrays. Assumed to be 
matched one-to-one with the dose_arrays.
in_dose_arrays -- A list of the input dose arrays. Assumed to be matched 
one-to-one with the fluence_arrays.
in_deconv_order -- The order of the devonvolution array to set up.
```

# We should be set to setup the coefficients. If in_deconv_coefs is
# None, set_deconv_coefs will create a default.
self.set_deconv_coefs(in_deconv_coefs, in_deconv_order)

# Setup the arrays, any error handling is in the array setup code.
self.set_arrays(in_fluence, in_dose)

# Set the pixel size.
sself.set_pixel_size(in_pixel_size)

def set_deconv_coefs(self, in_deconv_coefs=None, in_deconv_order=DECONV_ORDER):

```
Set the coefficients of the deconvolution kernel.
```

If in_deconv_coefs is set, it will set in_deconv_order and do a deepcopy.
If the deconv_coefs are not set, we will create a default set of order 
in_deconv_order or DECONV_ORDER.

Keyword arguments:
in_deconv_coefs -- See __init__, optional.
in_deconv_order -- See __init__, optional.
if self.deconv_order:
    self.deconv_order = in_deconv_order

if self.deconv_coefs:
    if self.deconv_coefs != in_deconv_coefs:
        recompute = True  
        # Set the order to the length of the input array.
        self.deconv_order = len(in_deconv_coefs)
    else:
        self.deconv_coefs = [
            [float(i)/self.deconv_order, float(i+1)/self.deconv_order]  
            for i in range(self.deconv_order)]
        recompute = True

if self.deconvolver and recompute:
    # Deconvolver has been set up, reconfigure it after setting
    # the local variable value. We assume if it has been setup
    # we are alright to recompute it.
    self.deconvolver.set_deconv_coefs(self.deconv_coefs)

def set_pixel_size(self, in_pixel_size):
    # Set the physical size of each pixel.
    # If the deconvolution object exists, we resend the pixel size.
    # If not, we just set self.pixel_size and it will be taken care of later.
    # set the pixel size
    self.pixel_size = in_pixel_size
    if self.deconvolver:  
        # Deconvolver has been set up, reconfigure it after setting
        # the local variable value. We assume if it has been setup
        # we are alright to recompute it.
        self.deconvolver.set_pixel_size(self.pixel_size)

def set_arrays(self, in_fluence_array, in_dose_array):
    # Set the array pairs up.
    # if in_fluence_arrays and in_dose_arrays do not have the same number of
    # members, we will raise an error and exit. This includes if one is
    # passed and the other is not.
    if in_fluence_array and in_dose_array:
        # Check if we have a list or a single item.
        # If we have a list, numpy.shape of in[0] will have len()==2
        # If we have a single item, numpy.shape of in[0] will have len()==1
        # If we have anything else, we have the wrong parameters
        in_fluence_shape = numpy.shape(in_fluence_array[0])
        in_dose_shape = numpy.shape(in_dose_array[0])
        if len(in_fluence_shape) == len(in_dose_shape) == 2:
            # We have a list, check if they are the same size.
            if len(in_fluence_array) == len(in_dose_array) != 0:
                # They are the same size, deepcopy them in so external changes
                # won't affect us here.
                map(self.append_array_pair, in_fluence_array, in_dose_array)
elif len(influence_array) != len(dose_array):
    # Arrays not the same size, raise ValueError
    raise ValueError('%s: influence_array and dose_array not the same size.
(%i, %i resp.)' % (str(self.__name__), len(influence_array), len(dose_array)))
else:
    # Both arrays exist but are empty, raise ValueError.
    raise ValueError('%s: influence_array and dose_array are both empty. (If
not initializing with values, do not pass arrays)' %
        str(self.__name__))

elif len(influence_shape) == len(dose_shape) == 1:
    # Not a list, add them to the array.
    self.append_array_pair(influence_array, dose_array)
else:
    # Not the same, or not the right input shape. Raise ValueError.
    raise ValueError('%s: influence_array and dose_array not right shape. (%s, %s
resp.)' %
        (str(self.__name__), str(numpy.shape(influence_array)),
        str(numpy.shape(dose_array))))

def append_array_pair(self, influence_array, dose_array):
    ""
    Append an array pair to the fluence_dose_arrays list.
    If influence_arrays and dose_arrays do not have the same number of members, we will raise
an error and exit.
    This includes if one is passed and the other is not.
    Keyword arguments:
    influence_arrays -- An input influence array. Assumed to be matched one-to-one with the
dose_array.
    dose_arrays -- A list of the input dose arrays. Assumed to be matched one-to-one with
the fluence_array.
    ""
    if not self.array_size: # This is the first array, we will set up array size.
        self.array_size = numpy.shape(influence_array)
    # Check for correct shape.
    if numpy.shape(influence_array) != numpy.shape(dose_array) != self.array_size:
        raise ValueError('%s: Shape of influence_array and dose_array is not
consistent.\n%s\n%s\n%s' %
            (str(self.__name__), str(numpy.shape(influence_array)),
            str(numpy.shape(dose_array)), str(self.array_size)))
        self.fluence_dose_arrays.append(deepcopy((influence_array, dose_array))))

def compute_quality(self, deconv_coefs=None, run_deconv=run_deconv_res):
    ""
    Computes the "quality" of the match for the current parameters.
    Quality of the match will be determined by normalizing each pair to the average value of
all
    pixels above a threshold of the maximum pixel value: AVG_THRESHOLD=20% by default.
    The two arrays will then be subtracted and the average deviation percentage in the
thresholded
    region (in absolute) will be used as the quality term. This allows for a minimization
iterative
    solver to find the optimal value.
The quality terms for each input fluence/dose pair will be added in quadrature for the final term. This allows values that are significantly off to have a large effect on the final outcome.

```python
if deconv_coefs is not None:
    arranged_coefs = numpy.array(deconv_coefs).reshape((-1,2)).tolist()
    self.set_deconv_coefs(arranged_coefs)

if len(self.fluence_dose_arrays) == 0:
    raise ValueError("%s: In compute_quality- fluence_dose_arrays empty!") %
    str(self.__name__)

if not self.deconvolver: # We haven't set up the deconvolver function yet, do so now.
    self.deconvolver = Deconv(self.deconv_coefs, self.pixel_size, self.array_size)

quality_sq = 0.0
for fluence, dose in self.fluence_dose_arrays:
    quality_sq += run_deconv(self.deconvolver.deconvolve, fluence, dose)

print numpy.sqrt(quality_sq)
return numpy.sqrt(quality_sq)
def optimize_quality(self):
    """ Adjusts the current values of self.deconv_coefs to optimize the quality function. """
    coefs_flat = numpy.array(self.deconv_coefs).reshape(-1).tolist()

    # set the bounds for each item
    bounds = [ (1e-12, None) for x in coefs_flat ]

    # We will use scipy.optimize.fmin_l_bfgs_b with func=self.compute_quality
    return optimize.fmin_l_bfgs_b(self.compute_quality, coefs_flat,
                                 approx_grad=True, bounds=bounds, iprint=1)

def optimize_quality_nlp(self, solver='ralg'):
    """ Adjusts the current values of self.deconv_coefs to optimize the quality function. """
    coefs_flat = numpy.array(self.deconv_coefs).reshape(-1).tolist()

    # set the bounds for each item
    lower_bound = [ 1e-12 for x in coefs_flat ]

    # We will use openopt.NLP with func=self.compute_quality
    p = NLP(self.compute_quality, coefs_flat, lb=lower_bound, iprint=10,
            plot=1, show=True, scale=1e-3)

    r = p.solve(solver)
    return r

def optimize_quality_nlp_ll(self, shared_array, solver='ralg'):
    from openopt import NLP
    """ Adjusts the current values of self.deconv_coefs to optimize the quality function. """
    coefs_flat = numpy.array(self.deconv_coefs).reshape(-1).tolist()

    # set the bounds for each item
    lower_bound = [ 1e-12 for x in coefs_flat ]

    # We will use openopt.NLP with func=self.compute_quality
    p = NLP(self.compute_quality, coefs_flat, lb=lower_bound, iprint=10,
            plot=1, show=True, scale=1e-3)

    try:
        self.deconv_array = numpy.frombuffer(shared_array.get_obj())
    except AttributeError: # otherwise, we have the object.
        self.deconv_array = numpy.frombuffer(shared_array)
```
self.deconv_array.shape = self.array_size

# We will use openopt.NLP with func=self.compute_quality
p = NLP(self.compute_quality_ll, coefs_flat, lb=lower_bound, iprint=10, plot=1, show=True)

r = p.solve(solver)
if self.workers:
    for worker in self.workers:
        self.task_queue.put('DONE')
    self.task_queue.join()
    self.task_queue.close()
    self.results_queue.close()
    self.workers = None

return r

def compute_quality_ll(self, deconv_coefs=None, run_deconv=run_deconv_res):
    ""
    Computes the "quality" of the match for the current parameters.
    Quality of the match will be determined by normalizing each pair to the average value of all
    pixels above a threshold of the maximum pixel value: AVG_THRESHOLD=20% by default.
    The two arrays will then be subtracted and the average deviation percentage in the thresholded
    region (in absolute) will be used as the quality term. This allows for a minimization iterative
    solver to find the optimal value.
    The quality terms for each input fluence/dose pair will be added in quadrature for the final
term. This allows values that are significantly off to have a large effect on the final outcome.
    ""
    if deconv_coefs is not None:
        arranged_coefs = numpy.array(deconv_coefs).reshape((-1,2)).tolist()
        self.set_deconv_coefs(arranged_coefs)
        self.deconv_array[:] = self.deconvolver.deconv_array

    if len(self.fluence_dose_arrays) == 0:
        raise ValueError('"%s: In compute_quality- fluence_dose_arrays empty!" %
            str(self.__name__) )

    if not self.deconvolver: # We haven't set up the deconvolver function yet, do so now.
        self.deconvolver = Deconv(self.deconv_coefs, self.pixel_size, self.array_size)

    if not self.workers: # We don't have a pool set up yet.
        self.workers = []
        NCPU = min(cpu_count(), len(self.fluence_dose_arrays))
        self.task_queue = JoinableQueue()
        self.results_queue = Queue()
        for i in range(NCPU):
            print("Starting Worker %i" % i)
            self.workers.append(Process(target=parallel_deconv_worker, 
                args=(self.deconv_array, self.task_queue, self.results_queue)))
            self.workers[i].daemon = True
            self.workers[i].start()

    quality_sq = 0.0

    for data in self.fluence_dose_arrays:
        self.task_queue.put(data)

    self.task_queue.join()

    for i in range(len(self.fluence_dose_arrays)):
```python
quality_sq += self.results_queue.get()

print "Q ll: ", numpy.sqrt(quality_sq)
return numpy.sqrt(quality_sq)

def main(argv=None):
    display = True
    benchmark = True
    output = False
    AVG_THRESHOLD = 0.2
    f_readfile = None
    f_writefile = None
    if len(sys.argv) < 2:
        sys.stderr.write("Usage: %s input_dataset.\n" % sys.argv[0])
        return(
    fn_readfile = sys.argv[1]
    # Try to open the input file for reading and abort if we cannot.
    # Also checks to see if the file looks like a gzip file.  If it seems like it is
    # we load it as such, and read the first 5 characters to test this.  If it is a
    # false positive, the data was corrupt anyway (first 2 characters should be normal text)
    try:
        with open(fn_readfile, 'rb') as f_readfile:
            if (f_readfile.read(2) == '\x1f\x8b'):
                f_readfile = gzip.GzipFile(fileobj=f_readfile)
                f_readfile.rewind()
                f_readfile.read(5)
                f_readfile.rewind()
            else:
                f_readfile.seek(0)
                # We have the file open and an interface ready to read from it.
                # Now lets loop through the input datasets.
                try:
                    # Set up the dictionary to hold our set.
                    data_sets = dict()
                    # Loop
                    while 1:
                        try:
                            # We want to catch only EOFError
                            # (means we have finished reading from the file)
                            (field_size, fluence, dose) = cPickle.load(f_readfile)
                            data_sets[field_size] = (fluence, dose)
                            print('Added %s' % field_size)
                        except EOFError:
                            # EOFError should mean we are at the end of the file.
                            break
                        except cPickle.UnpicklingError as e:
                            sys.stderr.write("Error unpickling data, aborting. (Error: %s)\n" % e.strerror)
                            return(1)
                        except IOError as e:
                            sys.stderr.write("Could not open %s, it may not exist. (Error: %s)\n" % (fn_readfile, e.strerror))
                            return(1)
                if '10x10' in data_sets:
                    fluence, dose = data_sets['10x10']
                    # Normalization
                    cutoff = fluence >= numpy.max(fluence) * AVG_THRESHOLD
                    fluence_avg = numpy.mean(fluence[cutoff])
                    dose_avg = numpy.mean(dose[cutoff])
                    dose_norm = fluence_avg / dose_avg
        return(1)
```
Normalizations = {'2x2': 0.761, '5x5': 0.761, '10x10': 1, '20x20': 2.78, '25x25': 6.92}

for size in Normalizations:
    data_sets[size] = tuple(Normalizations[size] * array for array in data_sets[size])

fluence_arrays, dose_arrays = zip(*data_sets.values())

print(fluence_arrays, dose_arrays)

for dose in dose_arrays:
    dose *= dose_norm

deconv_coefs = [[9.9707, 22.8806],
                 [0.852811, 2.27858],
                 [0.0367546, 0.614349],
                 [0.10102e-5, 0.0638869],
                 [0.56198e-8, 0.00574182]]
#deconv_coefs = [[ 2.86773276e+07,   8.38625193e+03],
#                 [ 4.16705648e+07,   5.1669550e+03],
#                 [ 3.39912833e+13,   2.55941836e+13],
#                 [ 4.47829145e+14,   8.56560594e+15],
#                 [ 5.20701791e+07,   1.27395524e+09]]
optimizer = Coef_Optimizer(in_fluence=fluence_arrays, in_dose=dose_arrays, in_pixel_size=0.04, in_deconv_coefs=deconv_coefs)

quality = optimizer.compute_quality(run_deconv=run_deconv_res)

if display:
    print 'Initial Quality is: %s' % str(quality)

shared_array = Array('d', numpy.prod(optimizer.array_size), lock=False)
#opt_res = optimizer.optimize_quality_nlp_l1(shared_array)
opt_res = optimizer.optimize_quality_nlp()

if display:
    print('Array is: ', numpy.array(opt_res.xf).reshape(-1,2))
    print('Final quality is: %s' % str(optimizer.compute_quality(run_deconv=run_deconv_res)))

return 0

def main():
    return 0

if __name__ == '__main__':
    import cPickle
    import sys
    import os
    import gzip
    import matplotlib.pyplot as plt

    status = main()
    sys.exit(status)
Appendix F

Fluence Calculation Tools

Assembled in this appendix are the various tools used to create the parameter space, calculate the fluence, and display the results.

F.1 create_deconv_parameter_space.py

```python
#!/usr/bin/python
import os
import sys
import numpy
import matplotlib.pyplot as plt
import cPickle
import gzip
import struct
from signal import signal, SIGINT, SIG_IGN
from time import time
from scipy import sparse
import subprocess
from utils import openfile, si, buf_data_from_file
from utils import MODE0_dt, MODE2_dt, sp_coo_append

if __name__ == '__main__':

    benchmark = True
    output = True
    TESTING = True

    out_fluence_bins = 1024
    if TESTING == True:
        # A quick test to make this equivalent to the normal histogram binning
        # program. Otherwise it does as it should.
        shift_num = 5
        phsp_bin_ct = 2**shift_num
    else:
        phsp_bin_ct = 1024
    def_min = -20.48
    def_max = 20.48
```
def_range = (((def_min, def_max), (def_min, def_max)))
def_points = numpy.linspace(def_min-def_min/(out_fluence_bins), def_max-def_max/(out_fluence_bins), out_fluence_bins)
def_readblock = 1000000

# PHSP header format: char MODE_RW[5]; int NPPHSP, NPOTPHSP; float EMAXPHSP, ERMINPHSP, NINCPHSP; pad[3]
fmt_PHSPhead = '-5s 2I 3f'
f_readfile = None
f_writefile = None

if len(sys.argv) < 2:
sys.stderr.write("Usage: %s input_points [output_hist].\n" % sys.argv[0])
sys.exit(1)

fn_readfile = sys.argv[1]

# Param 2 is output filename, or we assign it based on input filename.
if len(sys.argv) == 3:
    fn_writefile = sys.argv[2]
else:
    fn_readfile_basename = os.path.basename(fn_readfile)
    # Strip off ".gz" from the basename, if it is there.
    if fn_readfile_basename.endswith(('.gz'), -3):
        fn_readfile_basename = fn_readfile_basename.rpartition('.')[0]
    # Strip off ".egsphsp?" extention, if there.
    if fn_readfile_basename.endswith(('.egsphsp'), 0, -1):
        fn_writefile_basename = fn_readfile_basename.rpartition('.')[0]
    else:
        fn_writefile_basename = fn_readfile_basename
    fn_writefile = os.path.dirname(fn_readfile) + fn_writefile_basename + ".spparam.gz"

# Try to open the input file for reading and abort if we cannot.
## Also checks to see if the file looks like a gzip file. If it seems like it is
## we load it as such, and read the first 5 characters to test this. If it is a
## false positive, the data was corrupt anyway (first 2 characters should be ASCII 'MO')
mode_string = ""
try:
    f_readfile = open(fn_readfile, 'rb')
    if (f_readfile.read(2) == '\\xf1\\x8b'):
        f_readfile = gzip.GzipFile(fileobj = f_readfile)
        f_readfile.rewind()
        mode_string = f_readfile.read(5)
        f_readfile.rewind()
    else:
        f_readfile.seek(0)
        mode_string = f_readfile.read(5)
        f_readfile.seek(0)
except IOError as e:
sys.stderr.write("Could not open %s, it may not exist. (Error: %s)\n" % (fn_readfile, e.strerror))
sys.exit(1)

if output:
    # Try to open the output file for reading, if we can, it already exists and we abort.
    # Otherwise, so we should be safe to open for writing.
    try:
        open(fn_writefile, 'r')
    except IOError as e:
        if e.errno == 2:  # File not found error, we can continue.
            #try:
            #    f_writefile = gzip.open(fn_writefile, 'wb')
            #except IOError as wr_e:
            #    sys.stderr.write("Could not open %s for writing, do you have permission?\n" % (fn_writefile, wr_e.strerror))
            sys.exit(2)
pass
else:
    raise
else:
    sys.stderr.write("Warning: %s already exists.\n" % fn_writefile)
    #sys.exit(2)
f_writefile = open(fn_writefile,'wb')

mode_size = None
if mode_string[4] == '0':  # MODE0 file, use MODE0_dt (UINT LATCH; float ESHORT,X,Y,U,V,WT)
    mode_dtype = MODE0_dt
    mode_size = 28
elif mode_string[4] == '2':  # MODE2 file, use MIDE2_dt (UINT LATCH; float ESHORT,X,Y,U,V,WT,WT,ZLAST)
    mode_dtype = MIDE2_dt
    mode_size = 32
else:
    sys.stderr.write('Header format corrupted, or unusable mode (%s). Aborting.\n', mode_string)
sys.exit(1)

header = f_readfile.read(mode_size)
if len(header) <> mode_size:
    sys.stderr.write("Error: input file %s does not have a complete header.\n" % fn_readfile)
sys.exit(1)
else:
    # Add the appropriate padding to get up to mode_size length
    fmt_PHSPhead = fmt_PHSPhead + str(mode_size - struct.calcsize(fmt_PHSPhead)) + 'x'
    header_unpacked = struct.unpack(fmt_PHSPhead, header)
    # cleaner way of doing above, all in a single list
    header_unpacked = struct.unpack(fmt_PHSPhead, header)
    print('MODE: %s, np: %i, np_phot: %i, ekmax: %f, ekminE: %f, ninc: %.0f' % header_unpacked)
readblock_size = def_readblock * mode_size

Use numpy.histogram to bin particles.
for each particles in the output:
    get appropriate bin for self
    set in appropriate input bin condensed along X
    set in appropriate input bin condensed along Y

# Construct the arrays as sparse matrices. Should be less memory costly.
params_arrays = [(sparse.coo_matrix((out_fluence_bins, out_fluence_bins), dtype=numpy.int16)
        for x in range(phsp_bin_ct+1)]
        for y in range(phsp_bin_ct+1)]
data_bytearray = bytarray(readblock_size)
points = float(0)
start = time()
rate = []
try:
    #for data in buf_data_from_file(f_readfile, readblock_size):
        for bytes_read in buf_data_from_file(f_readfile, data_bytearray):
            # We either haven't set up particles, or we are at the tail end of the data
            #+ particles should be a list of floats, each 4 bytes long, so len(data)/4
            #+ floats.
            particles = numpy.frombuffer(data_bytearray, dtype = mode_dtype)
            if bytes_read != particles nbytes:
                # We have reached the end of the data, will set to break
                # after this round of the loop, and create a new array with just
                # this data. The slice operation on the data_bytearray will
                # create a copy, but it should be alright for the last set.
                particles = numpy.frombuffer(data_bytearray[bytes_read:len(data)], dtype = mode_dtype)
        else:
            # The particles array is setup, and we have data in it. Because
we used a buffer, the data has already been updated and we can
work right on it.

pass

Breakdown: The data has been loaded into an array with the
appropriate dtype which has set it up to use X and Y and LATCH.
We will take the LATCH and recover the origin location, binning it
into X and Y coordinates for each particles (inX, inY arrays). We
will then iterate over all of the latch bits and histogram each
partition that falls into that latch bit for X and then for Y. We
add these to the histogram arrays we already set up (1024 arrays
of histogram size, plus an extra array for those particles which
came from outside the original pixel binning.)

The latch bits were set as follows:
Latch bits are zero indexed here, i.e. bits 0-31
We want to attach a latch for X on bits 2-11 (giving 1024 values)
of the latch, and Y on bits 12-21. If we are
outside of the range (-512 - +512 pixels), we will only set bit 1.
We Will also want to remove any existing bits, this means using the
magic number 0xFF000001 to set all bits we are using to zero.

If bit 1 is set, other bits will be zero, so we can just subtract it.
This will make it have the "MAGIC" value of 1024 which should
never exist in the other bits.

inX = (particles['LATCH'] & 4092) >> 2 | (particles['LATCH'] & 2) << 9 # bits 2-11

store X

inY = (particles['LATCH'] & 4190208) >> 12 | (particles['LATCH'] & 2) << 9 # bits 12-21

store Y

if TESTING == True:
inX = inX>>(10-shift_num)
inY = inY>>(10-shift_num)

# Now that we are done, flush down to a csr for storage.
for x in range(len(paramsp_arrays)):
    for y in range(len(paramsp_arrays[x])):
        paramsp_arrays[x][y] = paramsp_arrays[x][y].tocsr()
if benchmark: end = time()
f_readfile.close()  # Close file

if benchmark: print 'total time: %f s, average speed: %spts/s' % ((end-start), si(points / (end - start)))

if output:
    #writer = subprocess.Popen(["pigz", ",-f"], stdout=f_writefile, stdin=subprocess.PIPE)
    #writable = writer.stdin
    writable = f_writefile
    cPickle.dump((def_points, def_points, params_arrays), writable, cPickle.HIGHEST_PROTOCOL)
f_writefile.close()

F.2 ll_create_deconv_param_space.py

#!/usr/bin/python

import os
import sys
import numpy
import cPickle
import gzip
import struct
from signal import signal, SIGINT, SIG_IGN
from time import time
from scipy import sparse
from subprocess import Popen, PIPE
from cPickle import dump
from zipfile import ZipFile
from multiprocessing import Pool
from multiprocessing.shared_memory import SharedMemory
from functools import partial
from time import time
from functools import partial
from signal import signal, SIGINT
from multiprocessing import Pool
from signal import signal, SIGINT
from functools import partial
from time import time
from multiprocessing import Pool
from signal import signal, SIGINT
from functools import partial
from time import time
from multiprocessing import Pool
from signal import signal, SIGINT
from functools import partial
from time import time

#def DEBUG(*args):
#    if len(args) == 1:
#        sys.stdout.write(args[0])
#    else:
#        print(args)
#
@ReturnOnKeyboardInterrupt
def parallel_worker(shared_buffers, in_consts):
    master_queue, task_queue, results_queue = queues
    mode_dtype, out_fluence_bins, def_range = in_consts
    particles = numpy.frombuffer(data_bytearray, dtype = mode_dtype)
inX = numpy.frombuffer(inX_buf, dtype = numpy.uint32)
inY = numpy.frombuffer(inY_buf, dtype = numpy.uint32)
    particles = None
    inX = None
    inY = None
    n_particles = 0
    for master_cmd in iter(master_queue.get, 'DIE'):
        if master_cmd != n_particles:
            n_particles = master_cmd
            particles = numpy.frombuffer(data_bytearray, dtype = mode_dtype, count=n_particles)
            inX = numpy.frombuffer(inX_buf, dtype=numpy.uint32, count=n_particles)
            inY = numpy.frombuffer(inY_buf, dtype=numpy.uint32, count=n_particles)
for x_pos, y_pos in iter(task_queue.get, 'DONE'):
    task_queue.task_done()
    latch_view = particles[(inX == x_pos) & (inY == y_pos)]
    hist = sparse.coo_matrix(numpy.histogram2d(latch_view['X'], latch_view['Y'],
        out_fluence_bins, def_range)[0], dtype=numpy.int16)
    # out_fluence_bins, def_range[0])
    results_queue.put((x_pos, y_pos, hist))
    del hist, latch_view
    task_queue.task_done()
#prev_part = particles.copy()
#prev_inX = inX.copy()
#prev_inY = inY.copy()
master_queue.task_done()
master_queue.task_done()

def launch_workers(shared_buffers, queues, in_consts, workers=None):
    """ Sets up and launches all workers, including creating shared memory
    segments for each worker. """
    if workers is None:
        workers = []
    for i in range(cpu_count()):
        # for i in range(1):
        new_worker = Process(target=parallel_worker,
            args=(shared_buffers, queues, in_consts))
            #shared_buffers, queues, in_consts)
        new_worker.daemon = True
        new_worker.start()
        workers.append(new_worker)
    return workers

if __name__ == '__main__':
    benchmark = True
    output = True
    TESTING = True

    out_fluence_bins = 1024
    if TESTING == True:
        # A quick test to make this equivalent to the normal histogram binning
        # program. Otherwise it does as it should.
        shift_num = 7
        phsp_bin_ct = 7**shift_num
    else:
        phsp_bin_ct = 1024
    def_min = -20.48
    def_max = 20.48
    def_range = ((def_min, def_max), (def_min, def_max))
    def_points = numpy.linspace(def_min, def_max, out_fluence_bins)
    def_max/(out_fluence_bins) def_max/
    def_readblock = 1000000
    # PHSP header format: char MODE_RW[5]; int NPPHSP, NPHOTPHSP; float EMAXPHSP, EKMINPHSPE,
    NINCPHSP; pad[3]
    fmt_PHSPhead = '-5s 2I 3f '
    f_readfile = None
    f_writefile = None
    if len(sys.argv) < 2:
        sys.stderr.write( "Usage: %s input_points [output_hist].\n" % sys.argv[0])
        sys.exit(1)
    fn_readfile = sys.argv[1]
    # Param 2 is output filename, or we assign it based on input filename.
    if len(sys.argv) == 3:
        fn_writefile = sys.argv[2]
    else:
fn_readfile_basename = os.path.basename(fn_readfile)
# Strip off ".gz" from the basename, if it is there.
if fn_readfile_basename.endswith(('.gz',),-3):
    fn_readfile_basename = fn_readfile_basename.rpartition(".")[0]

# Strip off ".egspshp?" extension, if there.
if fn_readfile_basename.endswith(('.egspshp'),0,-1):
    fn_writefile_basename = fn_readfile_basename.rpartition(".")[0]
else:
    fn_writefile_basename = fn_readfile_basename

fn_writefile = os.path.dirname(fn_readfile) + fn_writefile_basename + ".spparam.gz"

# Try to open the input file for reading and abort if we cannot.
# Also checks to see if the file looks like a gzip file. If it seems like it is
# we load it as such, and read the first 5 characters to test this. If it is a
# false positive, the data was corrupt anyway (first 2 characters should be ASCII 'MO')
mode_string = ""
try:
    f_readfile = open(fn_readfile,'rb')
    if (f_readfile.read(2) == '\x1f\x8b'):
        f_readfile = gzip.GzipFile(fileobj = f_readfile)
        f_readfile.rewind()
        mode_string = f_readfile.read(5)
        f_readfile.rewind()
    else:
        f_readfile.seek(0)
        mode_string = f_readfile.read(5)
        f_readfile.seek(0)
except IOError as e:
    sys.stderr.write("Could not open %s, it may not exist. (Error: %s)\n" % ( fn_readfile, e.strerror ))
    sys.exit(1)

if output:
    # Try to open the output file for reading, if we can, it already exists and we abort.
    # Otherwise, so we should be safe to open for writing.
    try:
        open(fn_writefile,'r')
    except IOError as e:
        if e.errno == 2: # File not found error, we can continue.
            #try:
            #    f_writefile = gzip.open(fn_writefile,'wb')
            #except IOError as wr_e:
            #    sys.stderr.write("Could not open %s for writing, do you have permission? (Error: %s)\n" % ( fn_writefile, wr_e.strerror ))
            #sys.exit(2)
        else:
            raise
    else:
        sys.stderr.write("Warning: %s already exists.\n" % fn_writefile)
        sys.exit(2)
    f_writefile = open(fn_writefile,'wb')

mode_size = None
if mode_string[4] == '0': # MODE0 file, use MODE0_dt (UINT LATCH; float ESHORT,X,Y,U,V,WT)
    7 * 4 Bytes
    mode_dtype = MODE0_dt
    mode_size = 28
elif mode_string[4] == '2': # MODE2 file, use MIDE2_dt (UINT LATCH; float ESHORT,X,Y,U,V,WT,ZLAST) = 8 * 4 Bytes
    mode_dtype = MIDE2_dt
    mode_size = 32
else:
    sys.stderr.write('Header format corrupted, or unusable mode (%s). Aborting.\n',
    mode_string)
    sys.exit(1)
# into X and Y coordinates for each particles (inX, inY arrays). We will take the LATCH and recover the origin location, binning it appropriate dtype which has set it up to use X and Y and LATCH.

Breakdown: The data has been loaded into an array with the appropriate dtype which has set it up to use X and Y coordinates for each particles (inX, inY arrays). We
# will then iterate over all of the latch bits and histogram each 
# particles that falls into that latch bit for X and then for Y. We 
# add these to the histogram arrays we already set up (1024 arrays 
# histogram size, plus an extra array for those particles which 
# came from outsize the original pixel binning.)

# The latch bits were set as follows: 
# (Latch bits are zero indexed here, i.e. bits 0-31) 
# We want to attach a latch for X on bits 2-11 (giving 1024 values) 
# of the latch, and Y on bits 12-21. If we are 
# outside of the range (-512 - +512 pixels), we will only set bit 1. 
# We Will also want to remove any existing bits, this means using the 
# magic number \( \text{xFF000001} \) to set all bits we are using to zero.

# If bit 1 is set, other bits will be zero, so we can just subtract it. 
# This will make it have the "MAGIC" value of 1024 which should 
# never exist in the other bits.

# 1023<< 2 = 4092    = X bit mask
# 1023<<12 = 4190208 = Y bit mask

# We will also want to remove any existing bits, this means using the 
# magic number \( \text{xFF000001} \) to set all bits we are using to zero.

# If bit 1 is set, other bits will be zero, so we can just subtract it.

inX.data[:] = (particles['LATCH'] & 4092) >> 2 | (particles['LATCH'] & 2) << 9 # bits

2-11 store X

inY.data[:] = (particles['LATCH'] & 4190208) >> 12 | (particles['LATCH'] & 2) << 9 # bits

12-21 store Y

if TESTING == True:
    inX.data[:] = inX.data >> (10-shift_num)
    inY.data[:] = inY.data >> (10-shift_num)

# from IPython.Shell import IPShellEmbed
# IPShellEmbed()()

n_sent = 0

for x_pos, y_pos in set((tuple(xy) for xy in numpy.column_stack((inX, inY))):
    # Make a new view on the array. Views are cheap and do not copy 
    # the data, so this is still a fast operation.
    # latch_view = particles[(inX == x_pos) & (inY == y_pos)]
    # if len(latch_view) != 0:
    #    sp_coo_append(paramsp_arrays[x_pos][y_pos],
    #        numpy.histogram2d(latch_view['X'],
    #        latch_view['Y'], out_fluence_bins, def_range)[0])
    #DEBUG("master: putting job: x: %s, y: %s" % (x_pos, y_pos))
    task_queue.put((x_pos, y_pos))
    n_sent += 1

for i in range(n_sent):
    #DEBUG("\rmaster: getting results: %s" % i)
    x_pos, y_pos, hist_coo = results_queue.get()
    sp_coo_append(paramsp_arrays[x_pos][y_pos], hist_coo)

for worker in workers:
    task_queue.put('DONE')

master_queue.join()

points += len(particles)

sys.stdout.write( '%i / %i (%.1f%) (%spts/s)' %
    (points, header_unpacked[1], points*100.0)/header_unpacked[1],
    si(points/(time() - start) )))
sys.stdout.flush()

else: # EOF reached!
    sys.stdout.write( '%i / %i (%.1f%) (%spts/s) Done!\n' %
    (points, header_unpacked[1], points*100.0)/header_unpacked[1],
    si(points/(time() - start) )))
    sys.stdout.flush()

# Now that we are done, flush down to a csr for storage.

for x in range(len(paramsp_arrays)):
    for y in range(len(paramsp_arrays[x])):
        paramsp_arrays[x][y] = paramsp_arrays[x][y].tocsr()
sys.stderr.write('Aborting!
')
sys.stderr.flush()  
sys.exit(3)

#except:
#    from IPython.Shell import IPShellEmbed
#    IPShellEmbed()

if benchmark: end = time()
f_readfile.close()  # Close file

if benchmark: print 'total time: %f s, average speed: %spts/s' % ((end-start), si(points / (end - start)))

if output:
    #writer = subprocess.Popen(['pigz', '-f'], stdout=f_writefile, stdin=subprocess.PIPE)
    #writable = writer.stdin
    writable=f_writefile
    cPickle.dump((def_points,def_points,paramsp_arrays),writable,cPickle.HIGHEST_PROTOCOL)
    f_writefile.close()

F.3 fluence_convolution.py

#!/usr/bin/python

""
Compute the fluence at exit using the weights of the fluence at entrance.
""

import numpy  
from scipy import sparse
import cPickle
import sys
import os
import gzip
from utils import si, openfile, sp_coo_append

class FluenceConvolution:
    ""
    Class for convolution of fluence from weights at entrance.
    ""
    Attributes:
        array_shape - Size of the output array in (pixels, pixels) format.
        num_arrs - Number of arrays and size of side of coefs input.
        X_array - Array of length num_arrs of array_shape grids.
        Y_array - Array of length num_arrs of array_shape grids.
    Functions:
        __init__ (param_arrays=None, params_filename=None)
        set_arrays(X_array, Y_array)
        read_parameter_space_file(params_filename='params.spparam.gz')
        compute_array(coefficients)
    ""
    X_array = None
    Y_array = None
    num_arrs = None
    array_shape = None
    points = None
    result = None
    prev_coefs = None
    coefs_shape = None
    paramsp_shape = None
    """ The coordinates of the center of each voxel. """
    cf_to_sp = None
cf_to_fl = None

def __init__(self, param_arrays=None, params_filename=None):
    """Initialize the class, setting the parameters if passed.

    Keyword arguments:
    param_arrays - Set of two arrays to be used for X_array, Y_array.
    params_filename  - File to read parameters from using read_parameter_arrays_file
    """
    if param_arrays is not None:
        self.set_arrays(param_arrays)
    elif params_filename is not None:
        self.read_parameter_space_file(params_filename)

def compute_array(self, coefficients, restart=False):
    """Compute an output array based on the input coefficients.

    Keyword arguments:
    coefficients - a 2d array of [num_arrs, num_arrs] consisting of the weights assigned
                  to each section of the fluence.
    restart - Force full recomputation, even if we have already run a calc
              already. (Useful if the coefs have changed drastically).

    Returns:
    An array of array_shape containing the computed output.

    Raises:
    ValueError if coefficients.shape is not the same as [num_arrs, num_arrs].
    """
    if self.prev_coefs == None or restart == True:
        self.prev_coefs = coefficients.copy()
        self.coefs_shape = self.prev_coefs.shape
        inarr = coefficients
        inarr = sparse.coo_matrix(coefficients, dtype=float)
        self.calc_fluence.col.resize(0)
        self.calc_fluence.row.resize(0)
        self.calc_fluence.data.resize(0)
    else:
        inarr = coefficients - self.prev_coefs
        inarr = sparse.coo_matrix((coefficients - self.prev_coefs),
                                   dtype=float)
        if inarr.shape[0] != self.num_arrs or inarr.shape[1] != self.num_arrs:
            raise ValueError("Input array does not match correct shape (%s vs %s)" %
                             (str(inarr.shape), str(self.num_arrs)))
        for index, weight in enumerate(inarr.data):
            # Loop over all of the coefs and sum them up.
            mult = self.X_array[inarr.row[index]].multiply(self.Y_array[inarr.col[index]])
            mult.data = numpy.sqrt(mult.data) * weight
            sp_coo_append(self.calc_fluence, mult)
        X, Y = inarr.nonzero()
        for index, weight in enumerate(inarr[X,Y]):
            # (Done in compute_point now)
            mult = self.X_array[X[index]].multiply(self.Y_array[Y[index]])
            mult.data = numpy.sqrt(mult.data) * weight
            sp_coo_append(self.calc_fluence, self.compute_point(X[index], Y[index], weight))
    return self.calc_fluence

def compute_point_XY(self, x, y, weight):
    """Computes the weight and returns csr sparse array. """
mult = self.X_array[x].multiply(self.Y_array[y])
mult.data = numpy.sqrt(mult.data) * weight
return mult

def compute_point_points(self, x, y, weight):
    """ Computes the weight and returns csr sparse array. """
    if self.coeefs_shape != self.paramsp_shape:
        raise ValueError('X_array and Y_array differ in length.
        """
    ps_X = int(x * self.cf_to_sp)
    ps_Y = int(y * self.cf_to_sp)
    mod X = int((x * self.cf_to_sp) - ps_X) * self.cf_to_fl
    mod Y = int((y * self.cf_to_sp) - ps_Y) * self.cf_to_fl
    ret_array = self.XY_arrays[ps_X, ps_Y].tocoo(copy=True)
    ret_array.data[invalid_shifts] = 0
    ret_array.data[invalid_shifts] = 0
    ret_array.data[invalid_shifts] = 0
    return ret_array.tocsr()

def set_arrays(self, paramsp_arrays):
    """ Sets the parameter space arrays, calling the appropriate set
    function depending on the inputs. """
    if isinstance(tuple, paramsp_arrays):
        if len(paramsp_arrays) == 2:
            self.set_arrays_xy(*paramsp_arrays)
        else:
            raise ValueError("Unsupported number of paramspace arrays\n"provided (%s) & len(paramsp_arrays))
    else:
        self.set_arrays_points(paramsp_arrays)

def set_arrays_points(self, paramsp_arrays):
    """ Sets the list of lists of sparse matrices of the parameter space. """
    self.XY_arrays = numpy.asanyarray(paramsp_arrays)
    self.array_shape = self.XY_arrays[0,0].shape
    self.num_arrys = len(self.XY_arrays)
    self.paramsp_shape = self.XY_arrays.shape
    assert numpy.equal(self.XY_arrays.shape)
    self.acquire_fluence = sparse.coo_matrix(self.array_shape, dtype=float)
    self.max_size = 10 * numpy.multiply(*self.array_shape)
    self.compute_point = self.compute_point_points

def set_arrays_xy(self, X_array, Y_array):
    """ Set the arrays. This function also sets the array_shape and num_arrys.
    Keyword arguments:
    X array - See class header.
    Y array - See class header.
    Raises:
    ValueError - raised if the shapes are not identical. """
    self.X_array = X_array
    self.Y_array = Y_array
    self.array_shape = X_array[0].shape
    self.num_arrys = len(X_array)
    if len(Y_array) != self.num_arrys:
        raise ValueError("X_array and Y_array differ in length.
self.paramsp_shape = (self.num_arrys, self.num_arrys)
for i in range(self.num_arrys):
    if (self.array_shape != X_array[i].shape or
def read_parameter_space_file(self, params_filename):
    
    # Loads the file and reads the set of X and Y arrays for deconvolution.
    Inputs:
    filename
    Outputs:
    None
    Side effects:
    Sets self.array_shape, self.num_arrs, and self.points.
    Throws ValueError on reading invalid data.
    
    try:
        infile = openfile(params_filename, 'rb')
        indata = cPickle.load(infile)
    except cPickle.UnpicklingError as e:
        raise ValueError("Error unpickling data, aborting. (Error: %s)"
                         % e.strerror)
    x_pts, y_pts = indata[:2]
    if len(indata) == 4:
        # We have X,Y points and X,Y arrays.
        X_array, Y_array = indata[2:]
        self.set_arrays_XY(X_array, Y_array)
    elif len(indata) == 3:
        point_arrays = indata[2]
        self.set_arrays_points(point_arrays)
    self.points = (x_pts, y_pts)

def main(argv=None):
    
    display=False
    benchmark=True

    if len(sys.argv) < 2:
        sys.stderr.write("Usage: %s input_image [output_image].\n" % sys.argv[0])
        return()
    fn_readfile = sys.argv[1]

    start = time()
    
    # coefs = numpy.zeros((1024,1024))
    # coefs[511,511] = 1
    # coefs = numpy.zeros((9,9))
    # coefs[4,4] = 0
    # coefs[1,1:8] = 2
    # coefs = numpy.array([[ 1, 2, 3, 4, 5, 6, 7, 8,99],
    #                       [ 2, 3, 4, 5, 6, 7, 8,9,99],
    #                       [ 3, 4, 5, 6, 7, 8,9,10,99],
    #                       [ 4, 5, 6, 7, 8,9,10,11,99],
    #                       [ 5, 6, 7, 8,9,10,11,12,99],
    #                       [ 6, 7, 8,9,10,11,12,13,99],
    #                       [ 7, 8,9,10,11,12,13,14,99],
    #                       [ 8,9,10,11,12,13,14,15,99],
    convolver = FluenceConvolution(params_filename=fn_readfile)
coefs = numpy.asarray(numpy.arange(convolver.num_arrs**2),
                      dtype=float).reshape((convolver.num_arrs,-1))

output_image = convolver.compute_array(coefs).A

if benchmark: end = time()
if benchmark: print('total time: %f s' % float(end-start))

if benchmark:
    n=10
    RANGE=range(n)
    start = time()
    for i in RANGE:
        x,y = numpy.random.randint(0,9,2)
        coefs[x,y] = (numpy.random.randint(0,2,1) *
                      numpy.random.random(1))
        convolver.compute_array(coefs)
    end = time()
    if benchmark:
        r_start = time()
        for i in RANGE:
            x,y = numpy.random.randint(0,9,2)
            coefs[x,y] = (numpy.random.randint(0,2,1) *
                          numpy.random.random(1))
        r_end = time()
        tt=float(end-start)
        rt=float(r_end-r_start)
        nt=tt-rt
        cs=float(n)/nt
        print('for %i loops-- total time: %f s, rand time: %f s, net time: %f s.'
             'Calcs/s: %f' % (n, tt, rt, nt, cs))

if display:
    xmi = convolver.points[0][0] * (1024.0 / 1023)
    xma = convolver.points[0][-1] * (1024.0 / 1023)
    ymi = convolver.points[1][0] * (1024.0 / 1023)
    yma = convolver.points[1][-1] * (1024.0 / 1023)
    extents=[xmi,xma,ymi,yma]
    plt.figure(0)
    plt.title('Input Fluence')
    plt.imshow(coefs, extent=extents, interpolation='nearest')
    plt.colorbar()
    plt.figure(1)
    plt.title('Output Fluence')
    plt.imshow(output_image, extent=extents, interpolation='nearest')
    plt.colorbar()
    plt.figure(2)
    plt.title('Manip. Output Fluence')
    plt.imshow(convolver.result.A, extent=extents, interpolation='nearest')
    plt.colorbar()
    # from IPython.Shell import IPShellEmbed
    # IPShellEmbed()()
    plt.show()

return 0

if __name__ == '__main__':
    from time import time
    status = main()
    sys.exit(status)
#!/usr/bin/python

""" Compute the fluence at exit using the weights of the fluence at entrance. """

import numpy
from scipy import sparse
from scipy.signal import correlate2d
from utils import openfile, sp_coo_append, ReturnOnKeyboardInterrupt
from fluence_convolution import FluenceConvolution
from multiprocessing import Process, JoinableQueue, Queue
from multiprocessing.managers import BaseManager
from signal import signal, SIG_IGN, SIGINT, alarm, SIGALRM
from os import getcwd, chdir
from time import time as otime
from functools import partial
from socket import gethostname
from os import getpid
import sys

COMPUTE_JOB = 'compute_array'
DERIVATIVE_JOB = 'compute_ddx_arr'
SET_VARIABLES = 'set_variables'

def LOGprint(*args):
    #print(args[0])
    #sys.stdout.flush()
    return

def timeout_exception(signum, frame):
    raise AssertionError("Gave up waiting...")

def time():
    return otime() - 1334273792

def FIVE_PT_STENCIL(dx):
    return([[2*dx, -(1/(12 * dx))],
            [dx, (2/(3 * dx))],
            [-dx, -(2/(3 * dx))],
            [-2*dx, -(1/(12 * dx))]])

def CENTRAL_LIMIT_STENCIL(dx):
    return([[dx, (1/(2 * dx))],
             [-dx, -(1/(2 * dx))]])

def NEWTON_FORWARD_STENCIL(dx):
    return([[dx, 1/dx],
             [0, -1/dx]])

def YieldPairs(low, skip, end):
    for high in range(skip, end+skip, skip):
        if high>end: high=end
        yield low, high

def get_residual_der(flm, kernel, prev_res_sq):
    # Return the square root sum of the residuals.
    # Ensure that the kernel is a sparse coo matrix.
    kernel=sparse.coo_matrix(kernel, copy=False)
class QueueManager(BaseManager): pass

@ReturnOnKeyboardInterrupt
def parallel_worker(cfn, shapes, shared_buffers, queues):
    (task_jqueue, result_queue, master_jqueue) = queues
    (calc_fluence_shape, coefs_shape) = shapes
    (rw_shared, ro_shared) = shared_buffers

    calc_fluence_buflen = numpy.prod(calc_fluence_shape)
    coef_space_buflen = numpy.prod(coefs_shape)

    # Worker function for deconvolution/comparison.
    worker_id = current_process().name
    cfl_coo = sparse.coo_matrix((calc_fluence_shape, dtype=float))
    calc_fluence = numpy.frombuffer(rw_shared, count=calc_fluence_buflen, dtype=float).reshape(calc_fluence_shape)
    coef_space = numpy.frombuffer(rw_shared, count=coef_space_buflen, dtype=float).reshape(coefs_shape)

    # ro_shared is read only (unenforced...but it shouldn't change) and filled with two
    # datasets, the [fluence] [M]inus [p]revious [c]alculated [fl]uence (flMpcfl) of length
    # calc_fluence_buflen and the current guess coefficient space (coefs_ro) of
    # length coef_space_buflen offset from the start by calc_fluence_buflen
    flMpcfl = numpy.frombuffer(ro_shared, count=calc_fluence_buflen, dtype=float).reshape(calc_fluence_shape)
    coef_space_ro = numpy.frombuffer(ro_shared, count=coef_space_buflen, offset=calc_fluence_buflen, dtype=float).reshape(coefs_shape)

    for master_cmd in iter(master_jqueue.get, 'DIE'):
        if COMPUTE_JOB in master_cmd:
            cfl_coo.col.resize(0)
            cfl_coo.row.resize(0)
            cfl_coo.data.resize(0)
            calc_fluence.flat[:] = 0
        for X,Y,weights_arr in iter(master_jqueue.get, 'DONE'):
            task_jqueue.task_done()
        for i,weight in enumerate(weights_arr):
            sp_coo_append(cfl_coo, cfn(X[i],Y[i],weight))
            calc_fluence.flat[:] = cfl_coo.A.flat[:]
        result_queue.put(worker_id)
        task_jqueue.task_done()

    elif DERIVATIVE_JOB in master_cmd:
        # Before starting here, the master will have loaded the ro_shared
        # buffer with the previous good value, and we will have the current
        # coef_space loaded aswell for us. For each entry, we will replace
        # the current coef_space value in the array with the calceld
        # residual change.
        coef_space.flat[:] = 0
        # Grab the previous residual value (squared).
        prev_res_sq = master_cmd[1]
        for Xmin,Xmax,Ymin,Ymax,diff_val,diff_mult in iter(master_jqueue.get, 'DONE'):
            task_jqueue.task_done()
        for x in xrange(Xmin,Xmax):
            # for y in xrange(Ymin,Ymax):
                # Calculate the change in array.
weight = coef_space_ro[x,y] + diff_val

coef_space[x,y] += get_residual_der(flMpcfl, cfn(x, y, diff_val), prev_res_sq) * diff_mult

result_queue.put(worker_id)
task_jqueue.task_done()

else:
    master_jqueue.task_done()

class ParallelFluenceConvolution(FluenceConvolution):
    
    Class for convolution of fluence from weights at entrance.

    Attributes:
    array_shape - Size of the output array in (pixels, pixels) format.
    num_arrs - Number of arrays and size of side of coefs input.
    X_array - Array of length num_arrs of array_shape grids.
    Y_array - Array of length num_arrs of array_shape grids.

    Functions:
    __init__(param_arrays=None, params_filename=None)
    set_arrays(X_array, Y_array)
    read_parameter_space_file(params_filename='params.spparam.gz')
    compute_array(coefficients)

    # Globals
    NODE_THRESH = 10
    WORKER_THRESH = 10
    NODE_GRANULARITY = 4
    WORKER_GRANULARITY = 1

    # Multi-process processing information
    workers = None
    n_workers = 0
    array_buffer = None
    numpy_array_buffer = None
    buf_size = 0

    # Multi-node processing information
    PBS = False
    am_master = False
    master_address = None
    nodes = None
    nodes_procs = None
    nnodes = 0

    # Queues (jqueue => JoinableQueue)
    master_jqueue = None
    task_jqueue = None
    results_queue = None
    node_master_jqueue = None
    node_task_jqueue = None
    node_results_queue = None

    # Arrays
    coef_space_der = None  # Coefficient space derivative approximation.
    flMpcfl = None  # Fluence minus prev. calculated fluence
    prev_coefs = None  # Previous coefficients

    # Write file for state saving
    fn_partfile = None

    def __init__(self, param_arrays=None, params_filename=None, master_address=None, authkey='insecure', nodes=None, **kwargs):
        
        Initialize the class, setting the parameters if passed.

        Keyword arguments:
        param_arrays - Set of two arrays to be used for X_array, Y_array.
        params_filename - File to read parameters from using read_parameter_arrays_file
FluenceConvolution.__init__(self, param_arrays, params_filename)

LOGprint("%9.2f: __init__: starting convolver" % time())

# Set the authkey now so it can be changed later if needed.
self.authkey = authkey
self.master_address = master_address

# Determine node environment. Also determine if we should be the
# master. Only the master will have nodes set, or in a PBS environment
# will have the environment variable 'PBS_NODEFILE' containing the list
# of nodes assigned to the task.
if getenv('PBS_ENVIRONMENT') == 'PBS_BATCH':
    self.PBS = True
    # Get some parameters.
    self.authkey = getenv('PBS_JOBCOOKIE')
    nodefile = getenv('PBS_NODEFILE')
    if nodefile != None:
        # We are the master, get the list of nodes.
        with open(nodefile, 'r') as nodefile:
            nodes = nodefile.read().splitlines()
            if not nodes:
                raise AttributeError("PBS_NODEFILE did not contain any nodes!")
        if nodes is not None:
            # With the list of nodes, make a dict counting the number of times
            # they appear. When called from the command line, this is
            # irrelevant, when called from PBS, this is the number of processes we
            # are allowed per node.
            from collections import defaultdict
            from socket import gethostname
            # We are the master.
            #print("%9.2f: nodes before: %s" % (time(), nodes))
            self.am_master = True
            self.nodes = defaultdict(int)
            for line in nodes:
                self.nodes[line] += 1
            # If we are in the list...
            if gethostname() in self.nodes:
                # Spare one proc for runner...maybe...
                self.n_workers = self.nodes.pop(gethostname()) - 2
                # Nah...
                self.n_workers = self.nodes.pop(gethostname())
            if 'localhost' in self.nodes:
                #print("%9.2f: self.nodes: %s" % (time(), self.nodes))
                self.nnodes = len(self.nodes)
                #print("%9.2f: self.nnodes: %s" % (time(), self.nnodes))
        if not self.PBS:
            # PBS should tell us how many processors we can use.
            # Otherwise, we use all but one.
            # Spare one proc for runner...maybe...
            #self.n_workers = cpu_count()-1
            # Nah...
            self.n_workers = cpu_count()

self.workers = []

# Create the queues for the local environment. (Queues for the
# multi-node environment are handled elsewhere)
self.master_jqueue = JoinableQueue()  
self.task_jqueue = JoinableQueue()
self.results_queue = Queue()

self.array_buffer = []
self.shared_ro_buf = None
self.worker_shared_fluence_arrays = dict()
self.worker_shared_coef_arrays = dict()

self.calc_fluence_shape = self.array_shape

if 'coefs_shape' in kwargs:
    self.coefs_shape = kwargs['coefs_shape']
else:
    self.coefs_shape = self.paramsp_shape

self.calc_fluence buflen = numpy.prod(self.calc_fluence_shape)
self.coef_space buflen = numpy.prod(self.coefs_shape)

# First buffer is large enough to hold either the coefficient space or
# the current fluence.
self.buf_size = max(self.calc_fluence buflen, self.coef_space buflen)

# Second buffer is large enough to hold both the coefficient space and
# the current fluence. ONLY ONE PER MASTER.
self.buf2_size = self.calc_fluence buflen + self.coef_space buflen

# Prepare the buffers.
self.shared_ro_buf = Array('d', self.buf2_size, lock=False)

self.fMpcfl = numpy.frombuffer(self.shared_ro_buf,
count=self.calc_fluence buflen,
dtype=float).reshape(self.calc_fluence_shape)

self.prev_coefs = numpy.frombuffer(self.shared_ro_buf,
count=self.coef_space buflen, offset=self.calc_fluence buflen,
dtype=float).reshape(self.coefs_shape)

# If we happen to be passed these
if 'fMpcfl' in kwargs:
    self.fMpcfl_flat[:] = kwargs['fMpcfl']
elif 'fluence' in kwargs:
    self.fMpcfl_flat[:] = kwargs['fluence']
if 'prev_coefs' in kwargs:
    self.prev_coefs_flat[:] = kwargs['prev_coefs']
if 'fn writeFile' in kwargs and self.am master:
    self.fn_partfile = kwargs['fn writeFile'] + "\part"

# Coefficient space derivative.
self.coef_space_der = numpy.zeros(self.coefs_shape, dtype=float)

self.cf_to_sp = float(self.paramsp_shape[0] - 1) / self.coefs_shape[0]
self.cf_to_fl = float(self.calc_fluence_shape[0]) / self.coefs_shape[0]
print('Got exception when trying to clean up: %s' % e)
finally:
sys.stdout.flush()

def launch_workers(self):
    """Sets up and launches all workers, including creating shared memory
    segments for each worker. """
    # Unpacked in function by the following:
    #(task_jqueue, result_queue, master_jqueue) = queues
    #(calc_fluence_shape, coefs_shape) = shapes
    queues = (self.task_jqueue, self.results_queue, self.master_jqueue)
    shapes = (self.calc_fluence_shape, self.coefs_shape)
    for i in range(self.n_workers):
        # Worker signature and expected inputs.
        #parallel_worker(cfn, shapes, shared_buffers, queues):
        #(task_jqueue, result_queue, master_jqueue) = queues
        #(calc_fluence_shape, coefs_shape) = shapes
        #(rw_shared, ro_shared) = shared_buffers
        # ro_shared is read only (unenforced...but it shouldn't change) and filled with two
        # datasets, the [fluence] minus [previous] calculated [fluence] (flMpcfl) of
        # length calc_fluence_buflen and the current guess coefficient space (coefs_) of
        # length coef_space_buflen offset from the start by calc_fluence_buflen
        # Allocate new buffer.
        new_buf = Array('d', self.buf_size, lock=False)
        shared_buffers = (new_buf, self.shared_ro_buf)
        new_worker = Process(target=parallel_worker,
                              args=(self.compute_point, shapes, shared_buffers, queues))
        new_worker.daemon = True
        new_worker.start()
        self.worker_shared_fluence_arrays[new_worker.name] = numpy.frombuffer(new_buf,
                                                                              count=self.calc_fluence_buflen,
                                                                              dtype=float).reshape(shapes[0])
        self.worker_shared_coef_arrays[new_worker.name] = numpy.frombuffer(new_buf,
                                                                              count=self.coef_space_buflen,
                                                                              dtype=float).reshape(shapes[1])
        self.workers.append(new_worker)
        self.array_buffer.append(new_buf)

def _launch_nodes(self):
    if self.am_master:
        # Set up the queues. We need them even if we don't have nodes.
        # We have nodes, and are the master. Start the shared queues.
        self.node_master_jqueue = JoinableQueue()
        self.node_task_jqueue = JoinableQueue()
        self.node_results_queue = Queue()
    if self.nodes and self.am_master:
        # Set up the sharing, etc.
        QueueManager.register('get_master_jqueue',
                              callable=lambda: self.node_master_jqueue)
        QueueManager.register('get_task_jqueue',
                              callable=lambda: self.node_task_jqueue)
        QueueManager.register('get_results_queue',
                              callable=lambda: self.node_results_queue)
        self.server = QueueManager(address=self.master_address,
                                     authkey=self.authkey)
        self.server.start()
        # Fill the task queue with the parameters needed for the nodes to
# Build a dict to hold the information.
kwargs = dict()
if self.PBS:
    kwargs['nodes'] = self.nodes.copy()
    kwargs['flat'] = self.flat.copy()
    kwargs['prev_coeffs'] = self.prev_coeffs.copy()
    for node in range(self.nnodes):
        self.node_task_jqueue.put(SET_VARIABLES)
    for node in range(self.nnodes):
        self.node_task_jqueue.put(kwargs)

if not self.PBS:
    # Not using PBS, launch the nodes using a subprocess and rsh, then
    # store the node in the self.nodes dictionary under the name.
    self.nodes Proc = []
    from subprocess import Popen, STDOUT, PIPE
    from os import getcwd
    from sys import argv
    from socket import gethostname
    # Launch all of the nodes
    for node_name in self.nodes:
        #node_cmd = "cd %s; screen -L -m -D %s -P %s %s" % (getcwd(),
        node_cmd = "cd %s; %s -D %s -P %s %s" % (getcwd(),
        argv[0], gethostname(),
        self.server.address[1], '`join`\"`argv[1]\"`)
        rsh = "ssh"
        #print("%9.2f: launching on %s: %s" % (time(), node_name, node_cmd))
        logfile = open("%s\"%s.txt\" % node_name, 'w+\")
        #logfile=open("%s\"%s\")
        self.nodes Proc.append(Popen([rsh, '-tt', node_name, node_cmd],
        stdout=logfile, stderr=STDOUT, stdin=PIPE))

self.nodes clear()
# When they get the master queue connection, they send:
# (node name, main_pid)
killfile = open("/tmp/ll kill.%i.sh\" % getpid(), 'w+\")
killfile.write("!/bin/bash\n")
for node in range(self.nnodes):
    (node name, main_pid) = self.node_results_queue.get()
    if node name not in self.nodes:
        self.nodes[node name] = main_pid
        killfile.write("\nssh %s "kill -SIGINT %i 2>/dev/null\") % (node name,
        main_pid))
    LOGprint("%9.2f: MASTER: Node started: %s (pid: %s)\" % (time(), node name,
        main_pid))
    # Wait for the nodes to finish getting set up.
    self.node_task_jqueue.join()

elif self.am master:
    # Master but no nodes, we will use the queues anyway.
    self.node master_jqueue = JoinableQueue()
    self.node_task_jqueue = JoinableQueue()
    self.node_results_queue = Queue()

elif not self.am master:
    # We have nodes but aren't the master, we must be a node. Start the
    # receivers for the shared queues.
    LOGprint("%9.2f: NODE: Not the master, getting queue objects" % time())
    QueueManager.register('get_master_jqueue')
    QueueManager.register('get_task_jqueue')
    QueueManager.register('get_results_queue')

    self.master = QueueManager(address=self.master_address,
        authkey=self.authkey)
    self.master.connect()

    self.node master_jqueue = self.master.get_master_jqueue()
    self.node_task_jqueue = self.master.get_task_jqueue()
self.node_results_queue = self.master.get_results_queue()

    def _kill_workers(self, timeout=0.1):
        # Workers are set as daemon, they die when we end.
        self.workers = None

    def _kill_nodes(self, timeout=2):
        print("%9.2f: %s: Entered kill nodes. %s, am: %s" % (time(), gethostname(), self.nodes, self.am_master))
        if self.nodes is not None and self.am_master:
            # We are a master and have nodes. Send the die command to the master queue.
            # We will wait timeout seconds for the nodes to die, then kill them, if we can.
            prev_SIGALRM = signal(SIGALRM, timeout_exception)
            alarm(timeout)
            timedout = False
            try:
                self.node_task_jqueue.join()
                for node in range(self.nnodes):
                    LOGprint("%9.2f: MASTER: Waiting for nodes to die..." % time())
                    self.node_master_jqueue.join()
            except AssertionError:
                LOGprint("%9.2f: MASTER: Gave up waiting for nodes to die. Killing." % time())
                timedout = True
            finally:
                alarm(0)
                signal(SIGALRM, prev_SIGALRM)

        if timedout:
            # The nodes wouldn't listen...we shall make them listen.
            from subprocess import Popen
            if self.nodes_procs is not None:
                for i, process in enumerate(self.nodes_procs):
                    LOGprint("%9.2f: MASTER: Node %s didn't die, killing harshly" % (time(),
                        self.nodes[i]))
                    process.send_signal(SIGINT)
            else:
                for node_name in self.nodes:
                    LOGprint("%9.2f: MASTER: Node %s didn't die, killing harshly" % (time(),
                        node_name))
                    node_cmd = """kill -SIGINT %i""" % int(self.nodes[node_name])
                    rsh = "ssh" self.nodes[node_name] = Popen([rsh, node_name, node_cmd])

        self.nodes = None
        self.nnodes = 0

    def _wake_workers(self, job):
        """ Get workers ready for job ""
        run_range = range(self.n_workers)
        for index in run_range:
            self.master_jqueue.put(job)

    def _sleep_workers(self):
        """ Job is done, put workers to sleep. ""
        run_range = range(self.n_workers)
        for index in run_range:
self.task_jqueue.put('DONE')

def _sleep_nodes(self):
    """ Job is done, put nodes to sleep. """
    run_range = range(self.nnodes)
    self.node_task_jqueue.join()
    for node in run_range:
        self.node_task_jqueue.put('DONE')
    sys.stdout.flush()   

def _wake_nodes(self, job):
    """ Get nodes ready for job """
    run_range = range(self.nnodes)
    self.node_task_jqueue.join()
    # Get everyone ready for the job
    for node in run_range:
        #print("%9.2f: MASTER: _wake_nodes: %s sent job: %s" % (time(), node, job))
        self.node_master_jqueue.put(job)

def save_partial(self):
    """ Saves the current coefs for fault tolerance. """
    if self.fn_partfile is not None:
        numpy.savez(self.fn_partfile, coefs=self.prev_coefs,
                    calc_fluence=self.calc_fluence.A)

def compute_array(self, coefficients, restart=False):
    """ Compute an output array based on the input coefficients. 
    Keyword arguments:
    coefficients - a 2d array of [num_arrs, num_arrs] consisting of the
                   weights assigned to each section of the fluence.
    restart - Force full recomputation, even if we have already run a calc
             already. (Useful if the coefs have changed drastically).
    Returns:
    An array of array_shape containing the computed output.
    Raises:
    ValueError if coefficients.shape is not the same as [num_arrs, num_arrs].
    """
    if not self.am_master:
        raise AttributeError("Cannot call this function on a worker node.")

    # If we have loaded the calculated fluence in self.flMpcfl we should
    # return to the original value before changing the value of self.calc_fluence
    # If self.calc_fluence is empty, we should be fine. Special case: if we are
    # restarting, we will assume that self.flMpcfl has been set to the
    # fluence fresh.
    if restart or self.prev_coefs == None:
        # Copy the incoming array and set the shape
        self.prev_coefs = coefficients.reshape(self.coefs_shape).copy()
        coef_delta = self.prev_coefs.copy()
        self.calc_fluence.col.resize(0)
        self.calc_fluence.row.resize(0)
        self.calc_fluence.data.resize(0)
    else:
        coef_delta = coefficients.reshape(self.prev_coefs.shape) - self.prev_coefs
        self.prev_coefs.flat[:] = coefficients

    if coef_delta.shape[0] < self.num_arrs or coef_delta.shape[1] < self.num_arrs:
        raise ValueError("lower resolution coefficient space not supported yet.")
    elif coef_delta.shape != self.coefs_shape:
        raise ValueError("Input array does not match correct shape (%s vs %s)" %
                         (str(coef_delta.shape), str(self.num_arrs)))
X, Y = coef_delta.nonzero()
num_elements = len(X)
run_range = range(self.nnodes)
LOGprint("%9.2f: MASTER: ARR: Beginning calculation, n: %s" % (time(), num_elements))

if num_elements == 0 and not restart:
    # No change and we are not restarting.
    LOGprint("%9.2f: MASTER: ARR: No change, not recalculating array." % time())
    return self.calc_fluence

elif self.calc_fluence.nnz != 0:
    # We have a previous calculation, un-subtract it from flMpcfl.
    LOGprint("%9.2f: MASTER: ARR: reverting flMpcfl to fluence" % time())
    self.flMpcfl += self.calc_fluence

if num_elements < self.NODE_THRESH * (self.nnodes + 1) or self.nnodes == 0:
    LOGprint("%9.2f: MASTER: ARR: Not using nodes, master calculating." % time())
    run_range = []

calc_fluence_delta = sparse.coo_matrix(self.calc_fluence.shape)

# Set up some criteria.
cpn = (num_elements // ((self.NODE_GRANULARITY * len(run_range)) + 1))

#print("%9.2f: MASTER: Preparing nodes..." % time())
if run_range:
    self._wake_nodes(COMPUTE_JOB)

#print("%9.2f: MASTER: Sending job to cluster. (%i)" % (time(), self.nnodes))
for beg, end in YieldPairs(0, cpn, num_elements):
    #print("%9.2f: MASTER: ARR: Sending to nodes %i:%i of %i" % (time(), beg, end, num_elements))
    lX = X[beg:end]
    lY = Y[beg:end]
    self.node_task_jqueue.put((lX, lY, coef_delta[lX,lY]))

# Compute job for the master.
try:
    self._compute_array(lX, lY, seq_data)
    self.node_task_jqueue.task_done()
    #LOGprint("%9.2f: MASTER: ARR: done with job: %s, %s, %s" % (time(),lX,lY,seq_data))
except Empty:
    # Now the queue is empty, as we were the only producer, we can tell
    # everyone else to sleep.
    LOGprint("%9.2f: MASTER: queue empty. we can move on." % (time()))
    # Tell the workers they are done.
    self._sleep_workers()
    #LOGprint("%9.2f: MASTER: ARR: Done with job: %s, %s, %s" % (time(),lX,lY,seq_data))
except:
    LOGprint("%9.2f: MASTER: Unhandled exception: %s" % (time(), sys.exc_info()))
    raise

LOGprint("%9.2f: MASTER: Finished reading, putting nodes to sleep." % time())
if run_range:
    self._sleep_nodes()

LOGprint("%9.2f: MASTER: Nodes asleep. Compiling answers." % time())

# Get the results from the nodes.
for node in run_range:
    sp_coo_append(calc_fluence_delta, self.node_results_queue.get(), self.max_size)
# Adjust the \texttt{flMpcfl} by the change we just computed.

\texttt{LOGprint("%.2f: MASTER: ARR: setting flMpcfl =\texttt{ calc_fluence\_delta" \% time())}

\texttt{self.flMpcfl =\texttt{ calc\_fluence\_delta}}

# Append the change to the \texttt{calc\_fluence}

\texttt{return sp\_coo\_append(self.calc\_fluence,}
\texttt{calc\_fluence\_delta,}
\texttt{self.max\_size)}

\begin{verbatim}
def compute_array(self, lX, lY, seq_data):
    """ Iterates over seq\_data and computes data. ""
    num_elements = len(seq_data)
    run_range = range(self.n_workers)

    if num_elements < self.WORKER_THRESH * self.n_workers:
        # Don't bother with full workers if we aren't computing a lot.
        for index, weight in enumerate(seq_data):
            sp_coo_append(self.calc\_fluence, self.compute\_point(lX[index], lY[index], weight))
        return self.calc\_fluence

    # Calcs per worker. Split up extra fine to load balance
    cpw = (num_elements // (self.WORKER_GRANULARITY * self.n_workers))

    for beg, end in YieldPairs(0, cpw, num_elements):
        self.task\_jqueue.put((lX[beg:end], lY[beg:end], seq_data[beg:end]))

    self.task\_jqueue.join()
\end{verbatim}

\begin{verbatim}
def get_compute_array_results(self, result):
    """ Gets the results of a compute\_array job from the local workers ""
    run_range = range(self.n_workers)

    # Get the results from the shared array.
    for index in run_range:
        sp_coo_append(result,
                      self.worker\_shared\_fluence\_arrays[self.results\_queue.get()],
                      self.max\_size)
    return result
\end{verbatim}

\begin{verbatim}
def compute_derivative(self, stencil=FIVE\_PT\_STENCIL, dx=1e-8, prev\_res\_sq=0):
    """ Computes the derivative of the current coef space. ""
    # Set up the finite difference stencil (default to a fairly accurate
    # five point stencil)
    stencil = stencil(dx)

    run_range = range(self.nnodes)

    LOGprint("%.2f MASTER: DER: stencil is: %s" \% (time(), stencil))

    num_elements = self.coefs\_shape[0]
    # If we don't have nodes, we run it ourselves.
    #if not self.nodes:
        # for diff\_val, diff\_mult in stencil:
        #    task = (0, num\_elements, diff\_val, diff\_mult)
        #    self\_compute\_derivative(*task)
    # return self.coef\_space\_der

    # Ensure the children nodes have the most current data.
    kwargs = dict()
    kwargs['flMpcfl'] = self.flMpcfl.flat[:]
    kwargs['prev\_coefs'] = self.prev\_coefs.flat[:]
    for node in run_range:
        self.node\_master\_jqueue.put(SET\_VARIABLES)
    # Wait for the nodes.
    for node in run_range:
        self.node\_task\_jqueue.put(kwargs)
\end{verbatim}
self._wake_nodes((DERIVATIVE_JOB, prev_res_sq))

cpn = (num_elements // ((self.NODE_GRANULARITY * len(run_range)) + 1))
for diff_val, diff_mult in stencil:
    # For each item in the stencil, run the entire coefficients.
    for Y_beg, Y_end in YieldPairs(0, cpn, num_elements):
        # The job will be along this small slice.
        task = (Y_beg, Y_end, diff_val, diff_mult)
        print("%9.2f: MASTER: DER: putting in node_task_jqueue: X(0:end) \\
        Y(%s:%s) dv=%s dm=%s % ((time()), + task))
        self.node_task_jqueue.put(task)

LOGprint("%9.2f: MASTER: DER: Processing myself." % time())

try:
    self._wake_workers((DERIVATIVE_JOB, prev_res_sq))
    for task in iter(self.node_task_jqueue.get_nowait, ' '):
        print("%9.2f: MASTER: DER: working on job: X(0:end) \\
        Y(%s:%s) dv=%s dm=%s % ((time()), + task))
        self._compute_derivative(*task)
        self.node_task_jqueue.task_done()
        print("%9.2f: MASTER: DER: done with job: X(0:end) \\
        Y(%s:%s) dv=%s dm=%s % ((time()), + task))
except Empty:
    # Now the queue is empty, as we were the only producer, we can tell
    # everyone to sleep and pull off the results from our own
    # workers.
    self._sleep_workers()
    self._get_derivative_results()
    pass
except:
    #print("%9.2f: MASTER: DER: Unhandled exception: %s" % (time(), sys.exc_info()))
    raise

LOGprint("%9.2f: MASTER: DER: Finished derivative compute, putting nodes to sleep." % time())
self._sleep_nodes()

LOGprint("%9.2f: MASTER: DER: Nodes asleep. Compiling answers." % time())
for index in run_range:
    self.coef_space_der += self.node_results_queue.get()
# Store the negative of the current fluence in our shared array (for
# use in derivative calcs)
    return self.coef_space_der.copy()

def _get_derivative_results(self):
    """ Finished compute derivative job, get responses. """

    self.coef_space_der.flat[:] = 0
    run_range = range(self.n_workers)
    for index in run_range:
        self.coef_space_der += self.worker_shared_coef_arrays[self.results_queue.get()]

def _compute_derivative(self, Ymin, Ymax, diff_val, diff_mult):
    """ Iterates over seq_data and computes data. """
    num_elements = self.coefs_shape[0]
    run_range = range(self.n_workers)

    # Calcs per worker. Split up extra fine to load balance
    cpw = (num_elements // (self.WORKER_GRANULARITY * self.n_workers))

    for X_beg, X_end in YieldPairs(0, cpw, num_elements):
        # Loop through the X elements.
        task = (X_beg, X_end, Ymin, Ymax, diff_val, diff_mult)
        self.task_jqueue.put(task)

        # Wait until all of the tasks have been grabbed for computation.
        self.task_jqueue.join()
```python
def child_node_loop(self):
    """ Child node for mpi processing of data."""

    from socket import gethostname
    from os import getpid

    hostname = gethostname()

    # Report back to the master that we have started. Do this on the results
    # queue so other nodes won't pull it off when they start up.
    self.node_results_queue.put(((hostname, getpid())))

    for master_cmd in iter(self.node_master_jqueue.get, 'DIE'):
        if COMPUTE_JOB in master_cmd:
            LOGprint("%9.2f: NODE (%s): Got job %s" % (time(), hostname, master_cmd))
            self.node_results_queue.put(self.calc_fluence() )
            self.node_results_queue.put(self.calc_fluence.data.resize(0))
            self._wake_workers COMPUTE_JOB )

            if iter(self.node_task_jqueue.get, 'DONE'):
                LOGprint("%9.2f: NODE (%s): ARR: received done, sending result: %r" % (time(), hostname, self.node_results_queue._get_compute_array_results(1)))
                LOGprint("%9.2f: NODE (%s): ARR: working on job: %s, %s, %s" % (time(), hostname, lX, lY, seq_data))
                LOGprint("%9.2f: NODE (%s): ARR: done with job: %s, %s, %s" % (time(), hostname, lX, lY, seq_data))
                LOGprint("%9.2f: NODE (%s): ARR: done sending result" % (time(), hostname))
                self._sleep_workers()

            elif DERIVATIVE_JOB in master_cmd:
                LOGprint("%9.2f: NODE (%s): DER: done with job: %s" % (time(), hostname, task))
                LOGprint("%9.2f: NODE (%s): SET VARIABLES in master_cmd" % (time(), hostname))

                vars_to_set = self.node_task_jqueue.get()  

                if item == 'prev_coefs':
                    self.prev_coefs[] = vars_to_set[item][]

                if item == 'nodes' and not self.workers:
                    if hostname in vars_to_set[item]:
                        self.n_workers = vars_to_set[item][hostname] - 1
                        Nah...
```
self.n_workers = vars_to_set[item][hostname]
    # Now we can launch our workers.
    self._launch_workers()
else:
    raise ValueError("Cannot handle item %s, val %s. %s",
        (item, vars_to_set[item], self.workers))
# print("%9.2f: NODE (%s): SET: done with item: %s, vts[item]: %s" % (time(),
hostname, item,
    # vars_to_set[item]))
LOGprint("%9.2f: NODE (%s): SET: done with setvars" % (time(), hostname))

self.node_master_jqueue.task_done()
LOGprint("%9.2f: NODE (%s): done with task '%s' in master_queue, waiting..." %
(time(), hostname, master_cmd))
    # Join the master and wait for all of the tasks of this type to
    # be finished by all nodes. (master is always set in a lump
    # before filling task, so it should never empty before everyone
    # is done)
    sys.stdout.flush()
self.node_master_jqueue.join()
LOGprint("%9.2f: NODE (%s): done waiting on node_master_jqueue!" % (time(),
hostname))

# Save the last node_task_jqueue.task_done() for until all of the
# nodes have finished. This way we make the master wait to give a
# new command until after all of the nodes are done.
self.node_task_jqueue.task_done()
# print("%9.2f: NODE (%s): Received DIE." % (time(), hostname))
self.node_master_jqueue.task_done()
# print("%9.2f: NODE (%s): After level of for master_cmd." % (time(), hostname))

def master(options, args):
    from time import time

display = options.display
benchmark = options.benchmark

    # Some defaults for testing.
    #nodes = ['blade%i' % (i+1) for i in range(16)]
    #nodes = ['localhost']
    nodes = ['blade1', 'blade2']

    start = time()

    kwargs = dict()

    if options.fn_spparam is None:
        kwargs['params_filename'] = args[0]
    else:
        kwargs['params_filename'] = options.fn_spparam

    if options.master_port:
        kwargs['master_address'] = ('', int(options.master_port))
    else:
        kwargs['master_address'] = ('', 50000)

    if options.fn_writefile:
        kwargs['fn_writefile'] = options.fn_writefile

coefs = None
if options.fn_fluence is not None:
    f_readfile = openfile(options.fn_fluence, 'rb')
    coefs = numpy.load(f_readfile)
    f_readfile.close()

kwargs['nodes'] = nodes

    if options.coefs_shape:
        kwargs['coefs_shape'] = (int(options.coefs_shape), int(options.coefs_shape))
    else:
```python
kwargs['coefs_shape'] = (coefs[2].shape[0]//4, coefs[2].shape[1]//4)

with ParallelFluenceConvolution(**kwargs) as convolver:
    if coefs is None:
        coefs = numpy.asarray(numpy.arange(numpy.multiply(convolver.coefs_shape)),
                               dtype=convolver.coefs_shape[0])
    else:
        fluence = coefs[2]
        threshold = 0.1
        avg_fluence = numpy.average(fluence>threshold *
                                 numpy.max(fluence))

        # Copy the output fluence, resize it to fit the needed input data.
        if fluence.shape != convolver.coefs_shape:
            # Array shapes not the same, make an interpolated fluence to fit the
            # dataset we are working with.
            # Smooth first
            from scipy.ndimage.filters import gaussian_filter
            from scipy.interpolate import RectBivariateSpline
            m = float(fluence.shape[0]) / (convolver.coefs_shape[0])
            t = RectBivariateSpline(range(len(fluence)), range(len(fluence)), fluence)
            def f(x, y, mu):
                return t.ev((mu*x).flatten(),(mu*y).flatten())
            fluence_guess = numpy.fromfunction(f,
                                               convolver.coefs_shape,
                                               m,m).reshape(convolver.coefs_shape)
            fluence_guess = fluence_guess[numpy.multiply(fluence>threshold *
                                                      avg_fluence)]
        else:
            fluence_guess = numpy.zeros(shape=convolver.coefs_shape)
            X,Y = (fluence > (threshold * avg_fluence)).nonzero()
            fluence_guess[X,Y] = fluence[X,Y]

        coefs = fluence_guess

        output_image = convolver.compute_array(coefs).A

    if benchmark: end = time()
    if benchmark: print('total time: %f s' % float(end-start))

if benchmark:
    # n=1
    # RANGE=range(n)
    # start = time()
    #for i in RANGE:
        # x,y = numpy.random.randint(0,9,2)
        # coefs[x,y] = (numpy.random.randint(0,2,1) *
        #               numpy.random.random(1))
        # convolver.compute_array(coefs)
    #end = time()
    #r_start = time()
```

```python
# for i in RANGE:
#    x,y = numpy.random.randint(0,9,2)
#    coefs[x,y] = (numpy.random.randint(0,2,1) * 
#          numpy.random.random(1))
# for i in RANGE:
#    coefs[:,:] = (numpy.random.randint(0,2,coefs.shape) * 
#          numpy.random.random(coefs.shape))
#    convolver.compute_array(coefs)
# end = time()
# r_start = time()
# for i in RANGE:
#    coefs[:,:]= (numpy.random.randint(0,2,coefs.shape) * 
#          numpy.random.random(coefs.shape))
# r_end = time()
# tt=float(end-start)
# rt=float(r_end-r_start)
# nt=tt-rt
# cs=float(n)/nt
# print('
for %i loops-- total time: %f s, rand time: %f s, net time: %f s.' 
#      'Calcs/s: %f' % (n, tt, rt, nt, cs))
# print("compute derivative: %s" % convolver.compute_derivative())
# end = time()
# print("donetime %s" % (start - end))
if display:
xmi = convolver.points[0][0] * (1024.0 / 1023)
xma = convolver.points[0][-1] * (1024.0 / 1023)
ymi = convolver.points[1][0] * (1024.0 / 1023)
yma = convolver.points[1][-1] * (1024.0 / 1023)
extents = [xmi,xma,ymi,yma]
plt.figure(0)
plt.title('Input Fluence')
plt.imshow(coefs, extent=extents, interpolation='nearest')
plt.colorbar()
plt.figure(1)
plt.title('Output Fluence')
plt.imshow(output_image, extent=extents, interpolation='nearest')
plt.colorbar()
plt.figure(2)
plt.title('Manip. Output Fluence')
plt.imshow(convolver.result.A, extent=extents, interpolation='nearest')
plt.colorbar()
from IPython.Shell import IPShellEmbed
IPShellEmbed()
plt.show()

LOGprint("MASTER: Done with job, returning.")
return 0

def node(options, args):
    ''' Worker node. '''
    convolver = None
    kwargs = dict()
    if options.fn_spparam is not None:
        kwargs['params_filename'] = options.fn_spparam
    else:
        kwargs['params_filename'] = args[0]
    if options.master_port:
        kwargs['master_address'] = (options.master, int(options.master_port))
    else:
        kwargs['master_address'] = (options.master, 50000)
    if options.coefs_shape:
        kwargs['coefs_shape'] = (int(options.coefs_shape), int(options.coefs_shape))
    retval = None
    try:
        # ...
with ParallelFluenceConvolution(**kwargs) as convolver:
    #print("%9.2f: Worker started, entering loop." % time())
    retval = convolver.child_node_loop()
    #print("%9.2f: NODE: done with child_node_loop(), retval: %s" % (time(),
    #    retval))

except KeyboardInterrupt as e:
    from socket import gethostname
    #print("Node %s killed with SIGINT." % gethostname())
    retval = e
    return retval

if __name__ == '__main__':
    import sys
    from os import getenv, environ, chdir
    # Find our id and branch based on that.
    from optparse import OptionParser

    usage = "%prog [options] input_dataset [outfile]"
    parser = OptionParser(usage=usage)
    parser.add_option("-d", "--display", action="store_true",
                      default=False,
                      help="Display plots at the end.")
    parser.add_option("-b", "--benchmark",
                      action="store_true",
                      default=False,
                      help="Display benchmarking information.")
    parser.add_option("-o", "--output",
                      action="store_true",
                      default=False,
                      help="Output to out-file.")
    parser.add_option("-f", "--out-file",
                      dest="fn_writefile",
                      help="File to store result in.")
    parser.add_option("-P", "--port",
                      dest="master_port",
                      help="Port of the master.")
    parser.add_option("-l", "--fluence-file",
                      dest="fn_fluence",
                      help="Fluence input to compute for")
    parser.add_option("-c", "--coefs_shape",
                      help="Shape of one side of the coefficient space [default: paramsp]"
                      )

    (options, args) = parser.parse_args()
    if len(args) < 1 and options.fn_spparam == None:
        parser.error("We must have the input fluence dataset.")
    sys.exit(1)

if getenv('PBS_ENVIRONMENT') == 'PBS_BATCH':
    # We are in a batch run, so we can work on nodes.
    # Enter the workdir.
    chdir(getenv('PBS_O_WORKDIR'))
    # Set the port based on the job number plus 50000.  This will keep
    # it well into user space and fairly safe.
    if not options.master_port:
        job_id = int(getenv('PBS_JOBID').split('.')[0])
        options.master_port = 50000 + (job_id % 15535)
    if getenv('PBS_NODEFILE') != None:
        options.node = False
    else:
        options.node = True

    if options.node:
        if not options.master:
            parser.error("We are a node but don't have a master. Aborting.")
sys.exit(1)
status = node(options, args)
else:
    status = master(options, args)
sys.exit(status)

F.5 fluence_solver.py

#!/usr/bin/python

""
Determine optimal deconvolution parameters for the fluence to fluence comparison.
Written By Nicholas Sperling, 2012
""

import numpy
from scipy import optimize, sparse
from scipy.interpolate import RectBivariateSpline
from multiprocessing import Process, JoinableQueue, Queue
from multiprocessing import Array, cpu_count, active_children
import Queue as stdQueue
import fluence_convolution
import fluence_convolution

# We will be using the module hist_deconvolution to accomplish this. This
# module will be used to seek the best fit for deconv_coefs given a set of
# reference field fluence/dose array pairs.

# Quality of the match will be determined by normalizing each pair to the
# average value of all pixels above a threshold of the maximum pixel
# value: AVG_THRESHOLD=20% by default.
# The two arrays will then be subtracted and the average deviation percentage
# in the thresholded region (in absolute) will be used as the quality term.
# This allows for a minimization iterative solver to find the optimal value.
# The quality terms for each input fluence/dose pair will be added in
# quadrature for the final term. This allows values that are significantly off
# to have a large effect on the final outcome.

def get_residual(conv_func, fluence, guess):
    # Deconvolve and return the square sum of the residuals.
    guess_conv = conv_func(guess)

    # Comparison (fluence - dose) / fluence (1 - dose / fluence)
    residual_array = numpy.power((fluence - guess_conv), 2)

    # Evaluation
    return numpy.sqrt(numpy.sum(residual_array))

def parallel_derivative_worker(spparam_a, fluence, inqueue, outqueue, deconv_fn=get_residual):
    # Worker function for deconvolution/comparison.
    convolver = fluence_convolution(FluenceConvolution(param_arrays=spparam_a)
    result = None

    for guess, diff_val, diff_mult in iter(inqueue.get, 'DONE'):
        if result == None:
            result = numpy.zeros(shape=guess.shape)
        for index in range(len(guess.flat)):
            result.flat[index] = diff_mult * deconv_fn(convolver.compute_array, fluence, guess)
            guess.flat[index] -= diff_val
        outqueue.put(result)
inqueue.task_done()
else:
    inqueue.task_done()
class fluence_solver:

    Class for optimizing the deconvolution coefficients.

Attributes:
deconv_order -- A parameter defining the order of the deconvolution.
    Currently defaults to 5th order (DECONV_ORDER=5).
deconv_coefs -- The deconvolution kernel coefficient pairs Ai, Bi as list of
    2-tuples: [ [A0, B0], ..., [An, Bn] ]
fluence_arrays -- A list of the input fluence arrays. Assumed to be matched
    one-to-one with the dose_arrays.
dose_arrays -- A list of the input dose arrays. Assumed to be matched
    one-to-one with the dose_arrays.
array_size -- a 2-tuple containing the X and Y size in pixels.

Functions:
__init__(self, in_deconv_coefs=None, in_deconv_order=None,
    in_fluence_arrays=None, in_dose_arrays=None):
set_deconv_coefs(self, in_deconv_coefs=None, in_deconv_order=DECONV_ORDER):
set_arrays(self, in_fluence_array, in_dose_array):
append_array_pair(self, in_fluence_array, in_dose_array):
compute_quality(self): returns quality from deconvolution.
optimize_coefs(self, )

__name__ = "fluence_solver"

incoming_fluence = None
    """ The fluence we are solving to. """
fluence_guess = None
    """ The fluence we are solving for. """
guess = None

convolver = None

PIXEL_SIZE = 0.04
    """ The size of each pixel in cm """
array_size = None
    """ The size of the arrays being passed in.
    All arrays must be of the same size. """

workers = None

nlp = None
    """ Solver profile """

def __init__(self, params_filename=None, fluence=None):
    """ Initialize the class, setting the parameters if passed.
    Keyword arguments:
in_deconv_coefs -- The deconvolution kernel coefficient pairs Ai, Bi as
        list of 2-tuples: [ [A0, B0], ..., [An, Bn] ]
in_fluence_arrays -- A list of the input fluence arrays. Assumed to be
        matched one-to-one with the dose_arrays.
in_dose_arrays -- A list of the input dose arrays. Assumed to be matched
        one-to-one with the fluence_arrays.
in_deconv_order -- The order of the deconvolution array to set up.
    """
    if params_filename is not None:
        self.convolver = fluence_convolution.FluenceConvolution(params_filename=params_filename)
    else:
        self.convolver = fluence_convolution.FluenceConvolution()

    if fluence is not None:
        self.set_fluence(fluence)
def set_fluence(self, fluence, threshold=0.1):
    """
    Sets the fluence we are trying to match and creates an initial guess.
    """
    self.incoming_fluence = fluence
    avg_fluence = numpy.average(fluence[fluence > threshold])
    self.fluence_guess = avg_fluence

    # Copy the output fluence, resize it to fit the needed input data.
    if fluence.shape[0] != (self.convolver.num_arrs - 1):
        mu = float(fluence.shape[0]) / (self.convolver.num_arrs - 1)
        t = RectBivariateSpline(range(len(fluence)), range(len(fluence)), fluence)
        def f(x, mu):
            return t.ev((mu*x).flatten(),(mu*y).flatten())
        self.fluence_guess = numpy.fromfunction(f, (self.convolver.num_arrs,) +
            (self.convolver.num_arrs,), mu = 0).reshape((self.convolver.num_arrs,-1))

        # The last row and column are the scatter data, for the initial guess,
        # set them to zero.
        self.fluence_guess[-1,:] = 0
        self.fluence_guess[1,-1] = 0

        # Threshold the remaining values to ignore any low contributions.
        self.fluence_guess[fluence < threshold * avg_fluence] = 0
    else:
        self.fluence_guess = numpy.zeros(shape=(
            self.convolver.num_arrs,
            self.convolver.num_arrs))
        X, Y = (fluence > (threshold * avg_fluence)).nonzero()
        self.fluence_guess[X,Y] = fluence[X,Y]

        # Now we will normalize the output. As the weight is a simple
        # multiplication at the end, we can ratio the outputs and renormalize
        # our fluence_guess.
        computed_fluence = self.convolver.compute_array(self.fluence_guess)
        weight_factor = numpy.average(fluence) / numpy.average(computed_fluence)
        self.fluence_guess *= weight_factor

    def compute_quality(self, guess=None, get_residual=None):
        """
        Computes the "quality" of the match for the current parameters.

        Quality of the match will be determined by normalizing each pair to the average value of
        all pixels above a threshold of the maximum pixel value: AVG_THRESHOLD=20% by default.
        The two arrays will then be subtracted and the average deviation percentage in the
        thresholded region (in absolute) will be used as the quality term. This allows for a minimization
        iterative solver to find the optimal value.
        The quality terms for each input fluence/dose pair will be added in quadrature for the
        final term.
        This allows values that are significantly off to have a large effect on the final
        outcome.
        """
        if self.incoming_fluence is None:
            raise ValueError('%s In compute_quality: incoming_fluence not set!' %
                             str(self.__name__))
        if self.fluence_guess is None:
            self.fluence_guess = numpy.asarray(guess, dtype=float).reshape(self.array_shape)
        elif guess is None:
            guess = self.fluence_guess.copy()

        old_guess_shape = guess.shape
guess.shape = self.fluence_guess.shape

quality = get_residual(self.convolver.compute_array,
                        self.incoming_fluence, guess)

guess.shape = old_guess_shape

return quality

def optimize_quality(self):
    """ Adjusts the current values of self.deconv_coefs to optimize the quality function. """
    # We will use scipy.optimize.fmin_l_bfgs_b with func=self.compute_quality
    return optimize.fmin_l_bfgs_b(self.compute_quality, self.fluence_guess,
                                   approx_grad=True, bounds=bounds, iprint=1)

def optimize_quality_nlp(self, solver='algencan', **kwargs):
    from openopt import NLP
    """ Adjusts the current values of self.deconv_coefs to optimize the quality function. """
    # set the bounds for each item
    lower_bound = numpy.zeros(shape=self.fluence_guess.shape)

    # We will use openopt.NLP with func=self.compute_quality
    self.nlp = NLP(self.compute_quality, self.fluence_guess, lb=lower_bound, **kwargs)

    return self.nlp.solve(solver)

def optimize_quality_nlp_ll(self, solver='algencan', *args, **kwargs):
    from openopt import NLP
    """ Adjusts the current values of self.deconv_coefs to optimize the quality function. """
    # set the bounds for each item
    lower_bound = numpy.zeros(shape=self.fluence_guess.shape)

    # We will use openopt.NLP with func=self.compute_quality
    self.nlp = NLP(self.compute_quality, self.fluence_guess, lb=lower_bound, iprint=10,
                   plot=0, show=True, df=self.compute_der_ll, maxIter=1000, **kwargs)

    r = self.nlp.solve(solver)

    if self.workers is not None:
        for worker in self.workers:
            self.task_queue.put('DONE')
        self.task_queue.join()
        self.task_queue.close()
        self.results_queue.close()
        self.workers = None

    return r

def compute_der_ll(self, guess=None, get_residual=get_residual, diffInt=1.5e-8):
    """ Computes the derivative of the match for the current parameters. """
    if self.incoming_fluence is None:
        raise ValueError('(%s) In compute_quality: incoming_fluence not set!' %
                         str(self.__name__))

    if self.fluence_guess is None:
        raise ValueError('compute_der_ll called before setting everything up.')
# Now lets loop through the input datasets.

```python
quality = get_residual(self.convolver.compute_array,
                        self.incoming_fluence, self.fluence_guess)

stencil = [ ( 2*diffInt, 1),
            ( diffInt, 8),
            (-diffInt, -8),
            (-2*diffInt, 1)]

if not self.workers: # We don't have a pool set up yet.
    self.previous_result = numpy.zeros(shape=self.fluence_guess.shape)
    self.workers = []
    NCPU = min(cpu_count(), len(stencil))
    self.task_queue = JoinableQueue()
    self.results_queue = Queue()
    for i in range(NCPU):
        print("Starting Worker %i" % i)
        self.workers.append(Process(target=parallel_derivative_worker,
                                      args=((self.convolver.X_array, self.convolver.Y_array),
                                            self.incoming_fluence, self.task_queue, self.results_queue)))
    self.workers[i].start()

for diff_val, diff_mult in stencil:
    self.task_queue.put((self.fluence_guess, diff_val, diff_mult))

self.task_queue.join()

result = numpy.zeros(shape=self.fluence_guess.shape)

for i in range(len(stencil)):
    result += self.results_queue.get()

result /= (12*diffInt)
return result.flatten()
```

```py
def main(args=None):

    from optparse import OptionParser

    usage = "Usage: %prog [options] input_dataset [outfile]"
    parser = OptionParser(usage=usage)
    parser.add_option("-d", "--display", action="store_true", default=False,
                      help="Display plots at the end.")
    parser.add_option("-b", "--benchmark", action="store_true", default=False,
                      help="Display benchmarking information.")
    parser.add_option("-o", "--output", action="store_true", default=False,
                      help="Output to out-file.")
    parser.add_option("-n", "--no-output", action="store_false", dest="output",
                      help="Suppress writing output.")
    parser.add_option("-f", "--out-file", dest="fn_writefile",
                      help="File to store result in.")
    parser.add_option("-p", "--param-space-file", "--param-space-file", dest="fn_apparam",
                      help="File to store result in.")
    parser.add_option("-s", "--solver", dest="ralg",
                      help="File to store result in.")

    (options, args) = parser.parse_args()
    if len(args) < 1:
        parser.error("We must have the input fluence dataset.")
    return()

fn_readfile = args[0]
import cPickle

try:
    f_readfile = openfile(fn_readfile, 'rb')
    # We have the file open and an interface ready to read from it.
    # Now lets loop through the input datasets.
```
try:
    # Set up the dictionary to hold our set.
    data_sets = []
    # Loop
    while 1:
        try:
            # We want to catch only EOFError
            # (means we have finished reading from the file)
            (x_pts, y_pts, fluence) = cPickle.load(f_readfile)
            data_sets.append(fluence)
            print('Found fluence. Adding it to job.')
        except EOFError:
            # EOFError should mean we are at the end of the file.
            break
        except cPickle.UnpicklingError as e:
            sys.stderr.write("Error unpickling data, aborting. (Error: %s)\n" % e.strerror)
            return(1)
        except IOError as e:
            sys.stderr.write("Could not open %s, it may not exist. (Error: %s)\n" % (fn_readfile, e.strerror))
            return(1)

    quality = []
    for fluence in data_sets:
        optimizer = fluence_solver(params_filename=options.fn_spparam, fluence=fluence)
        quality.append(optimizer.compute_quality())

    if options.display:
        for val in quality:
            print 'Initial Quality is: %s' % str(val)

        opt_res = optimizer.optimize_quality_nlp_ll(solver=options.solver)

        if options.display:
            print 'Array is: ', numpy.array(opt_res.xf).reshape(-1,2)

        if options.output and options.outfile is not None:
            f_outfile = open(options.outfile, 'wb')
            cPickle.dump(opt_res.xf, f_outfile, cPickle.HIGHEST_PROTOCOL)

    return 0

if __name__ == "__main__":
    import sys
    status = main()
    sys.exit(status)
# Evaluation

```python
def __init__(self, in_deconv_coefs=None, in_deconv_order=None, in_fluence_arrays=None, in_dose_arrays=None):
    if in_deconv_coefs is not None:
        self.deconv_coefs = in_deconv_coefs
    if in_deconv_order is not None:
        self.deconv_order = in_deconv_order
    if in_fluence_arrays is not None:
        self.fluence_arrays = in_fluence_arrays
    if in_dose_arrays is not None:
        self.dose_arrays = in_dose_arrays
    if params_filename is not None:
        #incoming_fluence = None
        fluence_guess = None
        guess = None
        convolver = None
        PIXEL_SIZE = 0.04
        array_size = None
        def __init__ (self, params_filename=None, fluence=None, **kwargs):
            Initialize the class, setting the parameters if passed.

            Keyword arguments:
            in_deconv_coefs -- The deconvolution kernel coefficient pairs A\textsubscript{i}, B\textsubscript{i} as list of 2-tuples: [ [A\textsubscript{0}, B\textsubscript{0}], ..., [A\textsubscript{n}, B\textsubscript{n}] ]
            in_fluence_arrays -- A list of the input fluence arrays. Assumed to be matched one-to-one with the dose_arrays.
            in_dose_arrays -- A list of the input dose arrays. Assumed to be matched one-to-one with the dose_arrays.
            in_deconv_order -- The order of the devonvolution array to set up.

            if params_filename is not None:
                kwarg specialised_filename = kwarg specialised_filename
                self.convolver = ParallelFluenceConvolution(**kwarg specialised)
                if fluence is not None:
                    self.set_fluence(fluence)
```
sets the fluence we are trying to match and creates an initial guess.

```
def normalize_initial_guess(self, threshold=0.1):
    """ Compute an initial guess and normalize it for the input fluence. """
    # Copy the output fluence, resize it to fit the needed input data.
    fluence = self.incoming_fluence
    avg_fluence = numpy.average(fluence[fluence > threshold] *
                               numpy.max(fluence)))
    if (self.incoming_fluence.shape != self.convolver.coefs_shape):
        # Array shapes not the same, make an interpolated fluence to fit the
        # dataset we are working with.
        # Smooth first
        from scipy.ndimage.filters import gaussian_filter
        from scipy.interpolate import RectBivariateSpline
        def f(x, y, mu):
            return t.ev((mu*x).flatten(), (mu*y).flatten())
        self.fluence_guess = numpy.fromfunction(f,
                                                self.convolver.coefs_shape,
                                                (mu, mu)).reshape(self.convolver.coefs_shape)
        if self.incoming_fluence.shape != self.convolver.calc_fluence.shape:
            self.incoming_fluence = numpy.fromfunction(f,
                                                        (self.convolver.calc_fluence_shape,
                                                         self.convolver.calc_fluence_shape),
                                                        (mu, mu)).reshape((self.convolver.calc_fluence_shape, -1))

        # The last row and column are the scatter data, for the initial guess,
        # set them to zero.
        self.fluence_guess[-1, :] = 0
        self.fluence_guess[:, -1] = 0

        # Threshold the remaining values to ignore any low contributions.
        self.fluence_guess[self.fluence_guess < threshold * avg_fluence] = 0
    else:
        fluence_guess = numpy.zeros(shape=convolver.coefs_shape)
        X, Y = (fluence > (threshold * avg_fluence)).nonzero()
        fluence_guess[X, Y] = fluence[X, Y]

    # Now we will normalize the output. As the weight is a simple
    # multiplication at the end, we can ratio the outputs and renormalize
    # our fluence_guess.
    threshold = numpy.average(self.incoming_fluence) * threshold
    computed_fluence = self.convolver.compute_array(self.fluence_guess).A

    thresh_fl = (self.incoming_fluence > threshold).nonzero()
    fl_avg = numpy.average(self.incoming_fluence[thresh_fl])
    cfl_avg = numpy.average(computed_fluence[thresh_fl])
    weight_factor = fl_avg / cfl_avg
    self.fluence_guess *= weight_factor

    # Modify the child data so we don't have to run a spurious convolution.
    self.convolver.prev_coefs.flat[:] *= weight_factor
    self.convolver.calc_fluence.data[:] *= weight_factor
    self.convolver.flMpcfl.flat[:] = self.incoming_fluence - self.convolver.calc_fluence
```

```
def compute_quality(self, guess=None, get_residual_flMpcfl=get_residual_flMpcfl):
    """
    Computes the "quality" of the match for the current parameters.
    Quality of the match will be determined by normalizing each pair to the average value of
    all pixels above a threshold of the maximum pixel value: AVG_THRESHOLD=20% by default.
```
The two arrays will then be subtracted and the average deviation percentage in the thresholded region (in absolute) will be used as the quality term. This allows for a minimization iterative solver to find the optimal value.

The quality terms for each input fluence/dose pair will be added in quadrature for the final term. This allows values that are significantly off to have a large effect on the final outcome.

```python
if self.incoming_fluence is None:
    raise ValueError('(%s) In compute_quality: incoming_fluence not set!' % str(self.__name__))

if self.fluence_guess is None:
    self.fluence_guess = numpy.asarray(guess, dtype=float).reshape(self.array_shape).copy()
elif guess is None:
    guess = self.fluence_guess.copy()
else:
    self.fluence_guess.flat[:] = guess

self.convolver.compute_array(guess)
quality = get_residual_flMpcfl(self.convolver.flMpcfl)
return quality

def term_callback(self, p):
    self.convolver.save_partial()
    if exists('/tmp/terminate_solver'):
        return (80, '/tmp/terminate_solver found, ending run.')
    return False

def compute_derivative(self, guess=None):
    # We can ignore guess as we will have had it, just call compute on out convolver.
    prev_res_sq = self.compute_quality(guess)**2
    return self.convolver.compute_derivative(dx=self.nlp.diffInt, prev_res_sq=prev_res_sq)

def optimize_quality(self):
    """ Adjusts the current values of self.deconv_coefs to optimize the quality function. """

    # We will use scipy.optimize.fmin_l_bfgs_b with func=self.compute_quality

    return optimize.fmin_l_bfgs_b(self.compute_quality, self.fluence_guess, approx_grad=True, bounds=bounds, iprint=1)

def master(options, args):

    fn_readfile = args[0]

    import cPickle
    from util import openfile

    try:
        f_readfile = openfile(fn_readfile, 'rb')
    except IOError as e:
        sys.stderr.write("Could not open %s, it may not exist. (Error: %s)\n" % (fn_readfile, e strerror))
        return(1)

    x_pts, y_pts, fluence = cPickle.load(f_readfile)

    print('Found fluence. Adding it to job.')

    except cPickle.UnpicklingError as e:
        sys.stderr.write("Error unpickling data, aborting. (Error: %s)\n" % e strerror)
        return(1)

    except IOError as e:
        sys.stderr.write("Could not open %s, it may not exist. (Error: %s)\n" % (fn_readfile, e strerror))
        return(1)
```

215
kwargs = dict()
if options.master_port:
    kwargs['master_address'] = ('', int(options.master_port))
else:
    kwargs['master_address'] = ('', 50000)
kws['nodes'] = ['localhost', 'blade2']
kws['params_filename'] = options.fp_apparam
kws['fluence'] = fluence
kws['fn_writefile'] = options.fn_writefile
if options.coefs_shape:
    kwargs['coefs_shape'] = (int(options.coefs_shape), int(options.coefs_shape))

opt_res = None
print("%9.2f: SOLVER: launching optimizer" % time())
sys.stdout.flush()
optimizer = ll_fluence_solver(**kws)
# Use with to launch the workers.
with optimizer.convolver:
    print("%9.2f: SOLVER: getting initial guess" % time())
optimizer.normalize_initial_guess()
    if options.output and options.fn_writefile is not None:
        with open(options.fn_writefile + '.init', 'wb') as f_outfile:
            numpy.save(f_outfile, coefs=optimizer.fluence_guess, calc_fluence=optimizer.convolver.calc_fluence.A)
        print("%9.2f: SOLVER: initial guess quality: %s" % (time(), optimizer.compute_quality()))
sys.stdout.flush()
opt_res = optimizer.optimize_quality_nlp(solver=options.solver, callback=optimizer.term_callback, iprint=1, maxIter=1000, df=optimizer.compute_derivative, debug=True)

quality = optimizer.compute_quality(opt_res.xf)

if options.display:
    for val in quality:
        print('Initial Quality is: %s' % str(val))

if options.display:
    print('Array is: ', numpy.array(opt_res.xf).reshape(-1, 2))

if options.output and options.fn_writefile is not None:
    with open(options.fn_writefile, 'wb') as f_outfile:
        numpy.save(f_outfile, coefs=opt_res.xf.reshape(optimizer.convolver.coefs_shape), calc_fluence=optimizer.convolver.calc_fluence.A)
        #cPickle.dump((opt_res.xf,optimizer.convolver.calc_fluence.A), f_outfile, cPickle.HIGHEST_PROTOCOL)

print("%9.2f: SOLVER: Done, exiting." % time())
#from IPython.Shell import IPShellEmbed
#IPShellEmbed('')()
return 0

if __name__ == "__main__":
    from os import getenv, chdir
    import sys
    # Find our id and branch based on that.
    from optparse import OptionParser

    usage = "Usage: %prog [options] input_dataset [outfile]"
    parser = OptionParser(usage=usage)
# it well into user space and fairly safe.

# Set the port based on the job number plus 50000. This will keep
# it well into user space and fairly safe.

if not options.master_port:
    job_id = int(getenv('PBS_JOBID').split('.')[0])
    options.master_port = 50000 + (job_id % 15535)

if getenv('PBS_NODEFILE') != None:
    options.node = False
else:
    options.node = True

if options.node:
    if not options.master or not options.master_port:
        parser.error("We are a node but don't have a master. Aborting.")
        sys.exit(1)
    status = node(options, args)
else:
    status = master(options, args)

if logfile:
    sys.stdout = sys.__stdout__
    sys.stderr = sys.__stderr__
    logfile.close()

sys.exit(status)
Appendix G

Ancilary Utility Functions

The following programs were created to assist in viewing, manipulating, and processing the data used in this dissertation.

G.1 rtp2mlc\script.sh

```bash
#!/bin/bash

BEAM_TEMPLATE=`cat ./templates/beam.template`
CP_TEMPLATE=`cat ./templates/cp.template`

MLC_SCALE=".5102"
IMPAC_RTP_FILE="${1}"

if [[ ! -e ${IMPAC_RTP_FILE} ]] || ( echo "Error, File ${IMPAC_RTP_FILE} not found." >&2 && exit 1 );
then
    IGNORE_CP=true;
fi

OUTPUT_FILE_ROOT="${IMPAC_RTP_FILE%%.[Rr][Tt][Pp]}"

# NOTE: In order to use templates, we will have to substitute the following variables
# in to each template. In order to do this we must do Substitutions such as:
#  EXAMPLE_BEAM=${BEAM_TEMPLATE/'${BEAMNAME}'/${BEAMNAME}}
#  EXAMPLE_BEAM=${EXAMPLE_BEAM/'${NUMBER_OF_CP}'/${NUMBER_OF_CP}}
#  etc.

BEAM_LIST="";  # Pinnacle Formatted full list of beams generated from $BEAM_TEMPLATE

# Variable Subs stored in the $TRIAL_TEMPLATE File
CURRENT_CP_ID="";
GANTRY="";  # Gantry for this CP
COUCH="";  # Couch for this CP
COLLIMATOR="";  # Coll. for this CP
LEFT_JAW="";  # +X2 JAW (+ is +)
RIGHT_JAW="";  # -X1 JAW (- is +)
TOP_JAW="";  # -Y1 JAW (- is +)
```

218
BOTTOM_JAW=""; # +Y2 JAW (+ is +)
CONTROL_PT_WEIGHT=""; # Relative weight of this CP
MLC_LEAF_POSITIONS=""; # List of MLC Positions starting from Y1-most Pair as X1,-X2

MLC_START=32;

# Variable Subs Stored in the $BEAM_TEMPLATE File
BEAMNAME="";  # Name of Beam
NUMBER_OF_CP=0; # Total # of CP for this beam (Pinnacle Number)
SSD=""; # SSD for this Beam
BEAM_WEIGHT=""; # Relative Beam Weight in %. (e.g. 25% => 25)
CONTROL_POINT_LIST=""; # Pinnacle formatted list of CP, generated from SCP_TEMPLATE

BEAM_ARRAY=(); # This holds the full beam definition in Pinnacle Format for each beam.
BEAM_MU=(); # This holds the number of MUs for each beam.
SSD_ARRAY=(); # SSD for each beam

THIS_BEAM=""; # The current beam being added to the beam array
THIS_CP=""; # The current control point list
CONTROL_POINT_ARRAY=(); # CONTROL_POINT_LIST for each Beam.

THIS_CP_PCT=0; # Placeholder for each beam for the current relative MU %
LAST_CP_PCT=0; # Placeholder for each beam for the last relative MU %
NUMBER_OF_BEAMS=0;
TOTAL_PLAN_MU=0;
sNEG()
{
    echo "${1//"/} * -1" | bc
}
sPOS()
{
    echo "${1//"/}"
}
s2MLC()
{
    echo "${1//"/} * ${MLC_SCALE}" | bc
}

OLD_IFS=$IFS;
IFS=$'
';
for line in $(cat $IMPAC_RTP_FILE)
do
IFS="";
    CURRENT_INPUT=$line;
case $CURRENT_INPUT[0] in
        "PLAN_DEF"
            ;;
        "RX_DEF"
            ;;
        "FIELD_DEF"
            CUR_BEAM_INDEX=NUMBER_OF_BEAMS;
            (( NUMBER_OF_BEAMS++ ));
            echo "FIELD DEF for beam $CUR_BEAM_INDEX" >$2
            BEAM_NAME[$CUR_BEAM_INDEX]=""$CURRENT_INPUT[3]"/""
            BEAM_MU[$CUR_BEAM_INDEX]=""$CURRENT_INPUT[6]"/""
            SSD_ARRAY[$CUR_BEAM_INDEX]=""$CURRENT_INPUT[15]"/""
            GANTRY[$CUR_BEAM_INDEX]=""$CURRENT_INPUT[16]"/""
            COLLIMATOR[$CUR_BEAM_INDEX]=""$CURRENT_INPUT[17]"/""
            COUCH[$CUR_BEAM_INDEX]=""$CURRENT_INPUT[29]"/""
            ENERGY[$CUR_BEAM_INDEX]=""$CURRENT_INPUT[11]"/"
    ;;
    esac
done

229
TOTAL_PLAN_MU=$(echo "${BEAM_MU[$CUR_BEAM_INDEX]} + $TOTAL_PLAN_MU" | bc);

echo "TOTAL_PLAN_MU: $TOTAL_PLAN_MU" >&2

if [[ "$ERSION_CP" != "$NEW_CP" ]]; then
LAST_CP_PCT=$THIS_CP_PCT;
THIS_CP_PCT=1($CURRENT_INPUT[7]);

echo "This CP: $THIS_CP_PCT; Last CP: $LAST_CP_PCT" >&2

if [[ "$THIS_CP_PCT" == "0.000000" ]];
then
  # First CP in a beam. Reset the number of CP to 0 and 
  # Prepare to add the next CP to the new array.
  echo "New Control Point on BEAM $CUR_BEAM_INDEX" >&2
  NUMBER_OF_CP=0;
elif [[ "$THIS_CP_PCT" != "$LAST_CP_PCT" ]];
then
  # The second time we have seen this CP, this is the one we will 
  # add.

  # Note: This CP Weight is INDEX(I) for DYNVMLC

  echo "Adding New Control Point No: $NUMBER_OF_CP" >&2
  NUM_LEAVES=$(sPOS ${CURRENT_INPUT[3]});
  # THIS_CP=${CP_TEMPLATE//'${CURRENT_CP_ID}'/$NUMBER_OF_CP};
  # THIS_CP=${THIS_CP//'${GANTRY}'/${GANTRY[$CUR_BEAM_INDEX]});
  # THIS_CP=${THIS_CP//'${COUCH}'/${COUCH[$CUR_BEAM_INDEX]});
  # THIS_CP=${THIS_CP//'${COLLIMATOR}'/${COLLIMATOR[$CUR_BEAM_INDEX]});
  # THIS_CP=${THIS_CP//'${RIGHT_JAW}'/$(sNEG ${CURRENT_INPUT[19]})};
  # THIS_CP=${THIS_CP//'${LEFT_JAW}'/$(sPOS ${CURRENT_INPUT[20]})};
  # THIS_CP=${THIS_CP//'${TOP_JAW}'/$(sNEG ${CURRENT_INPUT[23]})};
  # THIS_CP=${THIS_CP//'${BOTTOM_JAW}'/$(sPOS ${CURRENT_INPUT[24]})};
  for ((i=MLC_START; i < MLC_START+NUM_LEAVES; i++))
  do
    if [[ -z "${THIS_CP_MLC_POS}" ]] 
      then
        THIS_CP_MLC_POS=${s2MLC ${CURRENT_INPUT[$i]}}, "${s2MLC ${CURRENT_INPUT[1$i]}}, 1"
    else
      THIS_CP_MLC_POS=${THIS_CP_MLC_POS}\n"${s2MLC ${CURRENT_INPUT[$i]}}, "${s2MLC ${CURRENT_INPUT[1$i]}}, 1"
  done
  THIS_CP=${CP_TEMPLATE//'${MLC_LEAF_POSITIONS}'/${THIS_CP_MLC_POS};
  THIS_CP_WEIGHT=$(sPOS ${THIS_CP_PCT});
  THIS_CP=${THIS_CP//'${CONTROL_PT_WEIGHT}'/${THIS_CP_WEIGHT});

  if [[ "${CONTROL_POINT_ARRAY[$CUR_BEAM_INDEX]}" != "$CUR_BEAM_INDEX" ]];
  then
    CONTROL_POINT_ARRAY[$CUR_BEAM_INDEX]=${THIS_CP};
  else
    CONTROL_POINT_ARRAY[$CUR_BEAM_INDEX]=""${CONTROL_POINT_ARRAY[$CUR_BEAM_INDEX]}\n"${THIS_CP};
  fi

  THIS_CP="";
  THIS_CP_MLC_POS="";
  (( NUMBER_OF_CP++ ));
fi

if [[ "$THIS_CP_PCT" == "1.000000" ]];
then  # Last CP in a beam. We should add the beam to the array.
    # TODO: Add beam to the beam list array.
    echo "Adding beam ${BEAM_NAME[$CUR_BEAM_INDEX]} to list" >&2

    THIS_BEAM=${BEAM_TEMPLATE//'${BEAMNAME}'+'/'+BEAM_NAME[$CUR_BEAM_INDEX]};
    THIS_BEAM=${THIS_BEAM//'${NUMBER_OF_CP}'+'/'+NUMBER_OF_CP} ;
    THIS_BEAM=${THIS_BEAM//'${CONTROL_POINT_LIST}'+'/'+CONTROL_POINT_ARRAY[$CUR_BEAM_INDEX]} ;
    CONTROL_POINT_ARRAY[$CUR_BEAM_INDEX]="";
    BEAM_ARRAY[$CUR_BEAM_INDEX]=${THIS_BEAM} ;
    THIS_BEAM="" ;
fi
fi
;;
esac
done

# We've finished populating the list of beams. Now we just have to set the weights and put it in all in the template.

for (( i=0; i < NUMBER_OF_BEAMS; i++ )); do
    OUTPUT_FILE=${OUTPUT_FILE_ROOT}_${BEAM_NAME[i]}_${ENERGY[i]}MV_${BEAM_MU[i]} ;
    echo "$BEAM_ARRAY[i]" >> $OUTPUT_FILE ;
    BEAM_ARRAY[i]="" ;
done

G.2 rtp2mlc/templates/beam.template

${BEAMNAME}
${NUMBER_OF_CP}
${CONTROL_POINT_LIST}

G.3 rtp2mlc/templates/cp.template

${CONTROL_PT_WEIGHT}
${MLC_LEAF_POSITIONS}

G.4 utils.py

#!/usr/bin/python
import gzip
from numpy import dtype, asarray, hstack
from scipy import sparse
from functools import wraps

# A simple decorator to silently return on a Keyboard Interrupt
```python
def ReturnOnKeyboardInterrupt(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        try:
            return func(*args, **kwargs)
        except KeyboardInterrupt:
            pass
        return wrapper

# A quick function to generate 4 digit precis. SI suffixes
def si(value):
    value_str = '%4.4g' % value
    if 'e' in value_str:
        SI = dict((3:'k', 6:'M', 9:'G', 12:'T', 15:'P'))
        mod = int(value_str.split('e')[1][:-1]) % 3
        exp = SI[int(value_str.split('e')[1][:-1]) - mod]
        o_str = '%s %s' % ('%.1f' % (float(value_str.split('e')[0]) * 10 ** mod), exp)
    else:
        o_str = '%s' % value_str
    return o_str.rjust(7)

def openfile(fn_infile, mode='rb'):
    f_infile = open(fn_infile, mode)
    if (f_infile.read(2) == '\x1f\x8b'):
        f_infile = gzip.GzipFile(fileobj=f_infile)
        f_infile.rewind()
    try:
        f_infile.read(5)
    except IOError:
        f_infile = f_infile.fileobj
        f_infile.seek(0)
    else:
        f_infile.seek(0)
    return f_infile

# This def lets us read in only small chunks of the file as we need it, rather than loading
# the whole phase space into memory (could be gigs...)
def buf_data_from_file(infile, byte_array):
    while True:
        bytes_read = infile.readinto(byte_array)
        if bytes_read == 0:
            break
        yield bytes_read

# Similar to above, but to read to a ctype array (an extra copy...)
def buf_data_from_file_to_arr(infile, byte_array, ctype_array):
    while True:
        bytes_read = infile.readinto(byte_array)
        #print('br: %s % bytes_read)
        ctype_array[:bytes_read] = byte_array[:bytes_read]
        if bytes_read == 0:
            break
        yield bytes_read

#def sp_coo_append(coo_input, other, max_size=10485760):
#  def sp_coo_append(coo_input, other, max_size=4194304):
#    # Max size defaults to a coo_matrix of about 100MB
#    if isinstance(other, tuple):
#        # Assume form i,j,data
#        innz=len(coo_input.data)
#        coo_input.col.resize(innz+1)
#        coo_input.row.resize(innz+1)
#        coo_input.data.resize(innz+1)
#        coo_input.col[-1] = other[0]
#        coo_input.row[-1] = other[1]
#        coo_input.data[-1] = other[2]
#        return coo_input
```
if coo_input.dtype != other.dtype:
    coo_other = sparse.coo_matrix(other, copy=False, dtype=coo_input.dtype)
else:
    coo_other = sparse.coo_matrix(other, copy=False)

if coo_input.shape != coo_other.shape:
    new_coo = sparse.coo_matrix(coo_input.todense() + coo_other.todense())
    coo_input.shape = new_coo.shape
    coo_input.col = new_coo.col
    coo_input.row = new_coo.row
    coo_input.data = new_coo.data
else:
    if coo_input.nnz > max_size:
        # Recondense, convert to csr then back to coo.
        tmp = coo_input.tocsr().tocoo()
        coo_input.row = tmp.row
        coo_input.col = tmp.col
        coo_input.data = tmp.data
        innz = len(coo_input.data)
        onnz = len(coo_other.data)
        # Expand to fit new data, do this first.
        coo_input.col.resize(innz + onnz, refcheck=False)
        coo_input.row.resize(innz + onnz, refcheck=False)
        coo_input.data.resize(innz + onnz, refcheck=False)
        # Put in new col & row, then data.
        coo_input.col[innz:] = coo_other.col
        coo_input.row[innz:] = coo_other.row
        coo_input.data[innz:] = coo_other.data

return coo_input

def sp_coo_inplace_assign(coo_input, coo_other):
    if coo_input.dtype != other.dtype:
        other = asarray(other, dtype=coo_input.dtype)

    coo_other = sparse.coo_matrix(other, copy=False)

    onnz = len(coo_other.data)
    # Expand to fit new data, do this first.
    coo_input.col.resize(onnz, refcheck=False)
    coo_input.row.resize(onnz, refcheck=False)
    coo_input.data.resize(onnz, refcheck=False)
    # Put in new col & row, then data.
    coo_input.col[:] = coo_other.col
    coo_input.row[:] = coo_other.row
    coo_input.data[:] = coo_other.data

    return coo_input

# MODE0 PHSP (no Z-LAST)
MODE0_dt = dtype({
    'names': ['LATCH', 'E', 'X', 'Y', 'U', 'V', 'WT'],
    'formats': ['u1', 'f4', 'f4', 'f4', 'f4', 'f4', 'f4']
})

MODE2_dt = dtype({
    'names': ['LATCH', 'E', 'X', 'Y', 'U', 'V', 'WT', 'ZL'],
    'formats': ['u1', 'f4', 'f4', 'f4', 'f4', 'f4', 'f4', 'f4']
})

G.5 disp_binned.py

#!/usr/bin/python
import os
import sys
import numpy
import matplotlib.pyplot as plt
import cPickle
import gzip
from time import time

if __name__ == '__main__':

display=True
benchmark=True
output=True
f_readfile = None
f_writefile = None

if len(sys.argv) < 2:
    sys.stderr.write( "Usage: %s input_hist
" % sys.argv[0])
    sys.exit(1)

fn_readfile = sys.argv[1]

# Param 2 is output filename, or we assign it based on input filename.
fn_readfile_basename = os.path.basename(fn_readfile)

# Strip off ".gz" from the basename, if it is there.
if fn_readfile_basename.endswith(".gz",-3):
    fn_readfile_basename = fn_readfile_basename.rpartition(".")[0]

# Try to open the input file for reading and abort if we cannot.
# Also checks to see if the file looks like a gzip file.  If it seems like it is
# we load it as such, and read the first 5 characters to test this.  If it is a
# false positive, the data was corrupt anyway (first 2 characters should be normal text)
try:
    f_readfile = open(fn_readfile,'rb')
    if (f_readfile.read(2) == '\x1f\x8b'):
        f_readfile = gzip.GzipFile(fileobj = f_readfile)
        f_readfile.rewind()
        f_readfile.read(5)
        f_readfile.rewind()
    else:
        f_readfile.seek(0)
except IOError as e:
    sys.stderr.write("Could not open %s, it may not exist. (Error: %s)\n" % (fn_readfile, e.strerror ) )
    sys.exit(1)

try:
    (X_Points,Y_Points,Dose_Array,Dose_Error_Array) = cPickle.load(f_readfile)
except cPickle.UnpicklingError as e:
    sys.stderr.write("First line has some formatting error, aborting. (Error: %s)\n" % e.strerror)
    sys.exit(1)

f_readfile.close() # Close file

extents=[min(X_Points[1:-1]),max(X_Points[1:-1]),min(Y_Points[1:-1]),max(Y_Points[1:-1])]
# Ignore dose errors for values less than 20% of the max dose value
Dose_Error_Array[Dose_Array < (0.2* max(Dose_Array.flatten()))] = 0
print "Average error Value in region > 20% Max dose: %f" % numpy.average(Dose_Error_Array[Dose_Error_Array <> 0])
plt.figure(1)
plt.imshow(Dose_Array[1:-1,1:-1], extent=extents, interpolation='nearest')
plt.colorbar()
plt.figure(2)
plt.imshow(Dose_Error_Array[1:-1,1:-1], extent=extents, interpolation='nearest')
plt.colorbar()
plt.show()
G.6 disp_binned_dcparam.py

```python
#!/usr/bin/python
import os
import sys
import numpy
import matplotlib.pyplot as plt
import cPickle
import gzip
from time import time
from scipy import sparse
if __name__ == '__main__':
    display=True
    benchmark=True
    output=True
    f_readfile = None
    f_writefile = None
    if len(sys.argv) < 2:
        sys.stderr.write( "Usage: %s input_hist\n" % sys.argv[0])
        sys.exit(1)
    fn_readfile = sys.argv[1]
    # Param 2 is output filename, or we assign it based on input filename.
    fn_readfile_basename = os.path.basename(fn_readfile)
    # Strip off ".gz" from the basename, if it is there.
    if fn_readfile_basename.endswith(".gz",-3):
        fn_readfile_basename = fn_readfile_basename.rpartition(".")[-3]
    # Try to open the input file for reading and abort if we cannot.
    # Also checks to see if the file looks like a gzip file.  If it seems like it is
    # we load it as such, and read the first 5 characters to test this.  If it is a
    # false positive, the data was corrupt anyway (first 2 characters should be normal text)
    try:
        f_readfile = open(fn_readfile,'rb')
        if f_readfile.read(2) == '\x1f\x8b':
            f_readfile = gzip.GzipFile(fileobj = f_readfile)
            f_readfile.rewind()
            f_readfile.read(5)
            f_readfile.rewind()
        else:
            f_readfile.seek(0)
    except IOError as e:
        sys.stderr.write("Could not open %s, it may not exist. (Error: %s)\n" % ( fn_readfile, e.strerror ) )
        sys.exit(1)
    try:
        (X_Points,Y_Points,X_array,Y_array) = cPickle.load(f_readfile)
    except cPickle.UnpicklingError as e:
        sys.stderr.write("First line has some formatting error, aborting. (Error: %s)\n" % e.strerror)
        sys.exit(1)
    f_readfile.close()  # Close file
    extents=[min(X_Points),max(X_Points),min(Y_Points),max(Y_Points)]
    for i,Xa in enumerate(X_array):
```
G.7 disp_binned_fl.py

#!/usr/bin/python

import os
import sys
import numpy
import matplotlib.pyplot as plt
import cPickle
import gzip
from time import time

if __name__ == '__main__':
    display=True
    benchmark=True
    output=True

    f_readfile = None
    f_writefile = None

    if len(sys.argv) < 2:
        sys.stderr.write("Usage: %s input_hist
" % sys.argv[0])
        sys.exit(1)

    fn_readfile = sys.argv[1]

    # Param 2 is output filename, or we assign it based on input filename.
    fn_readfile_basename = os.path.basename(fn_readfile)
    # Strip off ".gz" from the basename, if it is there.
    if fn_readfile basename.endswith(".gz",-3):
        fn_readfile basename = fn_readfile basename rpartition("")

    # Try to open the input file for reading and abort if we cannot.
    # Also checks to see if the file looks like a gzip file. If it seems like it is
    # we load it as such, and read the first 5 characters to test this. If it is a
    # false positive, the data was corrupt anyway (first 2 characters should be normal text)
    try:
        f_readfile = open(fn_readfile, 'rb')
        if (f_readfile.read(2) == '\x01\x8b'):
            f_readfile = gzip.GzipFile(fileobj = f_readfile)
            f_readfile.rewind()
            f_readfile.read(5)
            f_readfile.rewind()
        else:
            f_readfile.seek(0)
    except IOError as e:
        sys.stderr.write("Could not open %s, it may not exist. (Error: %s)\n" % (fn_readfile, e.strerror ))
        sys.exit(1)

    try:
        (X_Points, Y_Points, Dose_Array) = cPickle.load(f_readfile)
    except cPickle.UnpicklingError as e:
sys.stderr.write("First line has some formatting error, aborting. (Error: %s)\n" % e.strerror)
sys.exit(1)

f_readfile.close() # Close file

extents=min(X_Points),max(X_Points),min(Y_Points),max(Y_Points))
# Ignore dose_errors for values less than 20% of the max dose value

plt.figure()
plt.title('Dose Array')
plt.imshow(Dose_Array[1:-1,1:-1], extent=extents, interpolation='nearest')
plt.colorbar()
plt.show()

G.8 combine_phsp_using_beamdp.sh

#!/bin/bash

SIZE_TOTAL=0
END_SIZE=0
ENTRY_SIZE=32 # Each entry in the phasename file is 32 bytes for MODE2 and MODE3, and 28 for MODE0 and MODE1;

[ -z "which beamdp" ] && exit 1
BEAMDP=`which beamdp`

if [[ $# -eq 1 && -e "$(1)" ]]
then
echo "Working on file: $(1)"
FILE="$(1).egsphsp?"
FILE="$(FILE%w?)"
PHSP_N="$(#*.egsphsp)"
WORK1_FILE="${FILE}_w1.egsphsp${PHSP_N}"

OUTPUT_FILE="${FILE}.egsphsp${PHSP_N}"

if [[ ! -e "${WORK1_FILE}" ]]
then
echo "Could not find first file in sequence ($WORK1_FILE), giving up." >&2
exit 2
fi # [[ ! -e "$(WORK1_FILE)" ]]

if [[ -e "${OUTPUT_FILE}" ]]
then
echo "Output file ($OUTPUT_FILE) exists, aborting." >&2
exit 3
fi # [[ -e "$(OUTPUT_FILE)" ]]

for work_file in *.egsphsp${PHSP_N};
do
if [[ ! -e "$(OUTPUT_FILE)" ]]
then
cp "$(work_file)" "$(OUTPUT_FILE)"
touch "$(OUTPUT_FILE)"
MODE=`od -j4 -N1 -a -An $(OUTPUT_FILE)`
ENTRY_SIZE=$(MODE*28) # MODE0 is 28, MODE2 is 32 (28 + 2*2).

if [[ stat -c% -s $(work_file) -gt ENTRY_SIZE ]]
then
[BEAMDP] > /dev/null standard_error.log
fi

if [[ ! $# -eq 0 ]]
then
echo "Something went wrong...aborting."
```bash
#!/bin/bash

SIZE_TOTAL=0
END_SIZE=0
HEADER_SIZE=32  # The phasespace file header is 32 bytes on this system.
ENTRY_SIZE=32   # Each entry in the phasespace file is 32 bytes for MODE2 and MODE3, and 28 for MODE0 and MODE1;

if [[ -z "$BEAMDP" ]] && exit 1
BEAMDP=`which beamdp`

if [[ $# -eq 1 && -e "$1" ]] then
  echo "Working on file: $1"
  FILE="$1.egsphsp"
  FILEW="$FILE.w?"  # Existing file
  PHSP=$FILE.w?
  WORK1_FILE="$FILE_w1.egsphsp"  # Existing file
  OUTPUT_FILE="$FILE.egsphsp"  # Existing file
fi # [[ ! -e "$1" ]] then
  echo "Could not find first file in sequence ($1), giving up." >&2
  exit 2
fi # [[ ! -e "$1" ]]}
```
if [[ -e "${OUTPUT_FILE}" ]]
then
    echo "Output file ($OUTPUT_FILE) exists, aborting." >&2
    exit 3
fi # [[ -e ${OUTPUT_FILE} ]]

for work_file in ${FILE}_w*.egsphsp${PHSP_N};
do
    if [[ ! -e ${OUTPUT_FILE} ]]
    then
        mv "${work_file}" "${OUTPUT_FILE}"
        touch "${OUTPUT_FILE}"
    elif [[ `stat -c%s ${work_file}` -gt $HEADER_SIZE ]] # [[ ! -e ${OUTPUT_FILE} ]] then
        $BEAMDP >/dev/null
        BEAMDP_INPUT_HERE
        n
        10
        ${work_file}
        ${OUTPUT_FILE}
        BEAMDP_INPUT_HERE
        if [[ ! $? -eq 0 ]]
        then
            echo "Something went wrong...aborting."
            mv "${OUTPUT_FILE}" "${OUTPUT_FILE}.FAILED"
            exit 4
        fi # [[ ! $? -eq 0 ]]
        let "SIZE_TOTAL+=(`stat -c%s ${work_file}` - HEADER_SIZE)"
        echo "...deleting ${work_file}..."
        rm -f ${work_file}
    else [[ `stat -c%s ${work_file}` -gt 20 ]] # [[ ! $? -eq 0 ]]
        echo "${work_file} less than $HEADER_SIZE bytes, skipping." >&2
    fi
done # for work_file in ${FILE}_w*.egsphsp${PHSP_N};

let "END_SIZE=`stat -c%s ${OUTPUT_FILE}` - HEADER_SIZE"

if [[ ${SIZE_TOTAL} -gt ${END_SIZE} ]]
then
    echo "Total file size (less headers) does not add up! This could be an error!"
    echo "Total added size: ${SIZE_TOTAL}."
    echo "Final file size:  ${END_SIZE}."
else
    MODE=`od -j4 -N1 -a -An ${OUTPUT_FILE}
    [[ $MODE = -gt 1 ]] || let "ENTRY_SIZE=4"
    let "PARTICLE_COUNT=($END_SIZE / $ENTRY_SIZE)"
    echo "Addition complete, final size (less header): $END_SIZE with $PARTICLE_COUNT particles."
fi
else
    echo "No input file given, or incorrect format (did you quote?)..."
    echo "Assuming current directory contains files."
    for file in *w1.egsphsp?
do
        echo "Working with file: ${file}" $0 "${file}"
        exit
        done
fi # [[ $# -eq 1 && -e "$1" ]]}