I, Aishwariya Pattabiraman, hereby submit this original work as part of the requirements for the degree of Master of Science in Computer Engineering.

It is entitled:
Heterogeneous Cache Architecture in Network-on-Chips

Student's name: Aishwariya Pattabiraman

This work and its defense approved by:

Committee chair: Ranganadha Vemuri, PhD
Committee member: Wen Ben Jone, PhD
Committee member: Carla Purdy, C, PhD

2070
Heterogeneous Cache Architecture in Network-on-Chips

A thesis submitted to the
Division of Research and Advanced Students
of the University of Cincinnati

in partial fulfillment of the requirements
for the degree of

Master of Science
in the School of Electronic and Computing Systems
of the College of Engineering & Applied Sciences
University of Cincinnati
October 2011

by

Aishwariya Pattabiraman

B.Tech, Electronics and Communication Engineering
SASTRA University, Thanjavur, India
May 2008

Thesis Advisor and Committee Chair: Ranga Vemuri
Heterogeneous Cache Organization in Network-On-Chips

Abstract

Current trends in multicore research suggest that hundreds of cores will be integrated on a single chip in the future for high performance. Increasing the number of cores increases execution speed. However, performance of the system also depends on the cache access speed. Several ideas have been suggested for the cache management in multicore systems. The wire delay in large unified on-chip caches and the need for higher bandwidth have made banked caches connected by two dimensional switched network the choice for the last level cache organization. In the NoC structure accessing nearby cache banks is faster than accessing remote banks. Hence it is called Non-Uniform Cache Architecture (NUCA) [7]. In NUCA since the cache access latency depends on the cache bank that is accessed, we need to find efficient methods to place data in the cache banks close to the accessing core.

Data migration methods migrate data lines to the cache banks close to the accessing cores during their runtime. However, good initial data placement methods are necessary to achieve low cache access times. They can be used along with the existing data migration schemes. Many software techniques like locality aware data placement and management of cache capacity allocation between processes have been suggested in literature [9][10][14]. However, they fail to take advantage of the physical distribution of last level cache among the tiles. Much less has been done to combine both hardware and software techniques to reduce the cache access latency.

In this thesis, we have shown that equal distribution of the cache among the tiles in NoCs may not be the optimal cache distribution for all workloads. Therefore we propose static heterogeneous cache architecture for multi-programmed workloads. The heterogeneity and the appropriate scheduling by the OS will help to reduce network hops by placing more cache blocks close to the cores executing data intensive process.
Furthermore, we also propose dynamic heterogeneous cache architecture for multi-threaded workloads. In multi-threaded workloads, data lines are shared by a number of cores. Initial placement of the shared data lines close to one of the accessing cores may lead to higher access times for other cores. Also, the optimal cache configuration varies depending on how the data is shared between processes in each workload. These aspects are considered in this work to formulate the page coloring and cache allocation as a placement problem. A constructive heuristic has been presented which gives the optimal cache configuration and page coloring for each workload. Both static and dynamic cache configuration exploit the underlying architecture and existing OS level software policies to provide lower cache access latencies for future CMPs. Finally, both of these methods are scalable and suit future workloads effectively.
Acknowledgements

I would like to take this opportunity to thank all the people who helped me in this work and made the graduate school memorable.

I would like to thank Dr. Ranga Vemuri for all his guidance through this work. I would like to thank him for the knowledge he imparted through the classes and research work. I will always be grateful to him for giving me the opportunity to work with him in the lab.

I thank Dr. Jone and Dr. Purdy for being in my thesis committee. Thanks for taking time off your busy schedules to review my work. I would like to thank both of them for all the wisdom and guidance they provided during the course work.

I am thankful to Annie Avakian for the willingness to collaborate with me. I acknowledge all her inputs to this work and the efforts to review my writing. I would also like to thank Mike for reviewing my writing.

I am grateful to all my friends Shruthi, Vaidehi, Vignesh and Vishaul for making my graduate life in Cincinnati enjoyable. I am thankful to Lakshmi Narasimhan for all the support during the course work and during the initial phases of life away from home. Your support was very crucial for me to cope with the new environment.

I want to thank Manoj for all the patience and support during the final year of my stay in Cincinnati. The good times we had together in Cincinnati will never be forgotten. I am forever indebted to my parents and sister for their unconditional love and support. I am sure, but for their encouragement, this would not have been possible.
Contents

1 Introduction ........................................................................................................................................ 1
  1.1 Motivation for Thesis .................................................................................................................. 1
  1.2 Related Research ........................................................................................................................ 3
  1.3 Thesis Statement .......................................................................................................................... 3
  1.4 Organization of Thesis ................................................................................................................ 4
2 Heterogeneous Cache Architecture .................................................................................................. 5
  2.1 Factors that affect data retrieval time ........................................................................................ 6
  2.2 Random Search for Better Configuration ..................................................................................... 9
  2.3 Experimentation and Results ........................................................................................................ 10
  2.4 Observations and Conclusions ..................................................................................................... 12
3 Heterogeneous Cache - Static Architecture ....................................................................................... 13
  3.1 Proposed Heterogeneous Architecture ....................................................................................... 13
  3.2 Proposed Process Scheduling ...................................................................................................... 16
  3.3 Experimental Platform ................................................................................................................ 17
    3.3.1 Benchmark Generation .......................................................................................................... 17
    3.3.2 NoC Simulation Platform ..................................................................................................... 17
    3.3.3 Process scheduler .................................................................................................................. 18
    3.3.4 Simulation Flow .................................................................................................................... 18
  3.4 Experimental Results ................................................................................................................... 19
    3.4.1 Average Latency .................................................................................................................... 19
    3.4.2 Bus Latency .......................................................................................................................... 23
    3.4.3 Network Latency ................................................................................................................... 23
    3.4.4 Hop Count ............................................................................................................................. 24
    3.4.5 Cache sizes ............................................................................................................................ 25
4 Heterogeneous cache- Dynamic Architecture .................................................................................. 28
  4.1 Related Work ............................................................................................................................... 28
4.2 Problem Statement ................................................................. 28
4.3 Problem modeling and Assumptions ........................................... 31
  4.3.1 Objective, Cost function and Constraints .................................. 31
  4.3.2 Numerical Techniques- Force Directed .................................... 31
  4.3.3 Proposed Algorithm ............................................................. 33
4.4 Experimental Platform ............................................................ 36
  4.4.1 Synthetic Benchmark Generator .............................................. 36
  4.4.2 PARSEC Benchmarks: Princeton Application Repository for Shared Memory Computers .......................................................... 36
  4.4.3 Cache log pre-processing ....................................................... 37
  4.4.4 NoC Simulation Platform ....................................................... 37
  4.4.5 Simulation Flow ................................................................. 38
4.5 Results ........................................................................................ 39
  4.5.1 Synthetic Benchmarks ............................................................ 39
  4.5.2 PARSEC Benchmarks ............................................................ 43
  4.5.3 Observations ......................................................................... 48
  4.5.4 Comparison between Random Search and Force directed heuristic .......................................................... 50
4.6 Hardware Requirements ............................................................. 52
5 Conclusion and Future Work ......................................................... 53
  5.1 Conclusion ............................................................................. 53
  5.2 Future Work .......................................................................... 54
Bibliography ..................................................................................... 55
Appendix .......................................................................................... 58
List of Figures

Figure 1.1: 2D Mesh architecture-3x3 Routers................................................................. 2
Figure 2.1: Hybrid bus- network architecture (Source [3]) .............................................. 6
Figure 2.2: Average Latency in cycles for various percentages of remote cache access in a hybrid network with 16 routers and 6 cores per router. ................................................................. 7
Figure 2.3: Average Latency in cycles for various rates of request generation in a hybrid network with 16 routers and 6 cores per router. ................................................................. 8
Figure 2.4: Homogeneous Cache config Figure 2.5: A valid heterogeneous Cache config 10
Figure 2.6: Comparison of Average Latency between homogeneous and heterogeneous configuration given by random search in 4x4 Router network .................................................. 11
Figure 2.7: Comparison of Average Latency between homogeneous and heterogeneous configuration given by random search in 5x5 Router network .................................................. 11
Figure 2.8: Comparison of Average Latency between homogeneous and heterogeneous configuration given by random search in 6x6 Router network .................................................. 12
Figure 3.1: Proposed Static Heterogeneous Cache architecture for 4x4 Router Network ........ 14
Figure 3.2: Proposed Static Heterogeneous Cache architecture for 5x5 Router Network ........ 15
Figure 3.3: Proposed Static Heterogeneous Cache architecture for 6x6 Router Network ........ 15
Figure 3.4: Proposed process allocation for Static Heterogeneous Cache Configuration in 4x4 Network .................................................................................................................. 16
Figure 3.5: Simulation Flow from workload generation to results ....................................... 18
Figure 3.6: Comparison of average latency between homogeneous and static heterogeneous (6-4-2) configuration in 4x4 Router network ................................................................. 20
Figure 3.7: Comparison of average latency between homogeneous and static heterogeneous (6-4-2) configuration in 5x5 Router Network ................................................................. 21
Figure 3.8: Comparison of average latency between homogeneous and static heterogeneous (6-4-2) configuration in 6x6 Router Network ................................................................. 22
Figure 3.9: Comparison of bus latencies between homogeneous and static heterogeneous cache configurations in 5x5 network ................................................................. 23
Figure 3.10: Comparison of network latencies between homogeneous and static heterogeneous cache configurations in 5x5 network ................................................................. 24
Figure 3.11: Comparison of hops between homogeneous and static heterogeneous cache configurations in 5x5 network. ................................................................. 24
Figure 3.12: Average latency variation with respect to cache ratio in 4x4 network. .......... 25
Figure 3.13: Average latency variation with respect to cache ratio in 5x5 network. .......... 26
Figure 3.14: Average latency variation with respect to cache ratio in 6x6 network. .......... 26
Figure 4.1: Homogeneous cache Configuration in NoC with two cache blocks per router. ...... 29
Figure 4.2: A valid dynamic Heterogeneous Cache Configuration in NoC .................. 30
Figure 4.3: Sample Simics Cache Access log(Source [20]) ....................................... 37
Figure 4.4: Simulation Flow for PARSEC benchmarks. ........................................... 38
Figure 4.5: Comparison of average latencies over four iterations of force-directed heuristic for 4x4 network- Requests per process-1000 .......................................................... 40
Figure 4.6: Comparison of average latencies between homogeneous configuration and heterogeneous configuration given by force-directed heuristic for 4x4 network. Requests per process-1000 .......................................................... 40
Figure 4.7: Comparison of average latencies over four iterations of force-directed heuristic for 5x5 network. Requests per process-1000 .......................................................... 41
Figure 4.8: Comparison of average latencies between homogeneous configuration and heterogeneous configuration given by force-directed heuristic for 5x5 network. Requests per process-1000 .......................................................... 41
Figure 4.9: Comparison of average latencies over four iterations of force-directed heuristic for 6x6 network. Requests per process-1000 .......................................................... 42
Figure 4.10: Comparison of average latencies between homogeneous configuration and heterogeneous configuration given by force-directed heuristic for 6x6 network. Requests per process-1000 .......................................................... 42
Figure 4.11: Comparison of average latencies between homogeneous cache configuration and heterogeneous configuration after each iteration of the heuristic in PARSEC benchmarks simulated under 16 cores. .......................................................... 43
Figure 4.12: Comparison of average latencies between homogeneous cache configuration and force directed heterogeneous configuration of PARSEC benchmarks simulated under 16 cores. 44
List of Figures

Figure 4.13: Comparison of average latencies between homogeneous cache configuration and force directed heterogeneous configuration of PARSEC benchmarks simulated under 32/128 cores. ................................................................. 46

Figure 4.14: Comparison of average latencies between homogeneous cache configuration and force directed heterogeneous configuration of PARSEC benchmarks simulated under 32/128 cores. ................................................................. 46

Figure 4.15: Dynamic Heterogeneous Configuration obtained for Streamcluster .................... 48

Figure 4.16: Comparison of average latencies of different cache configurations: homogeneous, the one obtained through random search and the one suggested by force-directed heuristic in 5x5 network. ........................................................................ 50

Figure 4.17: Comparison of iteration count for obtaining better cache configurations through random search and force-directed methods in 5x5 network. ................................................................. 51

Figure 4.18: Comparison of execution time of the code for obtaining better cache configurations through random search and force-directed methods in 5x5 network. ................................................................. 51

Figure 4.19: A reconfigurable architecture for dynamic heterogeneous cache. ....................... 52

Figure 6.1: Average Latency in cycles for various percentages of Remote cache access in a hybrid network with 25 routers and 6 cores per router. ................................................................. 58

Figure 6.2: Average Latency in cycles for various rates of request generation in a hybrid network with 25 routers and 6 cores per router. ........................................................................ 58

Figure 6.3: Average Latency in cycles for various percentages of Remote cache access in a hybrid network with 36 routers and 6 cores per router. ................................................................. 59

Figure 6.4: Average Latency in cycles for various percentages of Remote cache access in a hybrid network with 36 routers and 6 cores per router. ................................................................. 59

Figure 6.5: Comparison of average latencies between homogeneous and static heterogeneous configuration (5-4-3) in 4x4 Router network. ................................................................. 60

Figure 6.6: Comparison of average latencies between homogeneous and static heterogeneous configuration (7-4-1) in 4x4 Router network. ................................................................. 60

Figure 6.7: Comparison of average latencies between homogeneous and static heterogeneous configuration (5-4-3) in 5x5 Router network. ................................................................. 61

Figure 6.8: Comparison of average latencies between homogeneous and static heterogeneous configuration (7-4-1) in 5x5 Router network. ................................................................. 61
Figure 6.9: Comparison of average latencies between homogeneous and static heterogeneous configuration (5-4-3) in 6x6 Router network................................................................. 62
Figure 6.10: Comparison of average latencies between homogeneous and static heterogeneous configuration (7-4-1) in 6x6 Router network................................................................. 62
Figure 6.11: Comparison of average latencies over four iterations on synthetic benchmarks for 4x4 router- Requests per process-5000............................................................................. 63
Figure 6.12: Comparison of average latencies between homogeneous configuration and heterogeneous configuration given by force-directed heuristic for 4x4 network.- Requests per process-5000 ......................................................................................................................... 63
Figure 6.13: Comparison of average latencies over four iterations on synthetic benchmarks for 5x5 router- Requests per process-5000............................................................................. 64
Figure 6.14: Comparison of average latencies between homogeneous configuration and heterogeneous configuration given by force-directed heuristic for 5x5 network.- Requests per process-5000 ......................................................................................................................... 64
Figure 6.15: Comparison of average latencies over four iterations on synthetic benchmarks for 6x6 router- Requests per process-5000............................................................................. 65
Figure 6.16: Comparison of average latencies between homogeneous configuration and heterogeneous configuration given by force-directed heuristic for 6x6 network.- Requests per process-5000 ......................................................................................................................... 65
Figure 6.17: Comparison of average latencies over four iterations on synthetic benchmarks for 4x4 router- Requests per process-10000............................................................................. 66
Figure 6.18: Comparison of average latencies between homogeneous configuration and heterogeneous configuration given by force-directed heuristic for 4x4 network.- Requests per process-10000 ......................................................................................................................... 66
Figure 6.19: Comparison of average latencies over four iterations on synthetic benchmarks for 5x5 router- Requests per process-10000............................................................................. 67
Figure 6.20: Comparison of average latencies between homogeneous configuration and heterogeneous configuration given by force-directed heuristic for 5x5 network.- Requests per process-10000 ......................................................................................................................... 67
Figure 6.21: Comparison of average latencies over four iterations on synthetic benchmarks for 6x6 router- Requests per process-10000............................................................................. 68
Figure 6.22: Comparison of average latencies between homogeneous configuration and heterogeneous configuration given by force-directed heuristic for 6x6 network.- Requests per process-10000 ........................................................................................................................................... 68

Figure 6.23: Comparison of average latencies of different cache configurations: homogeneous, the one obtained through random search and the one suggested by force-directed heuristic in 4x4 network. ........................................................................................................................................... 69

Figure 6.24: Comparison of iteration count for obtaining better cache configurations through random search and force-directed methods in 4x4 network. ........................................................................................................................................... 69

Figure 6.25: Comparison of execution time of the code for obtaining better cache configurations through random search and force-directed methods in 4x4 network. ........................................................................................................................................... 70

Figure 6.26: Comparison of average latencies of different cache configurations: homogeneous, the one obtained through random search and the one suggested by force-directed heuristic in 6x6 network. ........................................................................................................................................... 70

Figure 6.27: Comparison of iteration count for obtaining better cache configurations through random search and force-directed methods in 6x6 network. ........................................................................................................................................... 71

Figure 6.28: Comparison of execution time of the code for obtaining better cache configurations through random search and force-directed methods in 6x6 network. ........................................................................................................................................... 71
List of Tables

Table 3.1: Percentage Improvement in Average Latency for 4x4 Network under static heterogeneous configuration.................................................................20
Table 3.2: Percentage Improvement in Average Latency for 5x5 Network under static heterogeneous configuration.................................................................21
Table 3.3: Percentage Improvement in Average Latency for 6x6 Network under static heterogeneous configuration.................................................................22
Table 4.1: Execution times and number of requests processed in PARSEC benchmarks simulated under 16 cores. ........................................................................................................44
Table 4.2: Percentage Improvement in average latency of PARSEC benchmarks simulated under 16 cores. ........................................................................................................45
Table 4.3: Execution Times of the simulation and No. of Requests of PARSEC benchmarks simulated under 32/128 cores. ........................................................................................................47
Table 4.4: Percentage Improvement in average latency of PARSEC benchmarks simulated under 32/128 cores. ........................................................................................................47
Table 4.5: Access Patterns of PARSEC benchmark- Streamcluster under 4x4 mesh network....49
Chapter 1

Introduction

The increasing demand for more computational power has led us to the era of Chip Multiprocessors (CMPs). Earlier, processing speed was improved by increasing the frequency of the single processor systems. The diminishing speed improvements versus the power consumption have led to the integration of more transistors on the same chip in order to build multiple cores [1]. Multicore processors with two and four cores are commercially available [2]. Recently Intel has released a prototype of 80 core processor- the Teraflop Research chip [4]. These cores operate independent of one another rendering high processing power and share the memory system.

The two key parameters in building efficient processors are data processing speed and memory access speed. While the throughput of the CMP systems increases with the number of cores, cache access delay also plays an important role. Novel ideas have been suggested in multicore cache organization to achieve low data retrieval time. The common memory organization has private L1 caches for all the cores and a shared L2 cache [5], [6].

Previously, the workload of the CMPs consisted of processes that were sequential and independent but now we have mixed workload consisting of both independent processes and ones which share data among numerous cores [17], [26]. Hence it is essential to have efficient cache organization which can support future workloads in order to utilize multicore processors to its fullest potential.

1.1 Motivation for Thesis

The cache organization in CMPs is an active area of research. The last level cache is often shared by all the processors in CMPs to maximize utilization. Different architectures for the shared L2 cache exist in literature. Initially, a single unified L2 cache was considered. The drawbacks of using a single unified L2 cache (increased wire delays and heavy traffic with
increasing number of cores) led to Network On Chip (NoC) architectures in which the L2 cache is physically distributed and logically shared through network driven protocols [11].

Different topological structures of NoCs exist of which the most commonly considered is the 2D Mesh architecture [12]. NoCs provide parallel access to data in contrast to the unified cache designs which provide sequential access. The cache accesses in NoCs can suffer various network related latencies. In the physically distributed, banked cache structure accessing nearby cache banks is faster than accessing remote ones and hence the name Non Uniform Cache Architecture (NUCA) [7] [8]. If the data lines are mapped to specific location, similar to the traditional unified cache, it is known as Static NUCA (SNUCA) [7]. If the data is permitted to be mapped to many banks within the cache, the cache lines can be migrated to the cache banks that are close to the accessing cores during their runtime. This method of mapping is known as Dynamic NUCA (DNUCA) [7].

From the NUCA it is clear that data retrieval times in CMPs can be improved through architectural changes and efficient data mapping. Ideas that combine advantages of both distributed architecture and flexible data mapping are necessary for lower data retrieval time.

Figure 1.1: 2D Mesh architecture-3x3 Routers
1.2 Related Research

Several works have been published for the last level cache distribution in NoCs. NUCA architecture suggested by Beckmann and Wood [8] is the design trend for CMPs. Improvements over NUCA for managing network latencies have been suggested by Kim et al. [7]. Cho and Jin [9] have proposed allocation of pages in local cache blocks using the OS and page coloring techniques. Although this provides low latencies when the workload is multi-programmed with minimal data sharing, it does not provide a solution to future multi-threaded workloads.

Hammoud et al. [10] proposed changing the sharing degree of L2 caches among the threads in order to provide sufficient cache to all the threads. This caters to the varying workloads among the tiles making certain cache slices private and others shared by a varying degree. However, this method is costly in terms of cache coherency and locating the cache lines in larger systems.

1.3 Thesis Statement

In order to decrease the data retrieval time during the cache access in tiled CMP architecture, we proposed two methods in this thesis. Both these solutions take advantage of the physically distributed cache and existing OS level policies for NUCA architecture. The architectures explored so far in the literature have equal amount of cache on all the tiles. This is referred as the homogeneous architecture throughout this thesis. The high core to router density of hybrid architecture and future workloads suggests dividing the available L2 cache equally among all routers may not be the optimal configuration for a given workload. In the hybrid tiled architecture we have to balance the latency incurred by the accesses to the local cache (bus latency) and the latency incurred by the accesses to remote caches (network latency). Hence, depending on the data intensity of processes the optimal size of the L2 cache block in each tile might vary. The proposed heterogeneous cache architectures balance the bus and the network latency components. The three main contributions of this work are:
1. It has been shown that for a given workload, in the tiled architecture, the homogeneous cache configuration may not be the optimal cache configuration. There may exist better heterogeneous configurations compared to the homogeneous cache architecture. A better configuration is the one which has lower data retrieval time than the existing architecture.

2. Static heterogeneous architecture has been proposed which suggests a fixed, unequal cache distribution for the 2D mesh tiled architecture. It has been shown that the number of cycles required to access the cached data is lower in the static heterogeneous configuration compared to the homogeneous cache configuration.

3. Dynamic heterogeneous architecture has been proposed in which cache distribution can be changed among the routers depending on the workload. A heuristic which determines the ideal cache distribution for a given workload has been presented. OS level page coloring concept [9], which allows the OS to cache a page of a specific color in a particular cache block has been assumed as the data mapping policy.

1.4 Organization of Thesis

The thesis is organized into the following chapters:

Chapter 2 explains the idea of heterogeneous cache configuration in tiled CMP architecture. It explains the factors that affect the cache access times. It provides an insight as how there might exists an organization which is better than the homogeneous cache architecture through a random greedy search method.

Chapter 3 explains the proposed idea of static heterogeneous architecture. It gives details of the hardware and the software requirements to implement the proposed idea. Experimental platform is explained followed by the results of the experiments.

Chapter 4 explains the dynamic heterogeneous architecture. The assumptions on page allocation and page coloring are explained. The heuristic which determines the cache distribution for a given workload is also presented. The experimental platform is shown and the results are presented explaining the important parameters that can be chosen and varied.

Chapter 5 concludes this work and presents the possibilities of future work.
Chapter 2

Heterogeneous Cache Architecture

The 2D mesh architecture [11][12] is the basis for our design and experiments. In the traditional mesh architecture every processor core is associated with a router. L1 caches are often kept private to the core for lower hit times. Each router manages a portion of the L2 cache. The communication between routers takes place through network protocols. The last level cache is shared among all the cores for better utilization [14]. The 2D mesh architecture generally has physically distributed and logically shared L2 cache. In the physically distributed L2 cache, access latency depends on the location of data with respect to the requesting core. In other words the latency depends on the number of network hops the requests have to make to fetch the data. The number of hops depends on the physical proximity of the cache block.

Other architectures have been proposed recently that try to take advantage of both the bus and the network based architecture. Das et al. [11] suggest having the bus based architecture for local communication and the router based mesh architecture for global routing which is referred as hybrid bus-mesh architecture. It has been shown that the hybrid architecture suffers lower latencies compared to the traditional mesh architecture because of the significant decrease in the hop count [11]. Avakian et al. [3] suggests having the number of cores that are mapped to the bus reconfigurable. This will enable cores running related process to be grouped together.

In hybrid NoCs where the L2 is shared, the data requested by a processor can be cached anywhere in the address space. When a processor requests cached data, it can be present either in the nearest/local cache or in the any of the remote cache slices. When the requested data is present in the local cache slice, the request is queued up until the bus can process the request. When the requested data is present in the remote cache slice, a request is sent to the target router from the router of the requesting core. The request is serviced and the data is sent back to the originating router. This method of communication involves the overhead of network protocols, packetizing and de-packetizing the requests. Typical hybrid bus-network architecture is shown in figure 2.1.
2.1 Factors that affect data retrieval time

The following two factors contribute to the latency of data retrieval process.

1. Number of hops

In NoC, the location where a data line is cached depends on the physical address. Though physically distributed cache provides parallel access to data, it also provides non uniform latency for data access. Since data can be present in any cache block, access latency depends on the physical proximity of the cache block which contains the data. Each request through the network is packetized (known as flits) at the originating router and is de-packetized at the target router. The latency of the data request to the remote cache block hence depends on the network overhead and the distance between the source and the target routers (number of hops).

2. Request Rate

Traditionally, bus architecture was used to connect two or more cores to the shared L2 cache. Buses occupy less area, have simple protocols and hence provide faster communication when data access is not intensive. However buses provide sequential access to the L2 cache.
block connected to it resulting in lengthy queues when data request rate is high. Das et al [11] suggest limiting the number of cores that are connected to the bus to eight. As the number of cores connected to the bus increases, the queue builds up resulting in large wait times for the requests to get processed. The rate at which the queue builds up not only depends on the number of cores; but also on the data intensity of the threads running on the cores. This is called bus saturation [11]. Hence the rate at which the requests are generated from the cores connected to the same bus and the numbers of requests made to the local cache slice are important factors which determine processing time of the requests.

Average Latency

The average processing time of the requests generated by all the cores is measured by average latency. It takes both the number of hops and the queue delays into account. Hence average latency is the measure of the overall cache access time of the architecture.

Experiments to study the factors that affect the average latency were performed on a hybrid NoC platform developed in C++. More details on the experimental setup are given in Chapters 3 and 4. Figures 2.2 and 2.3 show the factors that affect the average cache access time.

Figure 2.2: Average Latency in cycles for various percentages of remote cache access in a hybrid network with 16 routers and 6 cores per router.
Chapter 2: Heterogeneous Cache Architecture

Figure 2.3: Average Latency in cycles for various rates of request generation in a hybrid network with 16 routers and 6 cores per router.

Figure 2.2 shows the variation in average latency for various percentages of remote cache access. The remote cache access percentages indicate the percentage of the total requests that has to hop through the network to get serviced. From the figure 2.2 it can be seen that as the remote cache access increases, the average latency increases. So caching data in the local cache block or as close as possible, which can reduce the number of hops is necessary. But making all the data local is not possible when the data lines are shared among various cores. Even when data lines are not shared among the cores, caching all the data lines accessed by a process in the local cache block may not be beneficial. If all the cache access for a data intensive process are sequential and happen through a single port, the wire delay of the large cache block will dominate over the decrease in latency [8]. Hence there is an upper limit for the size of the cache block that can be associated with a router.

Figure 2.3 shows the effect of cache access rate on the average latency. We can see that average latency increases as the request rates increase due to the bus saturation. The reason is that if the rate at which the requests are generated is high, lengthy queue builds up. There is a clear jump in the average latency above 20% because of the bus saturation as mentioned earlier. In this case only increasing the number of ports per cache slice will help, but it has been shown
to be very costly. Further experiments which confirm the effect of these factors on average latency in large meshes are provided in the Appendix.

2.2 Random Search for Better Configuration

The bus latency and the network latency are the two components which contribute to the average latency of the shared L2 cache access in the hybrid NoC as discussed above. Location aware data mapping using OS page allocation [9] has been suggested in literature which can map data to the local cache block of the processors accessing it. This can reduce the network hops significantly but might result in bus saturation for hybrid NoCs which in turn will increase the bus delay. In other words reducing the network hops does not necessarily mean reduction in average latency.

In all the NoC architectures proposed so far, the available L2 cache has been divided equally among all the routers. Therefore, regardless of the processes associated with the cores, the cores connected to a router can have only a fixed amount of data in the local cache block. Hence dividing the cache equally among all the routers may not give enough local cache space to all the cores since the processes are varied. If equal division of cache among the routers is not the optimal configuration of the cache, the question arises as to what will be the optimal amount of cache in every tile for a given workload. The optimal amount of cache for every router strictly depends on the workload associated with each router. Throughout this thesis, the architecture in which the L2 cache is equally distributed among the routers is referred to homogeneous cache architecture and the proposed one that has unequal cache distribution is referred to heterogeneous cache architecture.

Hypothesis

The homogeneous cache configuration may not be the optimal cache configuration for a given workload. There might exist a heterogeneous cache configuration for which the average latency is lower than the homogeneous cache configuration. The optimal cache configuration depends on the access patterns and hence varies with the workload.
2.3 Experimentation and Results

To prove the hypothesis, synthetic benchmarks were generated and tested in the hybrid NoC architecture. The modeling of the architecture and the simulations were done in C++. More details on the experimental platform are given in Chapter 4. Random search was performed to find a heterogeneous configuration which has lower average latency compared to the homogeneous configuration. Synthetic benchmarks with various access patterns were simulated. The algorithm to find the heterogeneous configuration was a greedy random search algorithm.

The algorithm for random search is explained below.

1. The benchmark is simulated under the homogeneous cache architecture.
2. The cache blocks are sorted in decreasing order based on the number of accesses.
3. Starting with the equal cache distribution, each cache block is moved to a router where the average latency is reduced. Once the cache block finds the place, it is locked and is not moved further.
4. Once all the cache slices find its new place, the algorithm exits.
Experiments were performed on 4x4, 5x5 and 6x6 mesh sizes and the results are shown in the Figures 2.6, 2.7 and 2.8.

**Figure 2.6:** Comparison of Average Latency between homogeneous and heterogeneous configuration given by random search in 4x4 Router network.

**Figure 2.7:** Comparison of Average Latency between homogeneous and heterogeneous configuration given by random search in 5x5 Router network.
Figure 2.8: Comparison of Average Latency between homogeneous and heterogeneous configuration given by random search in 6x6 Router network.

2.4 Observations and Conclusions

From the figures 2.6, 2.7 and 2.8 we conclude that for any given workload, multithreaded or multi-programmed, regardless of whether the data is shared or not, the homogeneous cache distribution may not be the optimal cache distribution for the CMP 2D-tiled architecture.

In the case of multi-programmed workloads in which the cache lines are not shared among the cores, OS based page allocation [9] will help to cache data in the nearest cache slice. However the optimal amount of cache that has to be allocated to each tile depends on the data intensity of the associated process. Hence heterogeneous cache architecture with appropriate scheduling may serve the purpose and provide better performance. In the case of multi-threaded workload in which the cache lines are heavily shared between related processes, page mapping is an issue. Also the naïve OS level page allocation [9] which cache the data lines close to the initial user, may not provide lower cache access times since successive sharers of data will suffer from increased number of network hops. Therefore we need a heuristic for finding the optimal page mapping and heterogeneous architecture depending on the workload information.
Chapter 3

Heterogeneous Cache - Static Architecture

In Chapter 2 it was shown that for a given workload, the homogeneous cache configuration may not be the optimal cache configuration. When the processes vary in data intensity and when NoCs are presented with mixed workloads, a heterogeneous cache structure can bring down the overall cache access time. A static heterogeneous cache configuration for 2D mesh architecture is presented in this work. The target workload consists of multi-programmed processes with minimum sharing of lines between them. Although OS level page allocation suggested in [9], [22] to map the data accessed by the cores to the nearest cache slice has been proven to reduce the number of hops required to access the data, the heterogeneous architecture provides room for improvement by reducing the number of hops further and by minimizing queue delays.

In order to reduce average latency, data intensive threads should have enough cache space close to them such that they can cache their required data locally. Another factor that has to be considered is queue delay. If all the highly data intensive threads are scheduled on closely located tiles, accesses to the spilled data will take a long time since the requests have to traverse long queues. Both these problems are addressed in this architecture, one through heterogeneous configuration and another through appropriate scheduling. The architecture is termed static because the size of the cache blocks associated with each router is fixed. The differences between the homogeneous architecture and the proposed heterogeneous architecture are the unequal distribution of cache and the scheduling of the processes.

3.1 Proposed Heterogeneous Architecture

The heterogeneous cache distribution in 2D mesh architecture presents an unequal cache distribution having a fixed size L2 cache. The difference in the sizes of cache blocks is an important parameter and is addressed in this work. Since the working set sizes can vary widely, there has to be a balance between the largest and the smallest cache block. Large cache blocks would result in sequential accesses and lead to bus saturation for data intensive working sets.
Small cache blocks increase the hop count for data intensive workloads and hence increase the delay. So a generic configuration which works for all the workloads is necessary.

The placement of the bigger and smaller cache blocks is another important consideration. The cache blocks have to be placed such that congestion is minimized. The optimal placement was observed to be such that the larger cache blocks are always at one hop distance from the smaller cache blocks. The reason for implementing this kind of placement is that even if the processes associated with the smaller cache blocks are larger than expected, they can find more cache to spill their data at a few hops distance. The idea presented here is scalable for large mesh sizes. No extra hardware is necessary to realize the static heterogeneous configuration. Since the sizes of cache blocks are determined at design time, experiments have to be done with various unequally sized cache blocks and placements to determine the distribution of cache.

The proposed heterogeneous cache distribution for 4x4, 5x5 and 6x6 mesh sizes are shown below. B,M and S represent three different sizes of cache blocks, ‘B’ being largest cache block size, ‘M’ being the medium sized cache block and ‘S’ being the smallest. The placements of the different sized cache blocks have been done such that large cache blocks are placed in the center of the mesh as much as possible. Also large and small cache blocks are placed in alternate tiles as mentioned before.

Figure 3.1: Proposed Static Heterogeneous Cache architecture for 4x4 Router Network.
Chapter 3: Heterogeneous Cache - Static Architecture

Figure 3.2: Proposed Static Heterogeneous Cache architecture for 5x5 Router Network.

Figure 3.3: Proposed Static Heterogeneous Cache architecture for 6x6 Router Network.
3.2 Proposed Process Scheduling

OS level page allocation ([9],[15],[22]) or page coloring is the data mapping policy. In order to efficiently utilize the heterogeneous cache architecture, the threads and the processes have to be mapped to the appropriate tiles. Just as the OS level page coloring is used to reduce the number of hops in a NoC, the process scheduling in heterogeneous cache configuration is aimed at reducing the total number of hops. Without appropriate scheduling of threads, the heterogeneous structure may result in increased number of hops and hence higher average latency. This is because, when a large workload is scheduled on a tile with a small cache block, only a small portion of the data can be mapped in the local cache block and the rest has to be cached in the neighboring tiles.

A simple process scheduling has been suggested in this work for the heterogeneous cache architecture. Assuming OS has some knowledge of the data intensity of the processes at any given time, the scheduler can sort the processes in the decreasing order of data intensity. The processes can then be allocated to the tiles with large cache block first followed by tiles with small cache blocks. This ensures that, the data intensive processes will have a large cache block locally and hence the hop count will be reduced. If the list of processes arranged in decreasing order of data intensity are P1, P2, P3 .... P16 which constitute the workload for a 4x4 Router network, the process allocation would be similar to the one shown in Figure 3.4.

![Figure 3.4: Proposed process allocation for Static Heterogeneous Cache Configuration in 4x4 Network.](image-url)
3.3 Experimental Platform

The simulation and testing of the static heterogeneous cache architecture has been carried out using various pieces of software. Platforms were developed in C++ for generating workloads, to simulate the NoC architecture and to simulate proposed process scheduling. Together they provide a complete experimental platform to study and validate the effectiveness of the proposed architecture.

3.3.1 Benchmark Generation

This piece of software provides useful workload for experimentation. The synthetic benchmarks simulate multi-programmed workload with little sharing of lines between processes. Various parameters can be adjusted manually in the generation of these benchmarks so that the target architecture can be stressed sufficiently. The number of processes can be determined by specifying the number of routers.

1. Number of requests: The number of requests generated by each tile is an independent parameter which can be adjusted for each process. This helps to define the total number of requests each process generates during runtime.
2. Rate of requests: Another important parameter is the rate at which the cores generate cache requests. This determines the rate at which the queues are filled and hence the congestion in the network.
3. Data intensity of the Process: Data intensity represents the amount of data that has to be cached for each process. This can be adjusted manually to present a mixed workload for the system.

3.3.2 NoC Simulation Platform

The NoC simulation platform is a custom simulator built in C++. The hybrid NoC architecture has been carefully modeled. The platform is generic, in which user can manually adjust various parameters for simulation. The number of routers and the total number of cache blocks can be varied depending on the mesh size that has to be simulated. The routers follow XY routing algorithm and the default data mapping policy is similar to the one described in [9],[22].

The NoC simulator takes in the architecture specifications and the benchmarks as input and produces the average latency incurred by the cores in accessing the L2 cache as the output.
The simulator simulates both the homogeneous and the heterogeneous configuration of L2 cache. For the heterogeneous configuration additional details of the architecture have to be provided.

3.3.3 Process scheduler

The process scheduler is an integrated module with the benchmark generating software. It can schedule the processes of the current simulation to the tiles according to the data intensity. As described earlier the process scheduling is done only for the simulation of the benchmarks under heterogeneous architecture.

3.3.4 Simulation Flow

Figure 3.5 shows the entire flow of the simulation setup. It shows how various pieces of the code have been put together to generate this NoC analysis. The squares represent two major pieces of code that have been used to produce results. The diamonds represent the input to the system which can be controlled during simulation. The stacked elements represent the output of the respective systems.

![Simulation Flow Diagram]

Figure 3.5: Simulation Flow from workload generation to results.
3.4 Experimental Results

Experiments were done using the above described experimental platform with synthetic benchmarks. The benchmarks were tested on 4x4, 5x5 and 6x6 mesh sizes. The baseline architecture is the homogeneous cache architecture against which the performance of the heterogeneous cache architecture has been compared. Comparison has been done for all the parameters that influence average latency. The heterogeneous architectures were simulated under different cache distributions in terms of the size of L2 cache blocks in each tile.

Simulations have been shown for mesh sizes 4x4 (16 routers), 5x5 (25 routers) and 6x6 (36 routers). The heterogeneous configurations for these networks are shown in the Figure 3.1, Figure 3.2 and Figure 3.3 respectively. The amount of cache in the three types of cache blocks were varied and tested, keeping the overall cache size equal to that of the homogeneous architecture.

3.4.1 Average Latency

Figures 3.6, 3.7, 3.8 show the difference in the average latency between homogeneous and static heterogeneous cache configuration in a 4x4, 5x5 and 6x6 network respectively. In all the three experiments the cache size ratio among the three types of cache blocks was kept as 6:4:2. Other ratios were explored and have been described in Section 3.5.5.

Figure 3.6 shows the decrease in the average latency for ten different workloads under heterogeneous cache architecture. The decrease in the average latency is attributed to the decreased number of hops in the network and many processes were able to cache all the necessary data in the local cache. However the latency incurred by the requests which access the local cache through the bus will increase because of the queue delay. The improvement in the average number of cycles required to access the cache shows that there is a significant decrease in the network delay which offsets the increase in the queue delay. So the heterogeneous configuration strikes a balance between the network delay and the queue delay.

On an average there is a 14.25% improvement in the average latency in the 4x4 network, 12.07% improvement in 5x5 network and 20.37% improvement in 6x6 network.
Figure 3.6: Comparison of average latency between homogeneous and static heterogeneous (6-4-2) configuration in 4x4 Router network

<table>
<thead>
<tr>
<th>Workload</th>
<th>Improvement in Average Latency (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.49</td>
</tr>
<tr>
<td>2</td>
<td>5.94</td>
</tr>
<tr>
<td>3</td>
<td>15.30</td>
</tr>
<tr>
<td>4</td>
<td>12.97</td>
</tr>
<tr>
<td>5</td>
<td>11.03</td>
</tr>
<tr>
<td>6</td>
<td>16.80</td>
</tr>
<tr>
<td>7</td>
<td>19.44</td>
</tr>
<tr>
<td>8</td>
<td>10.62</td>
</tr>
<tr>
<td>9</td>
<td>18.04</td>
</tr>
<tr>
<td>10</td>
<td>14.43</td>
</tr>
</tbody>
</table>

Table 3.1: Percentage Improvement in Average Latency for 4x4 Network under static heterogeneous configuration.
Figure 3.7: Comparison of average latency between homogeneous and static heterogeneous (6-4-2) configuration in 5x5 Router Network.

<table>
<thead>
<tr>
<th>Workload</th>
<th>Improvement in Average Latency (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.65</td>
</tr>
<tr>
<td>2</td>
<td>5.18</td>
</tr>
<tr>
<td>3</td>
<td>9.37</td>
</tr>
<tr>
<td>4</td>
<td>23.19</td>
</tr>
<tr>
<td>5</td>
<td>17.18</td>
</tr>
<tr>
<td>6</td>
<td>6.15</td>
</tr>
<tr>
<td>7</td>
<td>18.24</td>
</tr>
<tr>
<td>8</td>
<td>13.29</td>
</tr>
<tr>
<td>9</td>
<td>10.38</td>
</tr>
<tr>
<td>10</td>
<td>12.08</td>
</tr>
</tbody>
</table>

Table 3.2: Percentage Improvement in Average Latency for 5x5 Network under static heterogeneous configuration.
Figure 3.8: Comparison of average latency between homogeneous and static heterogeneous (6-4-2) configuration in 6x6 Router Network.

<table>
<thead>
<tr>
<th>Workload</th>
<th>Improvement in Average Latency (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.65</td>
</tr>
<tr>
<td>2</td>
<td>5.18</td>
</tr>
<tr>
<td>3</td>
<td>9.37</td>
</tr>
<tr>
<td>4</td>
<td>23.19</td>
</tr>
<tr>
<td>5</td>
<td>17.18</td>
</tr>
<tr>
<td>6</td>
<td>6.15</td>
</tr>
<tr>
<td>7</td>
<td>18.24</td>
</tr>
<tr>
<td>8</td>
<td>13.29</td>
</tr>
<tr>
<td>9</td>
<td>10.38</td>
</tr>
<tr>
<td>10</td>
<td>12.08</td>
</tr>
</tbody>
</table>

Table 3.3: Percentage Improvement in Average Latency for 6x6 Network under static heterogeneous configuration.
3.4.2 Bus Latency

Figure 3.9 shows the variation in the latency of the requests that get serviced by the local bus in the 5x5 network. There is a significant increase in the bus latency in the heterogeneous cache configuration compared to the homogeneous configuration. This is expected since in the heterogeneous configuration the number of requests to the local cache increases and is serviced through a single bus and port.

![5x5 Router Network](image)

**Figure 3.9:** Comparison of bus latencies between homogeneous and static heterogeneous cache configurations in 5x5 network.

3.4.3 Network Latency

The latency experienced by the requests that access the remote cache blocks is shown in figure 3.10. The decrease in network latency in heterogeneous configuration is due to the decrease in the number of requests to the remote caches and subsequently the number of hops. This significant decrease in the heterogeneous configuration is more than the increase in the bus latency and is the reason for the lower overall average latency. Comparison of network latency in 4x4 and 6x6 networks are shown in the Appendix.
Figure 3.10: Comparison of network latencies between homogeneous and static heterogeneous cache configurations in 5x5 network.

3.4.4 Hop Count

Figure 3.11: Comparison of hops between homogeneous and static heterogeneous cache configurations in 5x5 network.
The decrease in the number of hops in the heterogeneous cache configuration logically follows the decrease in the number of requests to the remote caches. Since each request to the remote cache has to be divided into flits, packetized and de-packetized, lower number of hops reduces the average cache access time provided, it is not dominated by the queue delay. The difference in the hops between homogeneous and the heterogeneous configuration is shown in the figure 3.11.

3.4.5 Cache sizes

So far the results in this section were shown for cache size ratio of 6:4:2 for all the three mesh sizes. Cache split up among the tiles in a heterogeneous cache is an important parameter which determines the average latency. The split up 6:4:2 denotes the ratio of the size of the cache in the three types of tiles B, M and S respectively. A split up ratio of 1:1:1 represents the homogeneous configuration. The average latencies for various ratios of cache size among the routers are shown in the Figures 3.12, 3.13, 3.14.

![Figure 3.12: Average latency variation with respect to cache ratio in 4x4 network.](image-url)

Figure 3.12: Average latency variation with respect to cache ratio in 4x4 network.
Figure 3.13: Average latency variation with respect to cache ratio in 5x5 network.

Figure 3.14: Average latency variation with respect to cache ratio in 6x6 network.
In all three networks, it can be observed that the latency initially decreases with the increase in the ratio of the cache size between the cache blocks which carry more cache and the ones which carry less cache. As the ratio of the cache size increases further, the average latency decreases to a point after which an increase can be observed. The initial decrease in the latency is because the heterogeneity was able to support mixed workloads and reduce remote cache access. The increase in the bus latency due to the sequential access of the cache was offset by the network latency decrease.

However, as the ratio of the cache sizes increase, some tiles do not have enough cache associated with them to support their process and hence the remote cache access for those processes increases. Another reason is the increase in the bus latency because of queue delays. If the decrease in the network delay is not sufficient to offset the increase in the queue delay, it might result in a high average latency. Hence the ratio of cache sizes among the tiles has to be chosen such that it provides a balance between data access through the bus and the network.
Chapter 4

Heterogeneous cache- Dynamic Architecture

Locality aware data mapping is necessary to manage data in shared L2 caches in CMPs. In static data mapping, memory blocks are mapped to fixed cache slice. The number of cycles required to retrieve the data depends on the cache block where the data resides and the time it takes to access the cache block. In a 4x4 network, a program on the corner tile will experience an average of 6 hop network delay (request and response) on each L2 cache access [9]. It is essential to map the data required by the cores to the closest cache block. This method of data mapping reduces network hops and the delay associated with it.

4.1 Related Work

Cho et al. in [9] and [22] have shown that data mapping at the granularity of a page by using techniques like page coloring at OS level can greatly reduce the number of hops across the network. [9] provides an initial placement to reduce the cache latency. However this work suggests coloring pages on the first touch of the core. Coloring pages such that data lines are cached in the nearest cache block to the core which accesses the data first has been proven to work well with multi-programmed workloads.

Future workloads are driving towards highly shared workload where a single data line is shared by several cores [17], [26]. To handle multi-threaded workloads of this kind, an optimal initial placement of data and methods which can handle dynamically changing access patterns is required. Several data migration policies such as optimal migration [19] and directional migration [20] have been suggested and explored that suit future workloads. These can be used orthogonally along with the initial data placement policies.

4.2 Problem Statement

As shown in Chapters 2 and 3, for every workload, there exists a heterogeneous configuration of caches which gives optimal latency. For multithreaded workloads, a fixed data mapping policy as in [9] or a static heterogeneous architecture does not guarantee low memory
access times. In future workloads where several cores share data, caching the shared data close to a particular core will lead to higher data access times with respect to the other cores. The problem of finding optimal initial data placement and cache configuration has been considered.

In this work, dynamic cache architecture and data mapping policy have been proposed which can serve workloads with high degree of data sharing. The dynamic heterogeneous cache architecture presents a reconfigurable layer of cache blocks where the number of cache blocks that is connected to each router is determined before the execution of the current set of processes. The number of cache blocks connected to each router is determined according to the amount of data that has to be cached closer to that tile.

The force directed placement of caches is a new method that addresses the problem of initial data placement in the NoC architecture. We assume that pages can be appropriately colored at the OS level so that data can be cached in any cache block. The solution proposed here is a combination of a dynamic architecture which can be configured according to the workload and the existing OS level page placement techniques [9], [15], [22].

![Figure 4.1: Homogeneous cache Configuration in NoC with two cache blocks per router.](image)
Consider the 4x4 router network shown in Figure 4.1. Each tile is associated with two cache blocks which are colored differently at the OS level. Consider the case where core 1 shares data with core 16. If the shared data is cached in slice 2, such that it is closer to the core 1, on average, requests from the core 16 would make 6 hops to fetch the data. An optimal data placement would color the pages shared by cores 1 and 16 in the color of the cache blocks 11 or 12. If the shared data is cached in the cache blocks associated with tile 6, it would be equidistant from the two sharing cores. The problem can be extended to multiple cores sharing the same data and its placement. Also in the above scenario, if core 6 wants its data to be cached in its local cache block, the architecture has to be flexible so that the number of cache blocks that is mapped to tile 6 can be increased. In a static architecture it is an impossible feat, since the amount of cache slices associated with each tile is fixed, but with the proposed dynamic architecture proposed it can be varied according to the workload information. A valid dynamic heterogeneous cache configuration is shown in figure 4.2

![Figure 4.2: A valid dynamic Heterogeneous Cache Configuration in NoC](image-url)
4.3 Problem modeling and Assumptions

We assume that access patterns of various processes can be obtained from the compiler or from previous executions. If the physical addresses of the data lines accessed and shared by the processes are known, the cache block to which they will be mapped under static mapping can be found. The problem is to find the optimal location in the cache to which the shared data has to be moved from the cache block to which it is statically mapped. Once it is determined where shared data in the cache blocks have to be mapped, number of cache blocks per router can be fixed. Therefore the problem comes down to, given the access patterns of the processes, finding the placement of the shared data and the optimal number of cache blocks in each tile.

The problem can be formulated as a placement problem in which the shared data have to be placed such that the total number of hops and hence the average latency incurred by the cores is minimal. The placement of cache blocks in order to minimize the total number of hops is an NP-Complete problem. Even in the simple case of one dimensional placement of ‘n’ cache slices, there are n!/2 linear arrangement of cache blocks. Hence brute force method is not feasible [25].

4.3.1 Objective, Cost function and Constraints

The objective is to find a physical location for each cache block given the number of accesses made by each tile to each cache block under homogeneous configuration. The cost function is to minimize the total number of hops through the network, through which the average cache access time can be minimized. The constraints are the number of cache blocks that can be connected to a tile. By having no constraint on the size of the cache on a router, the algorithm might place the entire cache on a single router in the worst case to decrease the number of hops. However that would make all the accesses sequential, increasing the average wait time of the requests.

4.3.2 Numerical Techniques- Force Directed

The placement problem can be transformed into numerical optimization problem. Force directed placement is a well known numerical technique and is used in this work to obtain the optimal placement of the shared data [23][24]. The idea behind the force directed placement is that the nodes are connected to each other by weighted nets and exert forces at one another. The magnitude of the force F exerted by one node ‘i’ on another node ‘j’ is proportional to the
weighted net that connects the two and the distance of the separation between them. This is analogous to the Hooke’s law in mechanics. Suppose that node ‘a’ is connected to another node ‘b’ by a net of weight $w_{ab}$ and separated by a distance $d_{ab}$, the force of attraction between the two nodes is proportional to the product $w_{ab} \times d_{ab}$. If the node ‘a’ is connected to several of nodes ‘b’ separated by distance $d_{ab}$ and connected by the nets $w_{ab}$, then the total force on the node ‘a’ $F_a$ is given by

$$F_a = \sum_b w_{ab} \times d_{ab}$$

When a node in such a system is free to move, it will move until it reaches a position where the force experienced by that node is zero. This position is called zero force target position [23] [24] [25]. When all the nodes reach their zero force target location, the cost is minimized. The zero force target locations of the nodes can be found by equating the x and y components of the force to zero.

$$\sum_b w_{ab} \times (x_b^0 - x_a^0) = 0$$

$$\sum_b w_{ab} \times (y_b^0 - y_a^0) = 0$$

Solving for $x_a^0$ and $y_a^0$,

$$x_a^0 = \frac{\sum_b w_{ab} \times x_b}{\sum_b w_{ab}}$$

$$y_a^0 = \frac{\sum_b w_{ab} \times y_b}{\sum_b w_{ab}}$$

This approach for finding the ideal location of the nodes can be generalized as a constructive heuristic for finding the optimal location of shared data. To do that, a bipartite graph is formed between the nodes and the cache blocks which are connected by weighted nets. The nets carry a weight equal to the number of accesses made by the router to that cache block. Starting with the homogeneous cache configuration, one cache block at a time is selected and its zero-force target location is computed. The process can be iterated to improve on the solution obtained. The cache blocks may be moved starting from the one which is accessed the most. The
zero force target location is computed and is moved to the target router from the current router. A constraint is set on the number of cache blocks that can be connected to a single router depending on the access delay. Once the target router is found, several options exist.

a) If the number of cache blocks in the target router is lesser than the maximum limit constraint, then the cache block is moved to the target router and locked for the current iteration.

b) If the number of cache blocks in the target router is equal to the maximum limit constraint, then the cache block in the target router that is not locked for the current iteration is replaced.

c) If the number of cache blocks in the target router is equal to the maximum limit constraint and all the cache blocks are locked, then the seed is placed in the neighboring router which has a free spot.

To avoid infinite loops once the seed is moved to the target router it is locked.

The heuristic is explained below. The algorithm is based on the force directed heuristic shown in [25].

The formula for finding target location is as follows:

\[ X_i = \frac{\sum_r W_{ir} \cdot X_r}{\sum_r W_{ir}} \]

\[ Y_i = \frac{\sum_r W_{ir} \cdot Y_r}{\sum_r W_{ir}} \]

\( r = \) Routers

\( W_{ir} = \) No. of requests from router ‘r’ to cache ‘i’.

\( X_r = \) ‘X’ co-ordinate of router ‘r’.

\( Y_r = \) ‘Y’ co-ordinate of router ‘r’.

4.3.3 Proposed Algorithm

Parameters:

Iteration Limit:
Number of times the algorithm is reiterated.

Abort Limit:
Limit after which the current iteration is aborted and the new iteration is started.

Max_Limit:
Maximum number of cache blocks that can be placed in a router.

**Algorithm force_directed_placement**

Compute total requests sent to each cache block.

Sort Caches in descending order of the no. of access made and store it in a list L

Seed = First module from L

**While (Iteration_count<Iteration_Limit)**

    end_ripple =false

    **while(end_ ripple=false)**

    Get the present router or position of cache;

    Compute the target router of the seed and round off to the nearest integer

    **If (target router != present router and target router has fewer than Max_limit cache blocks)**

    Move the seed to the target router and lock;

    end_ripple=true;

    abort_count=0;

    **else if (target router = present router and target router has < = Max_limit cache blocks**)

    Lock the seed in current router;

    end_ripple = true;

    abort_count=0;

    **else if (target router = as present router and target router has >Max_limit cache blocks**)


Find a neighboring router which has fewer cache blocks than Max_limit and place it there;

Lock the seed;

else if (No. of cache blocks in target router >= Max_limit cache blocks)

Check if all the cache blocks are locked;

If (All the cache blocks are locked)

Find a neighboring router which has fewer cache blocks than Max_limit

Put the cache block in the neighboring router

Lock the cache block;

Abort_count++;

end_ripple=true;

if (abort_count>Abort_limit)

go to the end of the list;

else

Find the cache block which is not locked;

Seed = cache not locked;

end_ripple=false;

abort_count=0;

end while

if (end of list L is reached)

Unlock all the cache blocks;

Iteration_count++;

else

Seed = Next unlocked element in L;

end while
4.4 Experimental Platform

Several pieces of software were used for experimentation. The heuristic was tested on both synthetic and PARSEC benchmarks. PARSEC benchmarks provided real workload for the simulation, testing and validation of the idea. The first piece of code is the synthetic benchmark generator which simulates parallel multi-threaded workload cache access. The second major piece is the pre-processing code which processes PARSEC benchmark cache access log. A NoC simulator which simulates hybrid NoC architecture is integrated with the piece of code which executes the heuristic.

4.4.1 Synthetic Benchmark Generator

The synthetic benchmark generator tool provides the workload to simulate multi-threaded processes which are representative of future workload. The threads can share data among one another and the sharing degree can be controlled externally. The number of threads and various parameters like the data intensity of each thread can be varied to provide different workload conditions.

4.4.2 PARSEC Benchmarks: Princeton Application Repository for Shared Memory Computers

The PARSEC benchmark suite was developed to replicate the multi-threaded workloads that are expected on future multicore devices [17]. Each benchmark provides different workload some of which are video streaming and financial analysis. The PARSEC benchmarks are specifically developed for multicore architectures. The benchmarks were simulated in multicore environment using Simics simulator. Simics is an operating-system-level, instruction-set simulator run in a user defined environment [18], [20]. In the simulation setup each core maintained a private L1 cache with data replicated in L2 cache. The L2 is shared and the memory size was proportional to the number of cores. Simics takes the PARSEC benchmarks and configuration files as input and generates a cache access log similar to the one shown in the figure 4.3.
Each line in the cache access log contains information on the accessed line, accessing core, core instruction and current cycle of the core. Also it contains physical data address, logical data address, tag information and read/write information. The cache log output from the simics serves as the input to the NoC simulation platform.

### 4.4.3 Cache log pre-processing

The cache log from the simics is used as the input to the NoC simulation platform. The necessary information from the cache log is extracted by using pre-processing scripts. Every core in simics runs asynchronously and reports its local clock cycle in the cache log. The pre-processing scripts normalize the clock cycles with respect to a global clock cycle. The output contains only cache accesses from the cores to the cache blocks with respect to a synchronous clock.

### 4.4.4 NoC Simulation Platform

The NoC simulation platform is a custom built C++ tool which can simulate hybrid NoCs. It is a generic tool which is capable of simulating massive NoC structures. This platform takes the simics cache log as the input and can simulate the cache accesses at the packet level. The routers follow XY routing algorithm. The configuration parameters for example like the number of routers, the number of cores, the cache size and number of cache blocks can be controlled for each simulation.

---

| 1L2 | line | | handle_read: transaction is a hit (line 925744) |
| 2L2 | line | | operate: inst. fetch (phys = 0x107700, log = 0x000000, tag = 0x002700, size = 128) |
| 2L4 | line | | handle_read: transaction is a hit (line 925774) |
| 2L5 | line | | operate: inst. fetch (phys = 0x107700, log = 0x000000, tag = 0x002700, size = 128) |
| 2L6 | line | | handle_read: transaction is a hit (line 925774) |
| 2L7 | line | | operate: inst. fetch (phys = 0x107700, log = 0x000000, tag = 0x002700, size = 128) |
| 4L2 | line | | handle_read: transaction is a hit (line 925774) |
| 4L3 | line | | operate: inst. fetch (phys = 0x107700, log = 0x000000, tag = 0x002700, size = 128) |
| 4L4 | line | | handle_read: transaction is a hit (line 925774) |
| 6L2 | line | | handle_read: transaction is a hit (line 925774) |
| 6L3 | line | | operate: inst. fetch (phys = 0x107700, log = 0x000000, tag = 0x002700, size = 128) |
| 6L4 | line | | handle_read: transaction is a hit (line 925774) |
| 8L2 | line | | handle_read: transaction is a hit (line 925774) |
| 8L3 | line | | operate: inst. fetch (phys = 0x107700, log = 0x000000, tag = 0x002700, size = 128) |
| 8L4 | line | | handle_read: transaction is a hit (line 925774) |
| 10L2 | line | | handle_read: transaction is a hit (line 925774) |
| 10L3 | line | | operate: inst. fetch (phys = 0x107700, log = 0x000000, tag = 0x002700, size = 128) |
| 10L4 | line | | handle_read: transaction is a hit (line 925774) |

Figure 4.3: Sample Simics Cache Access log(Source [20])
The function which implements the force directed heuristic is integrated with the NoC simulation platform. The entire simulation flow is shown in figures 4.4 and 4.5. After each iteration of the heuristic, the NoC simulation platform simulates the architecture with the new cache configuration and computes the average latency. The parameters iteration count, maximum cache blocks per router and abort count, which are related to the heuristic, can be varied for each simulation.

4.4.5 Simulation Flow

Figures 4.4 and 4.5 show how various pieces of software interact with each other during the simulation. Figure 4.4 shows the simulation flow of the experiments using synthetic benchmarks and figure 4.5 shows the simulation flow with the PARSEC benchmarks. In both figures, the big squares represent major blocks that are used in the simulation. The diamonds represent user inputs to the primary blocks. The stacked elements represent the generated output from the system.

Figure 4.4: Simulation Flow for PARSEC benchmarks.
4.5 Results

Experiments were done using the described simulation platform on various network sizes and number of cache slices. The average latency or the average number of clock cycles taken for a cache request to get serviced has been chosen to analyze the performance. In all the experiments, the average latency of the dynamic heterogeneous architecture has been compared to that of the homogeneous cache architecture.

4.5.1 Synthetic Benchmarks

Figures 4.6, 4.8, 4.10 show the results of the experiments on synthetic benchmarks. In all the experiments, the iteration count of the heuristic was kept as four. Repeated experiments showed that the heuristic converged in four iterations or less. The maximum number of cache blocks per router was set to two for all the experiments, though this can be changed for each simulation. The reason for having this limit is not to stack all the cache blocks in a single router which might increase the queue delay. Each data set represents a different access pattern and sharing degree. Other configuration parameters are given below along with the figures. Further experimental results are given in the Appendix.

It can be observed from the figures that the heuristic was able to find an optimal cache configuration for almost every workload that was presented. Out of 54 data sets presented here, the heuristic was able to find a better configuration for 48 data sets. Also, improvement in the latency is greater if the data accessed by the cores are confined to a minimum number of cache blocks i.e not so distributed, which is what we expect in future benchmarks. The percentage improvement decreases as the data becomes more distributed. In some cases in which the data required by a core is completely distributed across the NoC, then the homogeneous configuration performs better than any other configuration.

No.of Routers : 16
No.of cache slices : 16
No.of Requests per Process : 1000
Figure 4.5: Comparison of average latencies over four iterations of force-directed heuristic for 4x4 network- Requests per process-1000

Figure 4.6: Comparison of average latencies between homogeneous configuration and heterogeneous configuration given by force-directed heuristic for 4x4 network. Requests per process-1000

No.of Routers : 25
No.of cache slices : 25
No.of Requests per Process : 1000
Figure 4.7: Comparison of average latencies over four iterations of force-directed heuristic for 5x5 network. Requests per process-1000

Figure 4.8: Comparison of average latencies between homogeneous configuration and heterogeneous configuration given by force-directed heuristic for 5x5 network. Requests per process-1000

No. of Routers : 36
No. of cache slices : 36
No. of Requests per Process : 1000
Figure 4.9: Comparison of average latencies over four iterations of force-directed heuristic for 6x6 network. Requests per process-1000

Figure 4.10: Comparison of average latencies between homogeneous configuration and heterogeneous configuration given by force-directed heuristic for 6x6 network. Requests per process-1000
4.5.2 PARSEC Benchmarks

The PARSEC benchmarks were simulated under different core configurations in Simics. The cache access logs from Simics were pre-processed to extract the necessary information. In the experiments, the parameters such as number of cores, cache size etc remains consistent between Simics and the NoC simulation platform. The benchmarks were run under 16 cores, 32 cores and 128 cores in Simics and the results are provided below. Figure 4.12 shows the improvement in the average latency of the PARSEC benchmark which were run under 16 cores. Other configuration details are given below.

- No. of Routers : 16
- No. of cores : 16
- No. of caches : 16
- No. of Iterations : 5
- Max. No. of caches/ Router : 2

![PARSEC- 16 CORE](image)

Figure 4.11: Comparison of average latencies between homogeneous cache configuration and heterogeneous configuration after each iteration of the heuristic in PARSEC benchmarks simulated under 16 cores.
Figure 4.12: Comparison of average latencies between homogeneous cache configuration and force directed heterogeneous configuration of PARSEC benchmarks simulated under 16 cores.

<table>
<thead>
<tr>
<th>BENCHMARKS</th>
<th>EXECUTION TIMES (s)</th>
<th>NO. OF REQUESTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blackscholes</td>
<td>2146.25</td>
<td>1,290,03,50</td>
</tr>
<tr>
<td>Ferret</td>
<td>462.13</td>
<td>4,718,48</td>
</tr>
<tr>
<td>Fluidanimate</td>
<td>9516.56</td>
<td>4,567,7,239</td>
</tr>
<tr>
<td>Freqmine</td>
<td>1066.4</td>
<td>9,50,828</td>
</tr>
<tr>
<td>Streamcluster</td>
<td>3740.56</td>
<td>10,32,177</td>
</tr>
<tr>
<td>Swaptions</td>
<td>8877.81</td>
<td>5,41,96,848</td>
</tr>
<tr>
<td>Vips</td>
<td>1729.46</td>
<td>2,26,714</td>
</tr>
<tr>
<td>X264</td>
<td>8038.43</td>
<td>1,27,48,801</td>
</tr>
</tbody>
</table>

Table 4.1: Execution times and number of requests processed in PARSEC benchmarks simulated under 16 cores.
Table 4.2: Percentage Improvement in average latency of PARSEC benchmarks simulated under 16 cores.

Table 4.1 shows the number of requests processed in each benchmark and the execution time of the NoC simulation platform. The execution time includes the execution time of both that of the simulation of the NoC architecture to find the average latency and that of the heuristic. The run time of the heuristic is in microseconds and is negligible compared to the time taken for the simulation of the architecture. The table 4.2 shows the percentage improvement in the average latency for every benchmark.

Figure 4.14 compares the average latency of the PARSEC benchmarks simulated under 32/128 cores and homogeneous cache configuration with that of the heterogeneous cache configuration suggested by the force-directed heuristic. The configuration details are given below.

<table>
<thead>
<tr>
<th>BENCHMARKS</th>
<th>Improvement in average latency (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blackscholes</td>
<td>47.93</td>
</tr>
<tr>
<td>Ferret</td>
<td>47.18</td>
</tr>
<tr>
<td>Fluidanimate</td>
<td>55.27</td>
</tr>
<tr>
<td>Freqmine</td>
<td>66.90</td>
</tr>
<tr>
<td>Streamcluster</td>
<td>33.86</td>
</tr>
<tr>
<td>Swaptions</td>
<td>76.48</td>
</tr>
<tr>
<td>Vips</td>
<td>33.88</td>
</tr>
<tr>
<td>X264</td>
<td>41.27</td>
</tr>
</tbody>
</table>

Table 4.2: Percentage Improvement in average latency of PARSEC benchmarks simulated under 16 cores.

Table 4.1 shows the number of requests processed in each benchmark and the execution time of the NoC simulation platform. The execution time includes the execution time of both that of the simulation of the NoC architecture to find the average latency and that of the heuristic. The run time of the heuristic is in microseconds and is negligible compared to the time taken for the simulation of the architecture. The table 4.2 shows the percentage improvement in the average latency for every benchmark.

Figure 4.14 compares the average latency of the PARSEC benchmarks simulated under 32/128 cores and homogeneous cache configuration with that of the heterogeneous cache configuration suggested by the force-directed heuristic. The configuration details are given below.

No. of Routers : 16
No. of cores   : 32/128
No. of cores/Router : 2/8
Chapter 4: Heterogeneous Cache - Dynamic Architecture

No. of caches : 16
No. of Iterations : 5
Max. No. of caches/ Router : 2

Figure 4.13: Comparison of average latencies between homogeneous cache configuration and force directed heterogeneous configuration of PARSEC benchmarks simulated under 32/128 cores.

Figure 4.14: Comparison of average latencies between homogeneous cache configuration and force directed heterogeneous configuration of PARSEC benchmarks simulated under 32/128 cores.
Table 4.3 shows the number of requests processed in each benchmark and the execution time of the NoC simulation platform. Table 4.4 shows the percentage improvement in the average latency for every benchmark. It can be observed from the results that the force-directed heuristic gives a constructive way of improving the average latency by reducing the network hops.

<table>
<thead>
<tr>
<th>BENCHMARK</th>
<th>EXECUTION TIME (s)</th>
<th>NO.OF REQUESTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fluidanimate (32 cores)</td>
<td>86429.2</td>
<td>16,65,047</td>
</tr>
<tr>
<td>X264 (32 cores)</td>
<td>996512</td>
<td>16,33,545</td>
</tr>
<tr>
<td>Streamcluster (128 cores)</td>
<td>89889.1</td>
<td>84,20,672</td>
</tr>
<tr>
<td>X264 (128 cores)</td>
<td>207902</td>
<td>44,41,599</td>
</tr>
</tbody>
</table>

Table 4.3: Execution Times of the simulation and No. of Requests of PARSEC benchmarks simulated under 32/128 cores.

<table>
<thead>
<tr>
<th>BENCHMARK</th>
<th>Improvement in average latency (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fluidanimate (32 cores)</td>
<td>16.66</td>
</tr>
<tr>
<td>X264 (32 cores)</td>
<td>76.62</td>
</tr>
<tr>
<td>Streamcluster (128 cores)</td>
<td>41.04</td>
</tr>
<tr>
<td>X264 (128 cores)</td>
<td>61.49</td>
</tr>
</tbody>
</table>

Table 4.4: Percentage Improvement in average latency of PARSEC benchmarks simulated under 32/128 cores.
4.5.3 Observations

The details of the heterogeneous configuration generated for the PARSEC benchmark Streamcluster is explained here. Streamcluster was simulated in Simics under 128 cores to obtain the cache access patterns. Table 4.5 shows the access patterns. Each entry on the table indicates the number of requests sent from that router to the corresponding cache block. These serve as the weights in calculating the zero force target location of each cache block. Figure 4.15 shows the heterogeneous cache configuration obtained from the proposed heuristic. It can be seen from the heterogeneous configuration that the algorithm suggests that the most shared data to be placed in the cache blocks at the center. This will place the shared data lines, equidistant from the accessing cores. It can be observed that in the final heterogeneous cache configuration, large cache blocks are placed in the tiles associated with the cores executing data intensive (number of requests) processes or the cores for which data is distributed over several cache blocks. Thus if the data pattern is distributed across the cache for all the processes, homogeneous cache configuration would be the optimal cache configuration. It was also observed with the synthetic benchmarks.

Figure 4.15: Dynamic Heterogeneous Configuration obtained for Streamcluster
### Table 4.5: Access Patterns of PARSEC benchmark - Streamcluster under 4x4 mesh network

<table>
<thead>
<tr>
<th>Cache Block</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</th>
<th>R7</th>
<th>R8</th>
<th>R9</th>
<th>R10</th>
<th>R11</th>
<th>R12</th>
<th>R13</th>
<th>R14</th>
<th>R15</th>
<th>R16</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1 Routers</td>
<td>30242</td>
<td>32580</td>
<td>39568</td>
<td>40480</td>
<td>39427</td>
<td>37377</td>
<td>38341</td>
<td>40607</td>
<td>39734</td>
<td>42984</td>
<td>48151</td>
<td>40557</td>
<td>51754</td>
<td>43036</td>
<td>43344</td>
<td>48390</td>
</tr>
<tr>
<td>C2</td>
<td>160596</td>
<td>210198</td>
<td>360955</td>
<td>143260</td>
<td>149260</td>
<td>148965</td>
<td>149482</td>
<td>148229</td>
<td>149864</td>
<td>150079</td>
<td>150650</td>
<td>149916</td>
<td>152454</td>
<td>153211</td>
<td>153211</td>
<td>154318</td>
</tr>
<tr>
<td>C3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C16</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.5.4 Comparison between Random Search and Force directed heuristic

The results of the random search for a better heterogeneous cache configuration and that of the heuristic can be compared. Though the random search gives better configuration (configuration with lower average latency) for some benchmarks compared to the configuration suggested by the heuristic, the force directed placement of caches provides better configuration within fewer iterations and time.

Figure 4.16 compares the average latencies of the homogeneous configuration, the random search configuration and the dynamic heterogeneous configuration. Figure 4.17 compares the number of iterations; the random search takes to find a better configuration and the number of iterations the heuristic proposed takes to suggest the optimal configuration. It can be seen from the figure that the proposed force directed heuristic takes far less iterations in comparison to the random search and hence it is faster. Figure 4.18 compares the execution time of the NoC simulation platform in both the cases. Comparisons in the case of other mesh sizes are provided in the Appendix.

![5x5 Router Network](image_url)

Figure 4.16: Comparison of average latencies of different cache configurations: homogeneous, the one obtained through random search and the one suggested by force-directed heuristic in 5x5 network.
Figure 4.17: Comparison of iteration count for obtaining better cache configurations through random search and force-directed methods in 5x5 network.

Figure 4.18: Comparison of execution time of the code for obtaining better cache configurations through random search and force-directed methods in 5x5 network.
4.6 Hardware Requirements

In order to implement and realize the dynamic heterogeneous cache, we need an architecture that is flexible to map variable number of cache blocks to the routers. Since the optimal cache configuration and page mapping varies with each benchmark, we need a reconfigurable architecture which can change the number of cache blocks connected to each router in the NoC. The figure 4.19 shows a reconfigurable architecture for the proposed idea. The proposed architecture consists of a reconfigurable layer of cache blocks, which can be connected to the routers via switches. The number of cache blocks connected to each router can be changed according to the workload. The OS which maps the cache blocks to the routers should make sure that a cache block is connected to only one router at a time.

Given the workload at any instant, the OS can run the proposed heuristic for one iteration and configure the caches accordingly before the cores execute the process. The cost of the hardware and reconfiguration has not been explored in this work and has been proposed as a future work. Another way of doing the reconfiguration is to collect the history of workloads and finding the optimal configurations for each of them. If we know the kind of workloads or the benchmarks the system is going to handle, the heuristic can be used to find the optimal configuration for each workload. This information can be used by the OS to reconfigure the cache blocks when presented with the benchmark.

![Figure 4.19: A reconfigurable architecture for dynamic heterogeneous cache.](image)
Chapter 5

Conclusion and Future Work

5.1 Conclusion

The performance of the NoC architecture primarily depends on the data retrieval time from the cache. Several data migration and page coloring techniques have been proposed in literature to improve the data locality in non-uniform cache architecture. This work aims at further improving the data locality by exploring non uniform distribution of cache among the tiles in the NoC architecture. Since the workload in a multicore architecture consists of processes with different data intensities, flexible cache architecture is required to reduce the non uniform cache access latency. It has been shown that for the given workload in the NoC architecture, the homogeneous cache configuration need not be the optimal configuration and there might exist a heterogeneous cache configuration which gives lower cache access latency.

The static heterogeneous cache configuration has been proposed along with appropriate scheduling of processes. It has been shown that the static heterogeneous architecture is efficient in balancing processes with varying data intensities. Results show significant improvement in the average latency with the static heterogeneous cache configuration. The architecture proposed for 4x4, 5x5 and 6x6 networks can be extended to bigger network sizes easily with no additional hardware overhead. However, when the data intensities of processes are almost the same, homogeneous cache configuration will perform better.

The dynamic heterogeneous cache configuration has been suggested for multi-threaded workloads where the data is shared among a number of cores. Given the access patterns, the problem of initial placement of data has been addressed. The problem has been modeled as a placement problem and a constructive heuristic has been suggested to find the optimal amount of cache that each tile must contain. The force-directed heuristic attempts to minimize the number of hops across the network thereby minimizing the access latency. Since for each workload the cache configuration differs, a reconfigurable hardware has been proposed for dynamic
heterogeneous cache. The experiments have shown that the dynamic heterogeneous architecture outperforms the traditional homogeneous cache configuration.

5.2 Future Work

This work explored non-uniform distribution of cache among the routers in NoC architectures. Several parameters which affect the cache access time have been analyzed in this thesis. Different placements of larger and smaller cache blocks were analyzed for static heterogeneous cache configuration. However an extensive search would be beneficial in coming up with the optimal placement of various sized cache blocks. Also if the workload nature of the target architecture is known beforehand, the cache distribution can be further customized. A more advanced OS scheduling for static heterogeneous cache configuration would be beneficial and would make good use of heterogeneity. Runtime thread allocation and intelligent thread placement is also another area of research.

The dynamic heterogeneous cache presented in this thesis provides an optimal cache configuration with minimum number of network hops for each workload. Minimizing queue sizes and hops are two contradicting requirements which have to be balanced for obtaining an optimal configuration in terms of average latency. Though queue sizes contribute only a small amount to the average latency, research into faster networks and ports would be of tremendous benefit. The move function in the heuristic can also incorporate a weighted function which takes the rate of queue buildup into account. More can be explored in the area of reconfigurable hardware for dynamic heterogeneous cache. Incorporating the cost of hardware and reconfiguration into the heuristic would give a more accurate measure of improvement. Number of factors like reconfiguration time, maximum amount of cache per router, etc can be explored in detail.
Bibliography


doi: 10.1109/SOCC.2010.5784671
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5784671&isnumber=5784634

DOI=10.1023/B:SUPE.0000014800.27383.8f
http://dx.doi.org/10.1023/B:SUPE.0000014800.27383.8f

doi: 10.1109/RTCSA.2007.16

doi: 10.1109/TCS.1979.1084652

http://doi.acm.org/10.1145/103724.103725


doi: 10.1109/IISWC.2008.4636090
Chapter 6

Appendix

Figure 6.1: Average Latency in cycles for various percentages of Remote cache access in a hybrid network with 25 routers and 6 cores per router.

Figure 6.2: Average Latency in cycles for various rates of request generation in a hybrid network with 25 routers and 6 cores per router.
Figure 6.3: Average Latency in cycles for various percentages of Remote cache access in a hybrid network with 36 routers and 6 cores per router.

Figure 6.4: Average Latency in cycles for various percentages of Remote cache access in a hybrid network with 36 routers and 6 cores per router.
Figure 6.5: Comparison of average latencies between homogeneous and static heterogeneous configuration (5-4-3) in 4x4 Router network

Figure 6.6: Comparison of average latencies between homogeneous and static heterogeneous configuration (7-4-1) in 4x4 Router network
Figure 6.7: Comparison of average latencies between homogeneous and static heterogeneous configuration (5-4-3) in 5x5 Router network

Figure 6.8: Comparison of average latencies between homogeneous and static heterogeneous configuration (7-4-1) in 5x5 Router network
Figure 6.9: Comparison of average latencies between homogeneous and static heterogeneous configuration (5-4-3) in 6x6 Router network

Figure 6.10: Comparison of average latencies between homogeneous and static heterogeneous configuration (7-4-1) in 6x6 Router network
Figure 6.11: Comparison of average latencies over four iterations on synthetic benchmarks for 4x4 router- Requests per process-5000

Figure 6.12: Comparison of average latencies between homogeneous configuration and heterogeneous configuration given by force-directed heuristic for 4x4 network.- Requests per process-5000
Figure 6.13: Comparison of average latencies over four iterations on synthetic benchmarks for 5x5 router - Requests per process-5000

Figure 6.14: Comparison of average latencies between homogeneous configuration and heterogeneous configuration given by force-directed heuristic for 5x5 network.- Requests per process-5000
Figure 6.15: Comparison of average latencies over four iterations on synthetic benchmarks for 6x6 router- Requests per process-5000

Figure 6.16: Comparison of average latencies between homogeneous configuration and heterogeneous configuration given by force-directed heuristic for 6x6 network.- Requests per process-5000
Figure 6.17: Comparison of average latencies over four iterations on synthetic benchmarks for 4x4 router- Requests per process-10000

Figure 6.18: Comparison of average latencies between homogeneous configuration and heterogeneous configuration given by force-directed heuristic for 4x4 network.- Requests per process-10000
Figure 6.19: Comparison of average latencies over four iterations on synthetic benchmarks for 5x5 router- Requests per process-10000

Figure 6.20: Comparison of average latencies between homogeneous configuration and heterogeneous configuration given by force-directed heuristic for 5x5 network.- Requests per process-10000
Figure 6.21: Comparison of average latencies over four iterations on synthetic benchmarks for 6x6 router- Requests per process-10000

Figure 6.22: Comparison of average latencies between homogeneous configuration and heterogeneous configuration given by force-directed heuristic for 6x6 network.- Requests per process-10000
Figure 6.23: Comparison of average latencies of different cache configurations: homogeneous, the one obtained through random search and the one suggested by force-directed heuristic in 4x4 network.

Figure 6.24: Comparison of iteration count for obtaining better cache configurations through random search and force-directed methods in 4x4 network.
Figure 6.25: Comparison of execution time of the code for obtaining better cache configurations through random search and force-directed methods in 4x4 network.

Figure 6.26: Comparison of average latencies of different cache configurations: homogeneous, the one obtained through random search and the one suggested by force-directed heuristic in 6x6 network.
Figure 6.27: Comparison of iteration count for obtaining better cache configurations through random search and force-directed methods in 6x6 network.

Figure 6.28: Comparison of execution time of the code for obtaining better cache configurations through random search and force-directed methods in 6x6 network.