JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

ANALYZING THE PERFORMANCE OF PROCESSOR ARCHITECTURE ON HETEROGENEOUS MEMORY SYSTEMS FOR LINEAR PROGRAMMING

[1] **Md. Shakil Hussain**
[1]Lecturer
[1] Dept. of Electrical & Electronic Engineering
[1] Pundra University of Science & Technology, Bogura, Bangladesh

Abstract : Present microprocessors have hierarchical subsystems of memory with several accessibility levels such as L1/L2/L3 caches. These memory hierarchy has been introduced to boost processor's performance. In literature review it is seen that, performance of heterogeneous memory systems is observed in terms of cache and memory latency and bandwidth efficiency in [1][2][3][4] with STREAM benchmarking. But STREAM benchmarking do not cover all aspects of multicore architectures such as bandwidth and latency between different processor cores because STREAM benchmarking only deals with linear memory access, but real applications use a particular memory region several times. So it can't show actual bandwidth of the memory hierarchy.

In this research work analytical study has been conducted on Intel core-i3, core-i5 and core-i7 architecture's performance for linear programming. Test programs like matrix transpose, matrix multiplication and matrix inversion have been devised to assess the performance characteristics for transferring data between different level of caches and from cache to main memory. These operations are performed in a particular memory block for several times. Various matrix sizes have been taken to different memory blocks to execute these operations, and performance has been evaluated based on the system clock. This will help us to gain substantial insight of the processor's memory performance for linear programming and help us find out the exact bottleneck to achieve efficient use of the memory hierarchy.

***keywords* – Linear Programming, Processor's memory performance, Matrix, Benchmarking, Memory hierarchy.**

I. Introduction

Memory access performance is a key factor of overall system performance. Memory performance is paramount to overall system performance. When a CPU is not paired appropriately with suitable memory, it can lead to underutilization of the CPU, resulting in decreased efficiency and subpar overall performance. Two crucial aspects of performance determine how a computer system retrieves data: latency and bandwidth. These factors play significant roles in system performance and data manipulation [5]. Latency is an important factor in determining maximum transfer rates for small files; however, it is not very significant for big file downloads. Latency means the time it takes to establish a connection between two points. Bandwidth means how rapidly data can be transmitted between two connected systems. Bandwidth is the most essential factor when moving large files that are a one-way communication. Once the transmission has started, the bandwidth becomes the more important factor. So latency and bandwidth play a vital role in system performance.

Several factors influence latency and bandwidth in computer systems. Intel architecture employs a hierarchical memory structure, with instruction and data caches residing at the topmost level. The instruction and data caches, typically denoted as Level 1 (L1) Cache, are small and located closest to the core. A larger cache, known as Level 2 (L2) cache, with slower access times compared to L1 cache, is also present. Additionally, there is a Level 3 (L3) cache shared among cores within the same package. Beyond the cache hierarchy lies system memory, or DRAM, situated outside the processor and accessed via the Front Side Bus and memory controller hub. System memory typically incurs higher latency than local caches due to its external location.

CPU caches store a small amount of memory directly on the CPU, operating at the CPU's speed rather than the system bus speed. This design aims to expedite data access, leveraging the likelihood of data reuse. The cache's proximity to the CPU enhances data accessibility, thus impacting latency and bandwidth [6].

This study introduces a set of benchmarks designed to assess the performance characteristics of memory accesses, including cache and memory latency and bandwidth efficiency for various memory access patterns. It encompasses transfers between on-chip caches, cache to main memory, and off-chip cache to cache, addressing their growing significance in contemporary computing environments.

II. RELATED WORK

Robert Schöne, Wolfgang E. Nagel, and Stefan Pfluger assess bandwidth using the STREAM benchmark [2]. This benchmark comprises four distinct components, each measured independently. Each component executes a vector operation on double-precision floating-point data. These components include copy, scale, add, and triad.

Jenifer Hopper has discussed an example benchmark called lat_mem_rd, from the lmbench3 suite in [3]. lat_mem_rd is often used in the community to measure memory read latency across a variety of systems.

Alexander Frolov and Mikhail Gilmendinov introduce DISBench, an open-source benchmark suite crafted for evaluating memory performance across multicore processors and multiprocessor systems under various workloads [7]. DISBench encompasses three distinct memory access kernels: stream, stride, and random, each representing different categories of memory-intensive applications.

The BenchIT project offers a framework designed for conducting performance measurements on UNIX-based systems [8]. Their approach merges minimal software requirements with a straightforward interface for measuring kernels, along with a clear segregation of configuration, compilation, measurement, and result evaluation processes. In contrast to many other benchmark systems, the BenchIT environment offers functionalities across multiple levels: it facilitates measurements across varying problem sizes, graphical representation of results, and the ability to automatically compare with other measurements. This paper introduces the BenchIT platform and presents initial findings on selected machines.

In a separate article, cache and memory latency and bandwidth are evaluated utilizing the utilities offered by LMBench [9]. This paper specifically focuses on latency concerning CPU to memory and CPU to cache accesses. The methodology employed for determining latency remains consistent for both Three Chip and System-on-Chip (SoC) design architectures.

These benchmarking only deal with linear memory access, but real applications use a particular memory region several times. So it can’t show actual bandwidth of the memory hierarchy.

We know that theoretical hardware performance value does not always reflect real application performance due to many factors, like caching effects, data locality, and instruction sequences, among other things [3]. In this research work, we were not looking at these theoretical values. Instead, we were using a real application to measure a certain memory access pattern.

III. EXPERIMENTAL PROCEDURE

In this research work analytical study has been conducted on Intel core-i3, core-i5 and core-i7 architecture’s to measure performance for linear programming.

C program has been developed for solving linear equation. For this purpose, C program for Matrix Transpose, Matrix Multiplication and Matrix Inversion have been developed. Gauss-Jordan elimination method has been used to find out the inverse matrix (which is most commonly used for the linear systems). Different size of matrix has been used for different memory block for solving these operations. These operations have been performed in a particular memory block for several times and the performance has been measured in terms of system clock. Processing time have been observed for these operations on wide variety size of matrix ranging from some hundred to some millions. Then it is tried to hypothesize the correlation between the process time and the memory access time and measured the performance in terms of system clock.

IV. RESULTS AND DISCUSSIONS

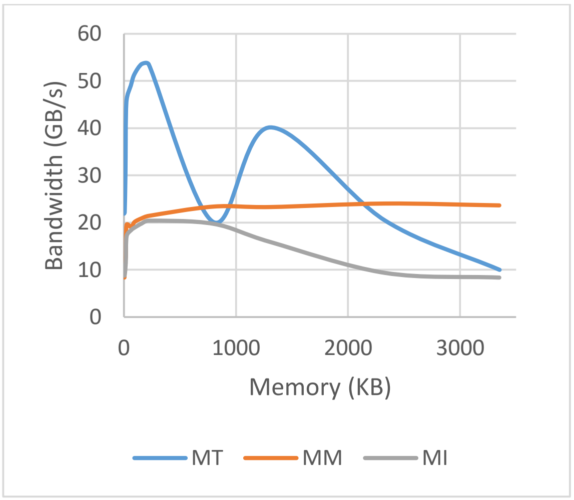
In this research work analytical study has been conducted on Intel core-i3, core-i5 and core-i7 architecture’s to measure performance for linear programming. Before collecting data, clock frequency has been fixed at their maximum and turbo boost technology have been disabled. Matrix transpose, matrix multiplication and matrix inversion operation has been performed on Ubuntu for computing bandwidth and latency. Bandwidth and latency are discussed for Intel core-i3, core-i5 and core-i7 architectures and also their virtual machines.

A. BANDWIDTH

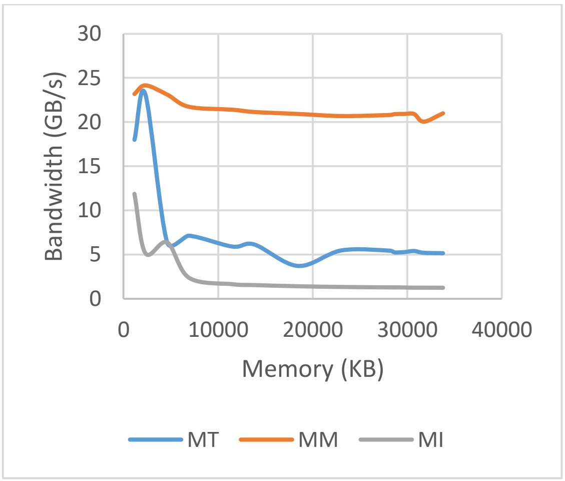
4.1 Core-i3 (DDR3L) architecture

We have performed matrix transpose (MT), matrix multiplication (MM) and matrix inversion (MI) operation on core-i3 (DDR3L) architecture for computing bandwidth.

Cache bandwidth and Main memory bandwidth of core-i3 (DDR3L) architecture for matrix transpose, matrix multiplication and matrix inversion operation are shown in figure-4.1.



a: Cache bandwidth



b: Main memory bandwidth

Figure-4.1: Bandwidth for different size of matrix

It is seen that, in the same memory region bandwidth is not same for all operations, because cache prefetch engine can't predict the next memory location perfectly for different operations.

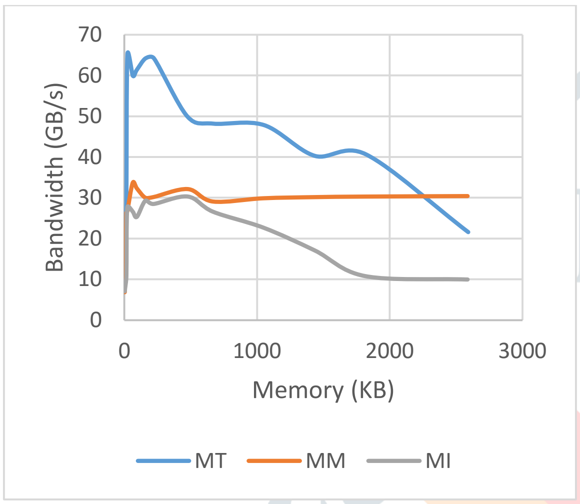
From figure-4.1(a) it is seen that, highest cache bandwidth for matrix transpose, matrix multiplication and matrix inversion operation are 53.55 GBps, 24.04 GBps and 20.41 GBps respectively. From the result of MT, it is seen that, there is a deep region of bandwidth. It may be occurred due to prefetching effect, non-uniform memory access effect (NUMA effect) etc.

From figure-4.1(b) it is seen that, highest main memory bandwidth for matrix transpose, matrix multiplication and matrix inversion operation are 23.13 GBps, 24.17 GBps and 11.85 GBps respectively.

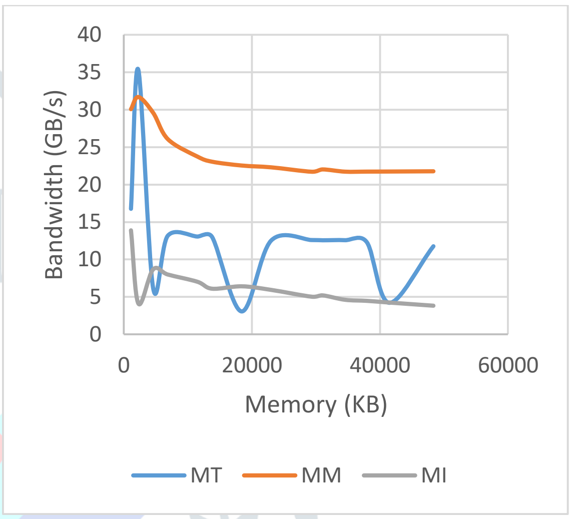
4.2 Core-i5 (DDR3L) architecture

We have performed matrix transpose (MT), matrix multiplication (MM) and matrix inversion (MI) operation on core-i5 (DDR3L) architecture for computing bandwidth.

Cache bandwidth and Main memory bandwidth of core-i5 (DDR3L) architecture for matrix transpose, matrix multiplication and matrix inversion operation are shown in figure-4.2.



a: Cache bandwidth



b: Main memory bandwidth

Figure-4.2: Bandwidth for different size of matrix

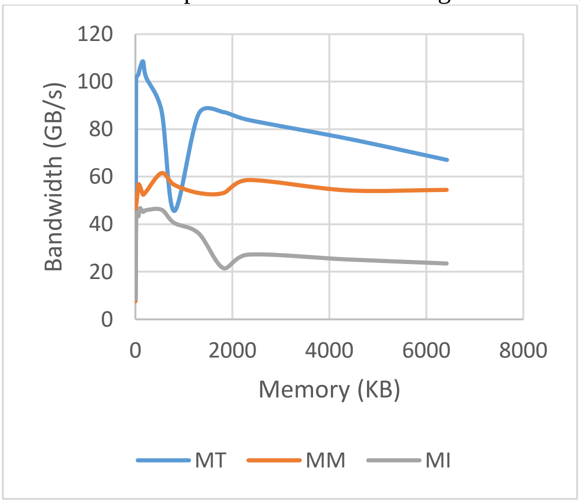
From figure-4.2(a) it is seen that, highest cache bandwidth for matrix transpose, matrix multiplication and matrix inversion operation are 65.15 GBps, 33.72 GBps and 30.29 GBps respectively.

From figure-4.2(b) it is seen that, highest main memory bandwidth for matrix transpose, matrix multiplication and matrix inversion operation are 35.32 GBps, 31.7 GBps and 13.89 GBps respectively.

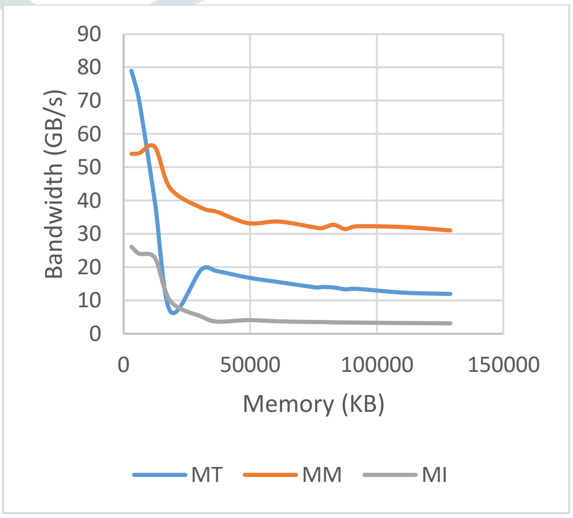
4.3 Core-i7 (DDR4) architecture

We have performed matrix transpose (MT), matrix multiplication (MM) and matrix inversion (MI) operation on core-i7 (DDR4) architecture for computing bandwidth.

Cache bandwidth and Main memory bandwidth of core-i7 (DDR4) architecture for matrix transpose, matrix multiplication and matrix inversion operation are shown in figure-4.3.



a: Cache bandwidth



b: Main memory bandwidth

Figure-4.3: Bandwidth for different size of matrix

From figure-4.3(a) it is seen that, highest cache bandwidth for matrix transpose, matrix multiplication and matrix inversion operation are 108.41 GBps, 61.44 GBps and 46.79 GBps respectively.

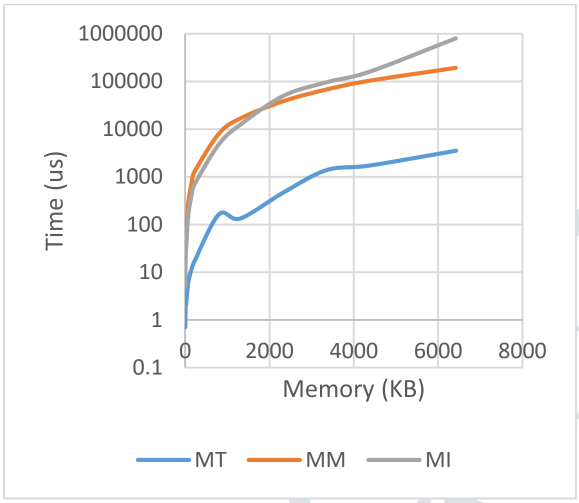
From figure-4.3(b) it is seen that, highest Main memory bandwidth for matrix transpose, matrix multiplication and matrix inversion operation are 78.97 GBps, 56.13 GBps and 26.12 GBps respectively.

B. LATENCY

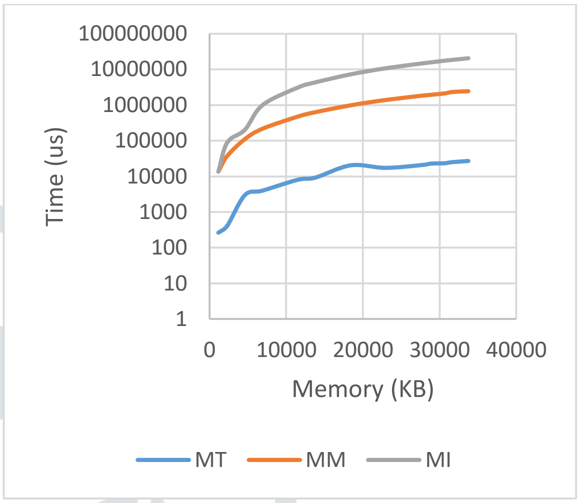
4.4 Core-i3 (DDR3L) architecture

We have performed matrix transpose (MT), matrix multiplication (MM) and matrix inversion (MI) operation on core-i3 (DDR3L) architecture for computing latency.

Cache latency and Main memory latency of core-i3 (DDR3L) for matrix transpose, matrix multiplication and matrix inversion operation are shown in figure-4.4.



a: Cache latency



b: Main memory latency

Figure-4.4: Latency for different size of matrix

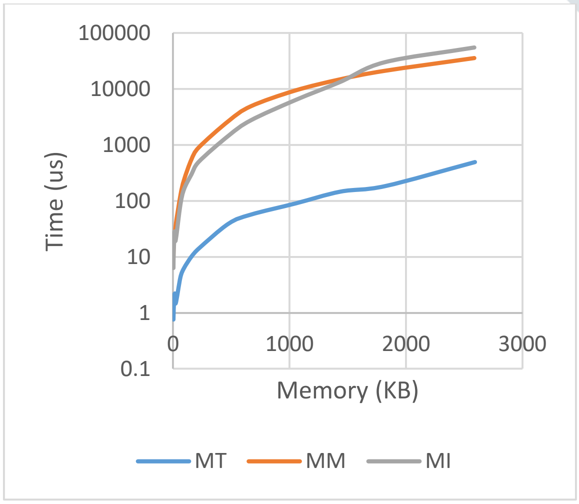
From the above figures, we can see that cache latency is increasing exponentially with the increase of size of matrix. From figue-4.4(a) it is seen that, cache latencies for MM and MS operations are almost same but latency for MT is very lower than MM and MI. So, logarithmic scale is used in y-axis in order to showing the results in one graph as time of MT operation is very small compared to MM and MI.

From figure-4.4(b), we can see that main memory latency is increasing exponentially with the increase of size of matrix, and latency at the time of MI operation is greater than MM and latency for MM operation is greater than MI.

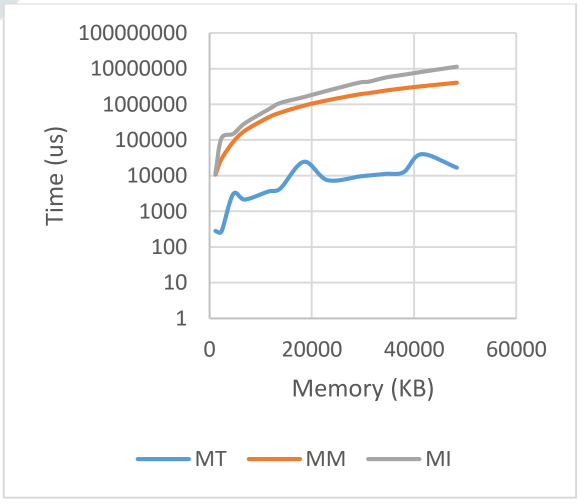
4.5 Core-i5 (DDR3L) architecture

We have performed matrix transpose (MT), matrix multiplication (MM) and matrix inversion (MI) operation on core-i5 (DDR3L) architecture for computing latency.

Cache and main memory latency of core-i5 (DDR3L) architecture for matrix transpose, matrix multiplication and matrix inversion operation are shown in figure-4.5.



a: Cache latency



b: Main memory latency

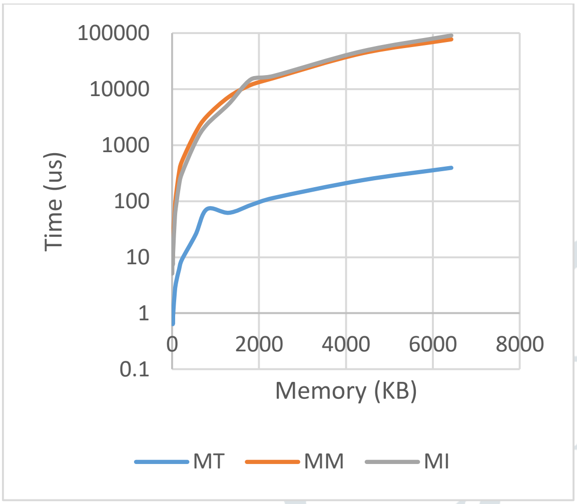
Figure-4.5: Latency for different size of matrix

From the above figures, we can see that latency is increasing exponentially with the increase of size of matrix and latencies for MM and MI operations are almost same but latency for MT is very lower than MM and MI. So, logarithmic scale is used in y-axis in order to showing the results in one graph as time of MT operation is very small compared to MM and MI.

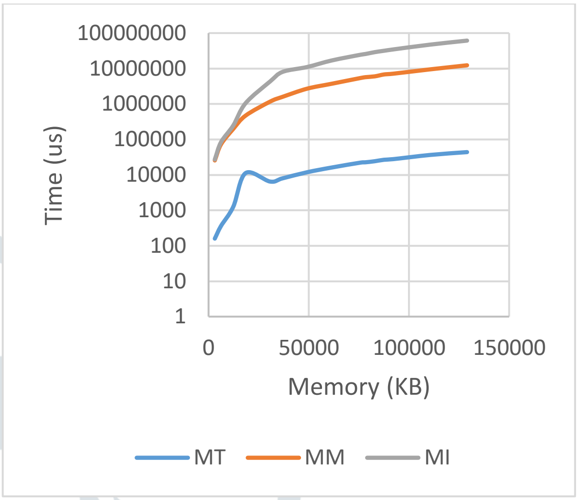
4.6 Core-i7 (DDR4) architecture

We have performed matrix transpose (MT), matrix multiplication (MM) and matrix inversion (MI) operation on core-i7 (DDR4) architecture for computing latency.

Cache and main memory latency of core-i7 (DDR4) architecture for matrix transpose, matrix multiplication and matrix inversion operation are shown in figure-4.6.



a: Cache latency



b: Main memory latency

Figure-4.6: Latency for different size of matrix

From the above figures, we can see that latency is increasing exponentially with the increase of size of matrix. From figue-4.6(a) it is seen that, cache latencies for MM and MI operations are almost same but latency for MT is very lower than MM and MI. So, logarithmic scale is used in y-axis in order to showing the results in one graph as time of MT operation is very small compared to MM and MI.

From figure-4.6(b), we can see that main memory latency is increasing exponentially with the increase of size of matrix, and latency at the time of MI operation is greater than MM and latency for MM operation is greater than MI.

V. CONCLUSION

In summary, analytical study has been conducted on Intel core-i3, core-i5 and core-i7 architecture's to measure performance for linear programming. Matrix transpose, matrix multiplication and matrix inversion operation have been performed for computing bandwidth and latency. Different size of matrix has been used for different memory block for solving these operations. These operations are performed in a particular memory block for several times and the performance has been measured in terms of system clock.

Results of cache benchmarking and memory benchmarking of Intel core-i3, core-i5 and core-i7 architectures are discussed in chapter IV. Data flow rate have analyzed to assess the performance characteristics for transferring data between different level of caches and from cache to main memory. It is seen that theoretical hardware performance value does not always reflect actual application performance. It is not possible to measure actual performance by linear benchmarking because real applications use a particular memory region several times and different applications have different access pattern. Moreover, the cache replacement policy is not optimized for all types of applications. If it is possible to optimize the cache replacement policy for a particular application, then performance will be better for that particular application.

VI. FUTURE WORK PLAN

This research work can be extended in several ways:

- Efficient use of heterogeneous memory hierarchy using these performance characteristics.
- Enhancement of performance of a processor by cache optimization.
- Estimation of power consumption during execution.

VII. ACKNOWLEDGMENT

I would like to express my sincere gratitude to my research supervisor Md. Mamun-ur-Rashid Khandker, and co-supervisor Md. Ashraful Islam, for their invaluable support, guidance, and mentorship throughout this research.

REFERENCES

[1] D. Molka, D. Hackenberg, R. Schöne, and M. S. Müller, “Memory performance and cache coherency effects on an intel nehalem multiprocessor system,” Parallel Archit. Compil. Tech. - Conf. Proceedings, PACT, pp. 261–270, 2009.

[2] R. Schöne, W. E. Nagel, and S. Pflüger, “Analyzing Cache Bandwidth on the Intel Core 2 Architecture,” Parallel Comput. Archit. Algorithms Appl., vol. 38, pp. 365–372, 2007.

[3] Jenifer Hopper, “The Linux on Power Community : Untangling memory access measurements - memory latency,” pp. 1–8, 2016.

[4] J. McCalpin, STREAM: Sustainable Memory Bandwidth in High Performance Computers. 1991.

[5] R. Ruggiero, “Measuring cache and memory latency and CPU to memory bandwidth,” White Pap. Intel Corp., no. December, pp. 1–14, 2008.

[6] “How CPU caching speeds processor performance.” [Online]. Available: https://searchwindowsserver.techtarget.com/tip/How-CPU-caching-speeds-processor-performance. [Accessed: 03-Apr-2019].

[7] A. Frolov and M. Gilmendinov, “DISbench: Benchmark for memory performance evaluation of multicore multiprocessors,” in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2013, vol. 7979 LNCS, pp. 197–207.

[8] G. Juckeland, W. E. Nagel, and S. Pfl, “Performance comparison and optimization : Case studies using BenchIT,” vol. 33, 2006.

[9] L. McVoy and C. Staelin, “lmbench: Portable Tools for Performance Analysis,” Proc. USENIX Annu. Tech. Conf., no. January, pp. 279–294, 1996.