The Pennsylvania State University
The Graduate School
College of Engineering

LATENCY ANALYSIS of DATA MINING CODES

A Thesis in
Computer Science and Engineering
by
Ali Ozdemir

© 2014 Ali Ozdemir

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science
August 2014
The thesis of Ali Ozdemir was reviewed and approved* by the following:

Mahmut Taylan KANDEMIR
Professor of Computer Science and Engineering
Thesis Advisor

Chita R. DAS
Professor of Computer Science and Engineering

Lee Coraor
Professor of Computer Science and Engineering & Director of Academic Affairs

*Signatures are on file in the Graduate School
ABSTRACT

According to the requirements posed by recent developments in computing, we need better architectures and software. In order to achieve these goals, we evaluate these systems by running test jobs and simulations. In this study, we analyze the latency of data mining applications from NU-MINEBENCH Benchmark suite from Northwestern University.

We present the results for 32 out of order cores, 4x8 mesh architecture with 32 kB L1 instruction and data caches, and 32 L2 cache banks distributed over the network with each bank having 512 kB capacity. We run multiple multithreaded benchmarks simultaneously and collect the results of latencies between L1-L2, L2-Memory Controllers (MCs), Memory, MC-L2 and L2-L1 for continuous 100 million cycles and same latencies until the end of simulation.

We present the results in two graphs for each workloads. The first one, Type A, shows the latencies for consecutive 100 million cycles after fast forwarding 1.1 billion cycles. The second graph, Type B, shows the breakdown of the total latency of memory requests among the different memory hierarchies.
# TABLE of CONTENTS

List of Figures................................................................................................................. vi
List of Tables .................................................................................................................... ix
Acknowledgements........................................................................................................... x
Chapter 1 ........................................................................................................................... 1
  Introduction....................................................................................................................... 1
Chapter 2 ........................................................................................................................... 2
  2. Background ................................................................................................................ 2
    2.1 Interconnection Networks....................................................................................... 2
      2.1.1 Basic Descriptions for Interconnection Networks.......................................... 2
      2.1.2 Processor – Memory Interconnect................................................................. 4
      2.1.3 Basic Information about Network.................................................................. 4
    2.2 Memory Systems...................................................................................................... 5
Chapter 3 ........................................................................................................................... 10
  3.1 Data Mining Benchmarks and Simulator ................................................................. 10
Chapter 4 ........................................................................................................................... 17
  4.1 Results ....................................................................................................................... 17
    SVM & K-Means & ScalParC & Fuzzy........................................................................ 18
    SVM-RFE & Fuzzy & Apriori & HOP......................................................................... 20
    SVM-RFE & Fuzzy & Apriori & Utility....................................................................... 21
    SVM-RFE & Fuzzy & Apriori & PLSA........................................................................ 22
    SVM-RFE & Fuzzy & Utility & PLSA......................................................................... 23
    SVM-RFE & ScalParC & Apriori & HOP..................................................................... 24
    SVM-RFE & ScalParC & Apriori & Utility.................................................................. 25
    SVM-RFE & ScalParC & HOP & Utility ..................................................................... 26
    SVM-RFE & ScalParC & Utility & PLSA.................................................................... 27
    SVM-RFE & K-Means & Apriori & HOP..................................................................... 28
    SVM-RFE & K-Means & Apriori & Utility.................................................................. 29
    SVM-RFE & KMeans & Apriori & PLSA..................................................................... 30
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MineBench Benchmarks</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>NoC Architecture</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>MPKI Values</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>Type A SVM &amp; K-Means &amp; ScalParC &amp; Fuzzy</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>Type A Apriori &amp; Hop &amp; Utility &amp; PLSA</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>Type A SVM-RFE &amp; Fuzzy &amp; Apriori &amp; HOP</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>Type A SVM-RFE &amp; Fuzzy &amp; Apriori &amp; Utility</td>
<td>21</td>
</tr>
<tr>
<td>8</td>
<td>Type A SVM-RFE &amp; Fuzzy &amp; Apriori &amp; PLSA</td>
<td>22</td>
</tr>
<tr>
<td>9</td>
<td>Type A SVM-RFE &amp; Fuzzy &amp; Utility &amp; PLSA</td>
<td>23</td>
</tr>
<tr>
<td>10</td>
<td>Type A SVM-RFE &amp; ScalParC &amp; Apriori &amp; HOP</td>
<td>24</td>
</tr>
<tr>
<td>11</td>
<td>Type A SVM-RFE &amp; ScalParC &amp; Apriori &amp; Utility</td>
<td>25</td>
</tr>
<tr>
<td>12</td>
<td>Type A SVM-RFE &amp; ScalParC &amp; HOP &amp; Utility</td>
<td>26</td>
</tr>
<tr>
<td>13</td>
<td>Type A SVM-RFE &amp; ScalParC &amp; Utility &amp; PLSA</td>
<td>27</td>
</tr>
<tr>
<td>14</td>
<td>Type A SVM-RFE &amp; K-Means &amp; Apriori &amp; HOP</td>
<td>28</td>
</tr>
<tr>
<td>15</td>
<td>Type A SVM-RFE &amp; K-Means &amp; Apriori &amp; Utility</td>
<td>29</td>
</tr>
<tr>
<td>16</td>
<td>Type A SVM-RFE &amp; K-Means &amp; Apriori &amp; PLSA</td>
<td>30</td>
</tr>
</tbody>
</table>
Figure 32 Type B SVM-RFE & Utility ......................................................... 46
Figure 33 Type A SVM-RFE & PLSA ......................................................... 47
Figure 33 Type B SVM-RFE & PLSA ......................................................... 47
Figure 34 Type A Fuzzy& Apriori ............................................................. 48
Figure 34 Type B Fuzzy& Apriori ............................................................. 48
Figure 35 Type A Fuzzy & HOP ............................................................... 49
Figure 35 Type B Fuzzy & HOP ............................................................... 49
Figure 36 Type A Fuzzy & Utility ............................................................. 50
Figure 36 Type B Fuzzy & Utility ............................................................. 50
Figure 37 Type A Fuzzy & PLSA ............................................................... 51
Figure 37 Type B Fuzzy & PLSA ............................................................... 51
Figure 38 Type A ScalParC & Apriori ....................................................... 52
Figure 38 Type B ScalParC & Apriori ....................................................... 52
Figure 39 Type A ScalParC & Utility ....................................................... 53
Figure 39 Type B ScalParC & Utility ....................................................... 53
Figure 40 Type A K-Means & Apriori ...................................................... 54
Figure 40 Type B K-Means & Apriori ...................................................... 54
Figure 41 Type A K-Means & HOP .......................................................... 55
Figure 41 Type B K-Means & HOP .......................................................... 55
Figure 42 Type A K-Means & Utility ...................................................... 56
Figure 42 Type B K-Means & Utility ...................................................... 56
Figure 43 Type A ScalParC & K-Means .................................................... 57
Figure 43 Type B ScalParC & K-Means .................................................... 57
Figure 44 Type A Fuzzy & K-Means ....................................................... 58
Figure 44 Type B Fuzzy & K-Means ....................................................... 58
List of Tables

Table 1 Benchmarks Explanation.................................................................11
Table 2 L1 & L2 Miss Rates & IPC & Inst. Count.............................................12
Table 3 Baseline Architecture.................................................................16
Acknowledgements

First of all, I want to thank my advisor Prof. Dr. Mahmut Kandemir for his guidance, patience and support. I really appreciate that he has always been very helpful during my studies.

I also thank to my lab friends. They always helped me whenever I need them and they gave me very good suggestions to make everything better.

I am very thankful to my family for their support. They are extremely important to me and I would not be the person I am today without the love and support they have given me. This works is dedicated to my father, Salih Ozdemir, mother, Fatma Ozdemir, brother, Emrecan Ozdemir, fiancée, Nihal Cam.

Finally I want to thank to Turkish Government, because of the opportunity they provided me to do masters.

Ali Ozdemir

July, 2014
Chapter 1

Introduction

Digital systems are being employed everywhere in our modern world. We use these systems from very small scale to gigantic scales, such as editing word documents in data centers. A digital system has three important components: logic, memory and interconnect [1]. Logic performs arithmetic operations. Memory stores data and provides it to the logic when a request arrives. The communication moves data from one place or component to other, such as from processor to memory. Digital systems are developing very fast because of the growing application and user demands. The problem due to this rapid advancement is that the bandwidth and latency of memory and network do not improve at the same rate. At the same amount of time, we are able to improve bandwidth by 2 times, whereas latency by only 1.2 times [2].

From the software side, data mining is becoming very popular in the last few decades and it has been highly used by the companies for marketing, business, bio-technology and internet services, but the data that we collect or our input data increase much faster than our systems that we use to examine this data.

Because of the current trends in both architecture and hardware aspects, in this study, we analyze latencies observed in data mining applications. We use GEM5 simulator [3] and NU-MINEBENCH benchmarks [4] in our evaluations.
Chapter 2

2. Background

2.1 Interconnection Networks

In this section, we describe the basics of interconnection networks, and answer the following questions: what is an interconnection, where do we use them, are they really important for digital systems?

First, interconnection network transports data from one point to other point in systems. We call these points as terminals. Interconnection networks consist of different components like buffers, channels, switches, and controls.

Second, we use interconnection networks in all digital systems which have at least two components to communicate with each other. The most common usage of interconnection networks is in computer systems. By using them, we can connect processors to memories, I/O devices to their controllers, input ports to output ports and etc.

To answer the last question, interconnection networks are very important in digital systems. For instance, they are the performance limiting factor in many computing systems. They determine the achievable latency and the bandwidth, which are very important for these systems.

2.1.1 Basic Descriptions for Interconnection Networks

In this section, we describe some key words that we use in the context of interconnection networks. We start with the terminals or ports. Our number of components, which need communication with other components should be connected to the network. Also circuit designer needs to know which port will communicate with which port.
The connections between terminals require some amount of bandwidth from the network, and this bandwidth is usually measured in bits per sec (bit/s). If nothing is emphasized specifically, it is assumed that each terminal will require the same amount of bandwidth from the network.

Even though we assume that terminals require the same bandwidth, it is highly expected that some terminals will require more bandwidth at some point during the course of execution, leading to peaks in bandwidth demands. Peak bandwidth is the maximum data rate that a terminal demands from the network for a period of time. The average bandwidth is similar with peak bandwidth in terms of description, but this time we measure the average data rate that is requested.

The time that is needed for delivering messages between two terminals is called latency. Ideal network supports high bandwidth and provides low latency. But there is a tradeoff between these two concepts. In order to have high bandwidth, system should keep resources busy. This leads to contention, because contention happens when two or more messages want to use the same resources. In such a scenario, only one of the messages is granted service and the other one should wait. This increases the latency.

We expect that the message size would also be important for network performance. We define message size as the length of the message in bits. There can be many different messages in terms of size in digital systems.

The transportation and distribution of these small or big messages from one port to another port defines the traffic pattern which is one of the important factors on performance. One port could send its messages to all other ports with equal probability which is called random traffic pattern, or a port could have high probability to send its messages to one specific port. If there is a case like this, we take advantage of this situation and exploit this locality to reduce access cost.

One of the most used terms for interconnection networks is Quality of Services (QoS). Quality of Services is fair distribution of services under some service policy. For instance, more than one message could compete for one of the resources on the system. In order to have system
with fairness, we employ scheduling policies and give priority to the message which came last
/first, or we can give priority to the message which spent more time than the others on the network.

Reliability is the criterion that shows how many of our request successfully completed by
the network. %100 reliability can be achievable with adding more hardware for error correction
and detection and higher level software support.

2.1.2 Processor – Memory Interconnect

In this section, we give an idea about how processor and memory communicate through
the interconnection network. Today, we have very complicated and very powerful processors.
These processors execute trillions of instructions per second and many of these instructions require
data from memory. This great amount data request from memory and interconnection should be
able to handle all these requests. For example, imagine that we have cache miss for load or store
instruction.

We mentioned earlier that our message sizes could be different from each other. Even
though our message sizes could be different, we can divide them into fixed length packets. These
packets are generally in two different formats. Read Request/Write Reply is one kind and Read
Reply/Write Request is the other one. First type does not have any data. It only has header and
address. Second type has data with it. That’s why it is much bigger than the first one. It has a header,
an address and 512 bits of data.

2.1.3 Basic Information about Network

In section 2.1.2 we explained that the communication between memory and processor is
significant. Our network should handle requests from multiple core and sustain good performance.
At this point, our topology, routing and flow control has great importance in our network. We will
describe these concepts below.
“The interconnection network consists of shared router nodes, which is connected by shared channels. Network topology determines the way of these nodes are connected [1].

The interconnection network is implemented between terminals by making several hops across the shared channels and nodes from its source to destination. We have many different options to deliver message from requester to destination. Routing determines which of these possible ways the request takes. Routing helps us to deliver packets as fast possible and also balance the system load across routers. If one resource of the system is highly utilized and the other sits idle, it means that the system is not balanced and it is not good for the system, because it reduces the bandwidth.

Finally, flow control determines which messages get access to particular network resources. This becomes important when utilization increases, and fairness is an issue.

2.2 Memory Systems

In this section, we briefly discuss the memory systems in computers and how they work.

As the data is stored in the memory, it is sent whenever a request is received from the processor. Instruction execution at the processor might require data, therefore the processor has to stall for the data to arrive. This affects the performance and thus we need the data to be available as soon as possible for the processor. In order to achieve this, computer scientists developed the idea of memory hierarchies. “Ideally one would desire an indefinitely large memory capacity such that any particular word (2 Bytes of data, 16 bits of data) would be immediately available. We are forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible. [5]”

To give better understanding for memory hierarchy, we explain how it works by giving the library analogy. Suppose that you need to write a paper about the history of memory systems in computers. In order to write this paper, you went to the library and asked for a book. For simplicity, we ask for one book at a time. To bring this book, the librarian should take the name of
the book and go to the storeroom then search for the book, find it and bring it. After you are done with the book, you give it back to the librarian and he will go back to storeroom and put it back in its place. But if we ask for the same book again, it is highly expected that the book that we used will be used soon again. And also it is expected that we could ask for a book which is nearby the book that we asked before, because they tend to be related with each other. Now for our each request, librarian should go all the way and come back. This is a very slow process. We can increase the speed of this process by using a different technique. Suppose that our librarian has a small shelf near him, which he uses it to put books which are brought back by the requesters. Now let’s start our process again. We went to the library and we asked for a book. The librarian went to storeroom and brought it. We read our book and gave it back. Then we asked for the same book again. This time the librarian did not go to storeroom, because when we gave the book back, he put it into the small shelf. It is there and he does not need to go the storeroom again. He can take the book from the small shelf and give us. This time, we have our book much faster. But what if we want a book that is not present in the small shelf? This time, the librarian will first check the small shelf, and then he will go to storeroom and bring it. Even though he spent extra time for checking the small shelf, that time is so small when we compare it with the whole way to the storeroom. Let’s apply this example to our memory system.

In the course of executions of programs, we do not need all the data that reside in memory, like we do not need all the books in the library and programs do not access its code or data with the same probability. Programs tend to access its code or data like in our library example; they generally ask for the same data or nearby data. This creates a concept of locality. This means that programs access very small part of its address space at a time. If we think our library example, we will realize that there are two different localities. First one is temporal locality, which means that the item, data, or book that is referenced, is likely to be referenced again. Second one is spatial locality, which means that the item, data, or book that is referenced before has a nearby item, data,
or book that is likely to be referenced soon. We can call them as a locality in time and locality in space, respectively.

In most of the programs that we use, we have loops. Because of the characteristics of loops, we tend to access the same data again and again or accessing the elements of arrays sequentially allows us to have spatial locality, because we access nearby data. Caches or memories are designed and implemented to take advantage of these scenarios. Engineers developed memory hierarchy idea to achieve this. It consists of levels of memories which are getting smaller, faster and more expensive when they get closer to core and vice versa. We call the closest cache to the CPU as an L1 cache, then we have L2 cache which is larger than L1 cache but slower, generally because of its bigger capacity. Then we have DRAM which has much larger capacity than L1 and L2 and cheaper but slower compared to them. Beyond DRAM, we have disk, which has even larger capacity and is very cheap in terms of cost per bit. We keep all our data inside of it. It is non-volatile unlike L1, L2 and DRAM. Also, another memory technology which is called SSD is becoming more and more common between DRAM and magnetic disk. It is also non-volatile and it allows good performance, because it is much faster than magnetic disks.

From now on, we will describe what is hit, miss, hit rate, miss rate, hit time and miss penalty. Then, we will explain how we handle misses and writes, because these concepts are related to our point of interests. After that we will end our discussion on caches.

When the data that is asked for by the processor is present in the upper level memory, L1 cache, we call this as a hit. If it is not there, then we call it as a miss. Through the execution of a program, we have many hits and misses. The proportion of number of hits to the number of requests gives us the hit rate or we can call it as a hit ratio [5]. Hit rate is generally used as a performance measure of memory hierarchy. Miss rate is the proportion of the number misses to the number of requests gives us the miss rate or we can calculate it simply 1- (hit rate).

The biggest reason for usage of memory hierarchy is the aim of increasing performance. By using memory hierarchy, we want to get our data as soon as possible. The time that we spent to
get our data is the collection of hit time and miss penalty, if we have miss in upper level. If we find our data in L1 then our waiting time is directly equal to hit time. Now let’s explain the hit time and miss penalty. Hit time is the time that we spend to access memory and required time to understand whether requested data is there or not. Miss penalty consists of the time required to fetch a block into a level of the memory hierarchy from the lower level, including the time to access the block, transmit it from one level to the other, insert it in the level that experienced the miss, and then pass the block to the requestor.

After talked about basic concepts, finally, we explain how in order processors handle with cache misses and handle with writes, and then we will end our basic memory discussion here and will start to talk about more specific topics about this thesis like simulator and benchmarks that we used in this work.

When the processor requests data and a miss occurs, there is a procedure to take care of this miss and there is a control unit which is responsible for doing it. Misses are handled with the collaborations of processor control unit and with a separate controller which starts the memory access and refills the cache.

In here we will explain how in order processors handle with instruction misses.

When we get a miss from an instruction access request, this means that the content of this instruction register are not there. We have to bring requested data from lower level of memory hierarchy. We can instruct the lower level of memory to find the data that we are looking for and it makes a read for the data. When it finds our requested data, it will send a response message with the data. After data arrives, we can restart the execution of instruction.

We have different scenario for writes. When we get the result and want to put it in the cache, we have a different problem. The data is changed is not consistent with the data in lower levels. That’s why the data in the lower levels also need to be updated. The easiest way to solve this problem is a scheme that is called write-through. This scheme offers to write data into the cache and memory at the same time. The other thing that we need to concern is that what happens if we
have a write miss. In this case, we first bring the words from memory and put it into its place in the cache. Then we are able to complete our write on cache but we need to change the data at memory. We can complete it with using the complete address of the word. Although this is a very simple way to handle the problem, it does not increase performance, because every memory write takes significant time and as a result of this problem, performance decreases.

In order to solve this performance problem, we use write buffers. Write buffer simply takes care of the write to memory. When we write data to the cache, we write it to the write buffer also and then keep executing new instructions. At the same time write buffer does memory writes and then when it completes the write, data is released from write buffer and opens space for the new writes. But if there is no available space in write buffer and if there is another write request, the processor stalls until there is space in the write buffer.

Rather than using write through policy, we can employ another policy which is called write back. This scheme is more complex when compared to write through. When there is a write, the data is only written into the cache. After that, the changed block is written to lower level. In this scheme, we do not write to memory and since memory writes take significant time, we avoid these latencies. Because we spend less time for writes, we can handle more concurrent write requests.
Chapter 3

3.1 Data Mining Benchmarks and Simulator

Data mining is the process that lets us analyze large amount of data from different point of views and getting useful information. This technique is highly used for marketing, business, biotechnology and internet searches. Companies use it also to guess customer needs and future products. Because of the companies which need this type information, data mining is becoming a very important field. Day by day, the information that we collect or our input data is increasing, but our computing systems are not developing at the same rate. In order to solve this problem, we need to understand bottlenecks of the systems. We use data mining benchmarks to analyze and test current and possible future systems.

The requirements of companies about data mining systems make data mining applications very important. In our study, we use MineBench benchmarks [4], which is a benchmark suite for data mining workloads and this suite has applications from clustering classification, association rule mining and optimization.

![Figure 1 MineBench Benchmarks](image-url)
In this study we used, SVM-RFE [6] which is stands for Support Vector Machines Recursive Feature Elimination and it used for disease finding with using the recursive feature elimination method. **K-means** [7] is a clustering algorithm which takes a parameter k and then the algorithm creates k clusters for a given data set with using distance function. **Fuzzy K-means** [8] does the same thing, but assumes that the data object we have in our dataset could have some degree of membership in clusters. Fuzzy K-means has more computation cost when we compare it with K-means. **HOP** [9] is another clustering method which creates clusters according to their densities. After clusters are created, it associates particles with the closest densest point. **ScalParC** [10] uses decision tree classification method. It splits the dataset until every data that we provide has a class. **Apriori** [11] looks for all the subsets of an input and finds out level wise mining of the property. **Utility mining** [12] is another ARM mining technique which looks for higher utility portions of the data that we provided according to its algorithm. **PLSA** [13] handles with the sequence matching problem with the using Smith and Waterman algorithm.
We have different types of data mining applications which try to solve discrete problems like classification, clustering, Association Rule Mining and optimization [15]. Classification has a training set which has example of records with a number of attributes. Classification algorithm uses this training set to put unclassified data into one of the defined class [6]. The other type of problem is clustering which is a kind of application that tries to find out groups of similar objects from the dataset [6]. Association Rule Mining or ARM aims to find relationships between the subsets of the input data. Also it looks for how a given subset of items influence the other subsets [6]. Optimization is used for sequence alignment. It is crucial for bioinformatics to align DNA, RNA and protein sequences in order to find out their similar portions [6].
We use GEM5 simulator in our studies. GEM5 has a model for network and it implements various cache coherence protocols. We use GEM5 ruby with garnet network. This led us to analyze network in detail and track all request messages and response messages in terms of their latencies. With the ability to deeply observe network, we can track network latencies in our workloads. We edited NetworkInterface_d.cc file in GEM5 ruby for our implementation. All memory requests are converted into flits “flitsize_message()”. We observe all the request and response messages that passes through the network from here and also all data request and response messages which passes from this function carries address information, whether it is request message or response message, current clock cycle, which core made that request, whether it is read or write request etc with them. We took these information and put them into the table which keeps all these information. After that we calculated the latency between stages which are L1 cache to L2 cache, L2 cache to Memory Controller, Memory Controller to Memory Controller (response) (we get the latencies request and response as off chip memory latency that’s why we indicate it as a memory controller to memory controller), Memory Controller to L2 cache and L2 cache to L1 cache.
We get all this information when the response comes back to L1 and makes the necessary calculations and we collect the results according to our experimental setup. We have 32 out of order cores, which are running at 2GHz frequency, and each core has its own L1 Instruction Cache and L1 Data Cache. Both these caches have 32 kB capacity with 3 cycle access latency. We have 32 L2 cache banks each having 512 kB capacity, with 15 cycle access latency and they are distributed over the network.
Figure 3 MPKI Values

In Figure 3, we give the normalized MPKI (Misses per Kilo Instructions) values according to ScalParC for the benchmarks we use. From this figure, we observe that SVM-RFE has by far the highest MPKI. For Hop, Apriori, Utility, Fuzzy we have mediocre MPKI. For K-Means, PLSA and ScalParC we have very low MPKI values.

For cache coherence protocol, we use MESI_CMP_directory (M: Modified, E: Exclusive, S: Shared, I: Invalid). GEM5 also supports different cache coherence protocols. Our network topology is 4x8 Mesh architecture and we have 4 memory controllers at the corners. We present our experimental setups as a table below.
<table>
<thead>
<tr>
<th>Table 3 Baseline Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
</tr>
<tr>
<td>NoC Architecture</td>
</tr>
<tr>
<td>Private L1 D&amp;I Caches</td>
</tr>
<tr>
<td>Shared L2 Caches</td>
</tr>
<tr>
<td>Cache Coherence Protocol</td>
</tr>
<tr>
<td>Memory</td>
</tr>
<tr>
<td>Simulator</td>
</tr>
<tr>
<td>Clock Cycle</td>
</tr>
</tbody>
</table>
Chapter 4

4.1 Results

We now present the experimental results collected using various workloads. In executing each workload, the simulation was fast forwarded 1.1 billion cycles. For the Figure type A the statistics are collected in the next 100 million cycles and for the figure type B the statistics are collected until the end of one benchmark. Results are for a 32-core system with the baseline configuration: as mentioned earlier, in this configuration, 32 cores are connected by a 4×8 2D mesh-based NoC and 4 memory controllers are placed in four corners of the mesh. Figure Type A are the cumulative graph of all the latencies between L1-L2, L2-MC, Mem, MC-L2 and L2-L1. Figure Type B, shows the breakdown of the latencies among the various memory hierarchies with respect to total latency. We bin the total latency into different ranges varying from 150 cycles to 700 cycles such as 150-200, 200-250, 250-300, 300-350, 350-400, 450-500 and 550+. We first give the results of simulations in which we run 4 multithreaded benchmarks at the same time. Then, we give simulation results which belong to two multithreaded benchmarks which are running together.
SVM & K-Means & ScalParC & Fuzzy

Figure 4 Type A SVM & K-Means & ScalParC & Fuzzy

Figure 4 Type B SVM & K-Means & ScalParC & Fuzzy
Apriori & Hop & Utility & PLSA

Figure 5 Type A Apriori & Hop & Utility & PLSA

Figure 5 Type B Apriori & Hop & Utility & PLSA
SVM-RFE & Fuzzy & Apriori & HOP

Figure 6 Type A SVM-RFE & Fuzzy & Apriori & HOP

Figure 6 Type B SVM-RFE & Fuzzy & Apriori & HOP
SVM-RFE & Fuzzy & Apriori & Utility

Figure 7 Type A SVM-RFE & Fuzzy & Apriori & Utility

Figure 7 Type B SVM-RFE & Fuzzy & Apriori & Utility
SVM-RFE & Fuzzy & Apriori & PLSA

Figure 8 Type A SVM-RFE & Fuzzy & Apriori & PLSA

Figure 8 Type B SVM-RFE & Fuzzy & Apriori & PLSA
SVM-RFE & Fuzzy & Utility & PLSA

Figure 9 Type A SVM-RFE & Fuzzy & Utility & PLSA

Figure 9 Type B SVM-RFE & Fuzzy & Utility & PLSA
SVM-RFE & ScalParC & Apriori & HOP

Figure 10 Type A SVM-RFE & ScalParC & Apriori & HOP

Figure 10 Type B SVM-RFE & ScalParC & Apriori & HOP
SVM-RFE & ScalParC & Apriori & Utility

Figure 11 Type A SVM-RFE & ScalParC & Apriori & Utility

Figure 11 Type B SVM-RFE & ScalParC & Apriori & Utility
SVM-RFE & ScalParC & HOP & Utility

Figure 12 Type A SVM-RFE & ScalParC & HOP & Utility

Figure 12 Type B SVM-RFE & ScalParC & HOP & Utility
SVM-RFE & ScalParC & Utility & PLSA

Figure 13 Type A SVM-RFE & ScalParC & Utility & PLSA

Figure 13 Type B SVM-RFE & ScalParC & Utility & PLSA
SVM-RFE & K-Means & Apriori & HOP

Figure 14 Type A SVM-RFE & K-Means & Apriori & HOP

Figure 14 Type B SVM-RFE & K-Means & Apriori & HOP
SVM-RFE & K-Means & Apriori & Utility

Figure 15 Type A SVM-RFE & K-Means & Apriori & Utility

Figure 15 Type B SVM-RFE & K-Means & Apriori & Utility
SVM-RFE & KMeans & Apriori & PLSA

Figure 16 Type A SVM-RFE & K-Means & Apriori & PLSA

Figure 16 Type B SVM-RFE & K-Means & Apriori & PLSA
Figure 17 Type A SVM-RFE & K-Means & HOP & Utility

Figure 17 Type B SVM-RFE & K-Means & HOP & Utility
SVM-RFE & K-Means & HOP & PLSA

Figure 18 Type A SVM-RFE & K-Means & HOP & PLSA

Figure 18 Type B SVM-RFE & K-Means & HOP & PLSA
SVM-RFE & K-Means & Utility & PLSA

Figure 19 Type A SVM-RFE & K-Means & Utility & PLSA

Figure 19 Type B SVM-RFE & K-Means & Utility & PLSA
Figure 20 Type A Fuzzy & ScalParC & Apriori & PLSA

Figure 20 Type B Fuzzy & ScalParC & Apriori & PLSA
Fuzzy & ScalParC & HOP & Utility

Figure 21 Type A Fuzzy & ScalParC & HOP & Utility

Figure 21 Type B Fuzzy & ScalParC & HOP & Utility
Figure 22 Type A Fuzzy & ScalParC & HOP & PLSA

Figure 22 Type B Fuzzy & ScalParC & HOP & PLSA
Fuzzy & ScalParC & Utility & PLSA

Figure 23 Type A Fuzzy & ScalParC & Utility & PLSA

Figure 23 Type B Fuzzy & ScalParC & Utility & PLSA
Fuzzy & K-Means & Apriori & HOP

Figure 24 Type A Fuzzy & K-Means & Apriori & HOP

Figure 24 Type B Fuzzy & K-Means & Apriori & HOP
Fuzzy & K-Means & HOP & Utility

Figure 25 Type A Fuzzy & K-Means & HOP & Utility

Figure 25 Type B Fuzzy & K-Means & HOP & Utility
ScalParC & K-Means & Apriori & HOP

Figure 26 Type A ScalParC & K-Means & Apriori & HOP

Figure 26 Type B ScalParC & K-Means & Apriori & HOP
ScalParC & K-Means & Apriori & Utility

Figure 27 Type A ScalParC & K-Means & Apriori & Utility

Figure 27 Type B ScalParC & K-Means & Apriori & Utility
ScalParC & K-Means & HOP & Utility

Figure 28 Type A ScalParC & K-Means & HOP & Utility

Figure 28 Type B ScalParC & K-Means & HOP & Utility
ScalParC & K-Means & Utility & PLSA

Figure 29 Type A ScalParC & K-Means & Utility & PLSA

Figure 29 Type B ScalParC & K-Means & Utility & PLSA
SVM-RFE & Apriori

Figure 30 Type A SVM-RFE & Apriori

Figure 30 Type B SVM-RFE & Apriori
SVM-RFE & HOP

Figure 31 Type A SVM-RFE & HOP

Figure 31 Type B SVM-RFE & HOP
SVM-RFE & Utility

Figure 32 Type A SVM-RFE & Utility

Figure 32 Type B SVM-RFE & Utility
SVM-RFE & PLSA

Figure 33 Type A SVM-RFE & PLSA

Figure 33 Type B SVM-RFE & PLSA
Figure 34 Type A Fuzzy & Apriori

Figure 34 Type B Fuzzy & Apriori
Figure 35 Type A Fuzzy & HOP

Figure 35 Type B Fuzzy & HOP
Figure 36 Type A Fuzzy & Utility

Figure 36 Type B Fuzzy & Utility
Fuzzy & PLSA

Figure 37 Type A Fuzzy & PLSA

Figure 37 Type B Fuzzy & PLSA
ScalParC & Apriori

Figure 38 Type A ScalParC & Apriori

Figure 38 Type B ScalParC & Apriori
ScalParC & Utility

Figure 39 Type A ScalParC & Utility

Figure 39 Type B ScalParC & Utility
K-Means & Apriori

Figure 40 Type A K-Means & Apriori

Figure 40 Type B K-Means & Apriori
K-Means & HOP

Figure 41 Type A K-Means & HOP

Figure 41 Type B K-Means & HOP
K-Means & Utility

Figure 42 Type A K-Means & Utility

Figure 42 Type B K-Means & Utility
ScalParC & K-Means

Figure 43 Type A ScalParC & K-Means

Figure 43 Type B ScalParC & K-Means
Fuzzy & K-Means

Figure 44 Type A Fuzzy & K-Means

Figure 44 Type B Fuzzy & K-Means
According to our results, we see that our latencies are mostly dominated by the memory latency. Between L1 to L2 and L2 to MC, we do not have too much latency in general. For the benchmarks with relatively high MPKI, we have higher latencies in terms of total latency. Sometimes we see high latency average for benchmarks that have low MPKI. For example, benchmark combinations which consist of benchmarks with low MPKI also have average latencies between 500-550 cycles.

Data mining benchmark combinations that we use in this study, we did not observe much NoC traffic. Big portion of our latencies related with memory. This means our benchmark does not have high MPKI values [13] and this leads less traffic in NoC. Our 32 core architecture able to handle with small amount of traffic and did not present any eye catching or interesting latency graph.
Chapter 5

5.1 Conclusion

We analyzed end to end memory access latencies of data mining applications from Nu-MineBench Benchmark suite. We run our workloads on 32 out of order core 4x8 mesh architecture simultaneously and collect statistics. We see that there is not much traffic in NoC. That’s why our statistics show that memory latency is the biggest portion of latencies.
Bibliography


