Memory Bandwidth and Latency in HPC: System Requirements and Performance Impact

Author:
Milan Radulović

Thesis director:
Dr. Petar Radojković
Tutor:
Dr. Eduard Ayguadé

This dissertation is submitted in fulfillment of the requirements for the degree of

Doctor of Philosophy

Departament d' Arquitectura de Computadors (DAC)
Universitat Politècnica de Catalunya (UPC)

March 2019

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Memory bandwidth and latency in HPC: system requirements and performance impact

Milan Radulović

To my family.

Mojoj porodici.

Acknowledgments

In the year when UNIX operating system is celebrating 50 years, World Wide Web is having its 30th birthday and v1.0 of Linux kernel is marking 25 years, I am giving a modest contribution in the form of this thesis. Finishing it has been an interesting journey, but it paid off; AMD company is naming their third generation of EPYC server processors after me [1].

The biggest thanks goes to my advisor Dr. Petar Radojković for all the help, guidance and support. I am grateful he gave me the opportunity to be a part of his group. My research ideas were considerably amplified and properly focused thanks to him, leading to positive outcomes. Him being motivational and incentive was very precious, especially during tight deadlines. It was a pleasure and honor working with him. Special thanks to Dr. Paul Carpenter, for his help in shaping and polishing of our studies. His cleverness significantly improved the quality of our research. Nevertheless, I would also like to thank professor Dr. Eduard Ayguadé for all the help and support in research and administrative topics, ensuring our work goes smoothly and without any problems.

In addition, thanks to the BSC Support and Operations departments for their endless patience and availability to realize dozens of requests I sent. Thanks to the BSC Tools department as well, for sharing their valuable advices and codes.

I would like to thank the pre-defense committee and the external reviewers for their constructive and meticulous comments which improved the quality of this thesis.

I am thankful to all the people with whom I have been sharing the office for past five years. You're ideas, smiles and cookies made everyday work enjoyable. I also thank other people that were in BSC during this time, for inspiring lunches and awesome basketball and football games.

My gratitude goes to my family and friends, from Barcelona and Serbia, for making my personal life joyful during PhD. Thank you all for your support and motivation, and for bringing rakija. Also, thanks to MI Labs® for including me in their stunning projects.

A huge thank you goes to my parents and my sister, who believed in me and encouraged me during these years. I wouldn't manage without their help.

Finally, I would like to thank my wife Milica, whose patience, love and understanding can move mountains, but produced this thesis instead. And, to my daughter Sofija, whose smile instantly casts sunlight in darkest moments. I am privileged being by their side.

Author

This work was supported by the Collaboration Agreement between Samsung Electronics Co., Ltd. and Barcelona Supercomputing Center; by the Spanish Ministry of Science and Technology (project TIN2015-65316-P), Generalitat de Catalunya (contracts 2017-SGR-1414 and 2017-SGR-1328), Severo Ochoa Programme (SEV-2015-0493) of the Spanish Government; and the European Union's Horizon 2020 research and innovation programme under ExaNoDe project (grant agreement № 671578).

Abstract

A major contributor to the deployment and operational costs of a large-scale high-performance computing (HPC) clusters is the memory system. In terms of system performance it is one of the most critical aspects of the system's design. However, next generation of HPC systems poses significant challenges for the main memory, and it is questionable whether current memory technologies will meet the required goals. This motivates a lot of research in future memory architectures, their capacity, performance and reliability. In this thesis we focus on HPC performance aspects of the memory system design, covering memory bandwidth and latency.

We start our study with an extensive analysis, evaluating and comparing three mainstream and five alternative HPC architectures, regarding memory bandwidth and latency aspects. Increasing diversity of HPC systems in the market causes their evaluation and comparison in terms of HPC features to become complex and multi-dimensional. There is as yet no well established methodology for a unified evaluation of HPC systems and workloads that quantifies the main performance bottlenecks. Our work provides a significant body of useful information for HPC practitioners and infrastructure providers, and emphasizes four usually overlooked aspects of HPC systems' evaluation.

Understanding the dominant performance bottlenecks of HPC applications is essential for designing a balanced HPC system. In our study, we execute a set of real HPC applications from diverse scientific fields, quantifying key performance bottlenecks: FLOPS performance and memory bandwidth congestion. We show that the results depend significantly on the number of execution processes, which is typically overlooked in benchmark suites, and argue for guidance on selecting the representative scale of the experiments. Also, we find that average measurements of performance metrics and bottlenecks can be highly misleading, and suggest reporting as the percentage of execution time in which applications use certain portions of maximum sustained values.

Innovations in 3D-stacking technology enable DRAM devices with much higher bandwidths than traditional DIMMs. The first such products hit the market, and some of the publicity claims that they will break through the memory wall. We summarize our preliminary analysis and expectations of how such 3D-stacked DRAMs will affect the memory wall for a set of representative HPC applications. Higher bandwidth may lower average latency, provided that our applications offer sufficient memory-level parallelism (MLP) and that CPU architectures can exploit it. We conclude that although 3D-stacked

DRAM is a major technological innovation, it is unlikely to break through the memory wall — at best, it moves it.

Novel memory systems are typically explored by hardware simulators that are slow and often have a simplified or obsolete model of the CPU. We propose an analytical model that quantifies the impact of the main memory on application performance and system power and energy consumption, based on the memory system and application profiles. The model is evaluated on a mainstream platform, comprising various DDR3 memory configurations, and an alternative platform comprising DDR4 and 3D-stacked high-bandwidth memory. The evaluation results show that the model predictions are accurate, typically with only 2% difference from the values measured on actual hardware. Additionally, we compare the model performance estimation with simulation results, and our model shows significantly better accuracy over the simulator, while being faster by three orders of magnitude, so it can be used to analyze production HPC applications on arbitrarily sized systems.

Overall, we believe our study provides valuable insights on the importance of memory bandwidth and latency in HPC: their role in evaluation and comparison of HPC platforms, guidelines on measuring and presenting the related performance bottlenecks, and understanding and modeling of their performance, power and energy impact.

Contents

Listing of figures

Listing of tables

90% of supercomputer design is the memory system...and so is the other 10%.
Shekhar Borkar, IEEE Fellow

1 Introduction

The use of high-performance computing (HPC) has become globally widespread across all branches of government, academia, industry and commerce, touching almost every aspect of daily life: medicine, energy, transportation, business, communications, etc. Besides traditional science applications in physics, biology, chemistry, and others, supercomputing is indispensable in engineering, weather prediction and climate modeling, complex financial modeling, business analytics, cryptography, signals processing and other computation-intensive research areas. HPC became widely accepted and accessible to greater diversity of institutional and commercial users. The growing need for more powerful systems motivated further research and development which resulted in speed increment of the worlds fastest supercomputers by a factor of roughly two million over the past 25 years. Not any other industry can compare with such a rapid transformation.

The future of HPC is focused on the achievement of the next step: *Exascale* supercomputers. Such machines will have the performance measured in *exaflops* (EFLOPS), i.e. 10^{18} floating operations per second (FLOPS) in double-precision. For the comparison, the most powerfull supercomputer today is *Summit* from the Oak Ridge National Laboratory with the performance of 143.5 PFLOPS [2]. However, the road to exascale milestone is not so straightforward. Challenging constraints primarily in power consumption of exascale supercomputers will force the future architectures to change dramatically in order to meet the next goals [3]. The doubling of per-chip transistors count every 18-24 months (known as the *Moore's Law*[1] [4]) has ended and it is replaced by increasing of number of cores or other

[1] Moore initially said transistors on a chip would double every year, and afterwards recalibrated it to every two years in 1975. David House, an Intel executive at the time, noted that the changes would cause computer performance to double every 18 months.

parallelism mechanisms. The architectures are becoming heterogeneous, employing accelerators, asymmetrical CPU cores, hybrid memory systems, etc. Nevertheless, the applications and algorithms will have to change and adapt to new architectures, managing locality, parallelism and resilience in order to achieve the exaflop performance. Exploring and simulating these large-scale architectures is complex and multi-dimensional [5, 6].

One of the major contributor to the deployment and operational costs of a large-scale HPC cluster is the memory system [3, 7, 8]. In terms of system performance, it is one of the most critical aspects of the system's design [9, 10, 11]. Challenges and research directions of HPC memory systems are explained more closely in the next section.

1.1 Memory systems in HPC

For last 40 years the dominant memory technology for all computing systems is Dynamic Random Access Memory (DRAM). At the time it showed up in the 1970s, it had high density, high performance and low cost per bit. However, it could not follow the aggressive advance of computing units and started to fall behind, widening the gap between the CPU cycle time and memory access latency, known as the *Memory wall* [11]. The community recognized the problem and memory system became one of the most important factors of system performance [11, 12]. Besides performance, power consumption of data movement will dominate the power consumption profile of future systems [13]. With the scaling of DRAM approaching its end in many dimensions while constraints of exascale systems [14] continue to push the limits in power, capacity, bandwidth and latency, it is questionable whether DRAM will meet the requirements of future HPC systems. In the next sections we briefly describe the research directions in the memory system landscape. Section 1.1.1 shows the current problems and future constraints of memory capacity. Innovative architectural technologies are presented in Section 1.1.2. Challenges in reliability related to memory systems can be found in Section 1.1.3. Afterwards, in Section 1.1.4 we describe the issues and future requirements of memory system performance in HPC.

1.1.1 Memory capacity

Back in the 1970s when first DRAM products appeared, they were satisfying the needs of typical computing systems. Manufacturing improvements that led to larger die sizes, innovative cell layout for better efficiency and increasing lithography resolution resulted in 4-fold increase in memory capacity roughly every three years, matching Moore's prediction [4]. It lasted until 128Mb chip was produced, after which the increment reduced to 2-fold every four years. Nowadays shrinking the cells further is a big challenge from technological side, inducing performance and reliability issues. Increasing DRAM capacity per chip leads to increment in time and energy for memory refresh operations [15]. Smaller capacitors have lower retention time and need to be refreshed more often, while increase in storage capacity increases the total time for refresh operations, since there are more cells to refresh. All this can significantly impact performance. Also, reducing the space between the cells increases the possibility of crosstalk between the adjacent wordlines. The crosstalk may induce flipping of bits which are not directly accessed, seriously affecting system security. The effect was recently discovered and named *Rawhammer* [16].

The future HPC systems' power consumption is one of the major constraints and it will impact DRAM capacity. On the other side, future applications will put even more pressure on memory capacity [14]. Capacity limitations can affect balance, parallel efficiency and scalability of future applications. Having less memory per single HPC node implicates less useful work can be done locally without additional communication to other nodes, which can introduce significant overhead [17].

Novel 3D-stacked DRAM products appeared recently, having much higher bandwidth than DDR devices. However, they still have a significantly higher cost and lower capacity with respect to the standard DIMMs. Contrary to them, storage-class memory technologies and products [18, 19], provide significant increment in memory capacity with a moderate increment in cost and power consumption. However, the access latency of these technologies exceeds the DRAM latency by orders of magnitude, significantly impacting performance.

1.1.2 Architectural innovations

As mentioned in the previous sections, mature DRAM technology came to saturation in many aspects which motivated further research of different memory technologies. It includes microarchitectural changes of the memory chips, new packagings of memory modules and innovative interfaces between the memory and the processor.

3D-stacking is one of the solutions to significantly increase the available memory bandwidth, improve energy per bit and energy per GB/s [20, 21, 22, 23]. Stacking DRAM dies or wafers reduces the interconnection length between them and latency of communication, while increasing the available bandwidth. Unfortunately, the capacity of these devices is still not on par with current DDR devices. The interfaces to connect these stacks with the processor are challenging, since they require high-speed signaling in order to deliver high bandwidths. Next generation of these interfaces could use optical links [13].

There are different memory technologies that provide non-volatility and achieve high densities i.e., capacity. Non-volatile memories (NVMs) eliminate the energy consumption of refresh operations and provide support for check-pointing of applications [24]. Check-pointing saves the current system state to a non-volatile storage, from where it can be restarted if the application terminates due to a failure. Phase Change Memory (PCM) stores data using a phase-change material that can be in crystalline or amorphous physical state [25, 26, 27]. It provides superior density, and therefore capacity, relative to DRAM. However, write endurance limits PCM devices lifetime. Resistive Random Access Memory (R-RAM) is another promising technology, storing data as different resistance of a solid-state dielectric material [28]. Memristor technology [29] has a similar method of storing the data, using the element similar to resistor, however memristor changes its resistance as the electric charge passes through the device. Once the resistance is changed, it remains constant if no current is applied. Spin transfer torque magnetic RAM (STT-MRAM) [30] is a magnetoresistive memory, using magnetic storage elements instead of usual electric charge storage. STT-MRAM uses Magnetic Tunneling Junction's (MTJ) resistance to store data. It is mostly used in embedded systems, but an ongoing research suggest its use as a main memory [31, 32]. Nano-RAM (NRAM) [33] is a carbon nanotube (CNT) based resistive NVM, with high density and endurance, compatible with existing CMOS production process. One of the recent emerging non-volatile memory technologies was introduced by Intel and Micron, named 3D XPoint memory [18, 19], which technology is not publicly disclosed.

However, current NVM devices have higher latencies than conventional DRAMs and possibly lower than solid-state drives (SSDs). In order to overcome their limitations, future HPC architectures may include some hybrid combination of different memory technologies. Hybrid memory systems, with different levels of hierarchy, in theory could provide the benefits of each technology while hiding its weaknesses. However, design of these systems is complex; it requires analysis of various tradeoffs and design of a non-trivial algorithms for optimal data partitioning and distribution [34, 35].

1.1.3 Memory reliability

HPC clusters are vulnerable to various errors than can cause failure of the application process, whole application or the entire node. Clusters are composed of many nodes and failure of just one node can terminate the application which was executing for days. Modern error-correcting code (ECC) [36] protection can mitigate single-bit errors, however multi-bit errors are still a significant problem. Typically, every few days at least one of the nodes needs to be rebooted [37]. Exascale machines will be larger, more complex and more heterogeneous, which will probably increase the fault rate. Therefore, the resilience of future HPC clusters is one of the major concerns.

Memory system is one of the main sources of failures in modern HPC systems [38, 39]. Current memory technologies are susceptible to cosmic rays, crosstalk and technological imperfections, inducing temporal or permanent damage to storage cells, or cause data corruption on the memory bus [16]. Ever increasing density of memory devices shrinks the storage cells making them even more sensitive to any external disturbance, which will affect future HPC systems [40].

On the other side, none of the emerging memory technologies is failure safe and the same mechanism of error detection and correction have to be applied. Hence, further research in characterization of memory failures in large clusters is essential to understand the rates, distribution and patterns of memory failures.

1.1.4 Memory performance

Ever since the improvement rate of CPU performance started to surpass the improvement rate of DRAM performance, main memory became a bottleneck in memory intensive applications which are commonly found in HPC. Growing disparity between the CPU cycle time and memory access latency became known as the *Memory wall* (see Section 2.2.1). Much has been done since it was published to reduce its impact, however it still presents one of the main performance bottlenecks.

Memory performance has two components which are correlated, memory bandwidth and memory latency (see Section 2.2). Each of them has its performance role and both are important in HPC systems. Memory bandwidth of DDR memories is advancing slowly, while memory latency is almost constant. As number of cores per socket increases, memory bandwidth per CPU core decreases. FLOPs performance also increases more rapidly, causing Byte/FLOP ratio of modern architectures to decrease [41]. Byte/FLOP ratio of a platform is calculated by dividing platform's sustained memory bandwidth with its sustained FLOPs performance. Platforms with low Byte/FLOP ratio are well suited for compute intensive applications and in these platforms, memory bandwidth may easily become a performance bottleneck. The platforms with high Byte/FLOP ratio perform well with applications that put a high pressure on memory bandwidth, often present in HPC, but floating-point processing power may limit the performance. Since future applications will require higher bandwidth and lower latency, it is questionable whether conventional DRAMs will provide the necessary performance under the strict power consumption constraints.

Significant effort is invested in research of novel memory technologies that will cope better with increasing pressure on memory system by the future applications. It resulted in 3D-stacked main memories [21, 22, 23] (see Section 2.3), which have superior memory bandwidth compared to conventional DDR memories and achieve lower energy per GB/s. With this significant leap forward in memory bandwidth, Byte/FLOP ratio improved. However, this comes with a cost of memory access latency which is higher than of DDR DRAMs, although for some devices it is claimed being close [23]. High access latency comes from the complexity of memory controller and handling of memory requests, not from the memory device itself. Reducing memory latency on the other hand is not trivial. Reduced Latency DRAM (RLDRAM) devices have lower latency by introducing microarchitectural changes, such are small sub-arrays and activation of many rows, with the cost of smaller density and higher power consumption [42]. However they are mostly used in specialized applications. We intent to emphasize memory bandwidth and latency aspects of HPC systems evaluation in this thesis, making first steps towards more reliable and uniform evaluation methodology.

Although it is well known that HPC applications are bandwidth-sensitive, latency sensitivity may be even more important for certain applications [43, 44]. It should be noted that the Memory Wall was defined in terms of latency, not bandwidth [11]. Algorithms like sparse linear algebra, structured grids and combinational logic are more memory bandwidth limited, while spectral methods (FFT), unstructured grids, N-Body methods and graph traversals are memory latency bounded [43, 45]. Apart from HPC, algorithms used in machine learning like dynamic programing, backtrack and branch+bound, and graphical models are also memory bounded [43, 46]. Therefore, it is worth investing in characterization of HPC applications, in terms of bandwidth and latency, in order to quantify performance bottlenecks and better optimize future HPC architectures. This thesis analyses memory bandwidth requirements of large HPC applications on a production HPC cluster, with respect to scaling-out and granularity of measurements.

Finally, memory bandwidth and memory access latency present significant performance factors of future computer systems. Besides novel memory architectures, like 3D-stacked high-bandwidth solutions and storage-class 3D XPoint, conventional DDR memory space is moving forward. Emerging server processors are increasing number of DDR memory channels to eight or twelve per socket [47, 48], and DDR5 memory will roll out soon [49]. New

DDR5 will have number of improvements, like higher memory-bus frequency, two 32-bit channels per single DIMM, larger burst-size and higher number of banks in comparison to DDR4, innovative write pattern mode, etc. The amount of performance improvement these new memory architectures and topologies will introduce depends on the specific workload and its sensitivity to memory bandwidth and latency. In this thesis, we quantify performance, power and energy difference on a new memory system architecture or topology, using an analytical model. It is based on the application profile, target memory system profile and several architectural-dependent parameters. The proposed analytical approach shows high precision and accuracy, and has the advantage of being much faster than a usual simulation-based analysis with the ability to analyse complex workloads such are production HPC applications.

1.2 Thesis contributions

Our study analyses memory bandwidth and latency aspects of HPC systems and applications. We evaluate both mainstream and alternative HPC systems and characterize production HPC applications. Finally, we perform estimation of application performance and system power and energy, using analytical model based on the memory system evaluation.

1.2.1 Memory bandwidth and latency aspects in HPC systems evaluation

The analysis of memory bandwidth and latency is the important aspect of HPC systems evaluation and comparison, and it is the first study presented in this thesis. We share our experience of evaluating and comparing a diverse set of HPC systems, consisting of three mainstream and five alternative architectures. To our knowledge, there are no studies which analyze this many platforms, three mainstream and five alternative ones. In addition to presenting a large body of quantitative results, we emphasize four usually overlooked aspects of HPC platform evaluation related to memory bandwidth and latency.

First, we show a platform's performance and energy-efficiency depend significantly (n-fold) on the target application characteristics. We strongly advocate that any comparison among platforms should include High-Performance Conjugate Gradients (HPCG) measurements, alongside with High-Performance Linpack (HPL). HPCG adds the boundary for memory-intensive HPC applications, in addition to compute-intensive boundary from HPL.

Second, we detect a significant range in the main memory access latency, with a factor of three difference between the fastest and slowest platforms under study. As we described in Section 1.1.4, memory access latency has a direct performance impact for many applications. Therefore, it should be minimized in HPC servers, and any increment above about 100 ns should be analysed and justified.

Third, we find that the Byte/FLOP ratio can differ by a factor of up to 21× between platforms. While mainstream platforms show a decreasing tendency, alternative platforms trend upwards in this metric. We propose for a community to properly define this ratio for HPC applications, since it has a direct impact on system performance.

Fourth, our results show that sustainable FLOPS performance and memory bandwidth on the alternative platforms can deviate more than 70% from theoretical performance. An explanation could be that the overall system, including both hardware and system software, cannot fully utilize Single Instruction, Multiple Data (SIMD) floating-point execution units, i.e. not achieving close to maximum FLOPS performance, or data-transfer mechanisms, i.e. not saturating enough the available memory bandwidth. Therefore, we strongly suggest not rely on theoretical performance, even in a first-order system provisioning.

1.2.2 Memory bandwidth requirements of HPC applications

Next problem we analyse in the thesis are the memory bandwidth requirements of production HPC applications. A clear understanding of HPC system performance factors and bottlenecks is essential for designing an HPC infrastructure with the best features and a reasonable cost. In this study, we analyse the methodology of quantifying key performance bottlenecks: FLOPS performance and memory bandwidth congestion, and the implications of scaling-out. We execute seven production HPC applications, together with HPL and HPCG, on a production HPC cluster and reach two main conclusions.

When executing production HPC applications, our findings show that HPC application performance and CPU/memory system bottlenecks are strongly dependent on the number of application processes. This is typically overlooked in benchmark suites, which seldom define how many processes should be used. We argue that it is essential that HPC application suites specify narrow ranges on the number of processes, so that the results are representative of real world application use, or that they at least provide some guidelines.

Additionally, we demonstrate that average measurements of performance metrics and bottlenecks can be highly misleading. Reporting key application measurements using the average values may conceal bursty behavior, and give a misleading impression of how performance would be affected by changes in the platform's memory bandwidth. We suggest to avoid average figures when evaluating performance or bottlenecks, and instead measure the percentage of time that these figures are low, moderate and severe, with respect to their sustained peak, which gives a more precise picture of the application's or system's behavior.

1.2.3 First steps on the performance impact of memory bandwidth and latency

In the next study we give a preliminary analysis of the performance impact of 3D-stacked memories on HPC applications. Although it was defined more than two decades ago, the memory wall remains a fundamental limitation to system performance. Innovations in 3D-stacking technology enable DRAM devices with much higher bandwidths than traditional DIMMs. This study summarizes our preliminary analysis and expectations of how such 3D-stacked DRAMs will affect the memory wall for a set of representative HPC applications.

We conduct a preliminary evaluation of our analysis for a set of HPC applications running on a production system. We quantify the performance improvement of HPC applications by increasing the memory-bus frequency by 25%, which increases bandwidth by 25% as well. This change has no impact on memory latency, though. Our results show that bandwidth-hungry applications may benefit from increased available memory bandwidth. How well applications will exploit the higher bandwidth provided by emerging 3D-stacked DRAMs ultimately depends on the workload's memory-level parallelism. They will not improve the performance of applications with limited MLP. So in contrast to the publicity surrounding 3D DRAMs, they are unlikely to break through the memory wall — at best, they move it.

1.2.4 Memory system evaluation: Modeling system performance and energy without simulating the CPU

In the final study of this thesis we propose an analytical model that quantifies the impact of the main memory latency and bandwidth on application performance and system power and energy consumption. Since it is becoming questionable whether DRAM DIMMs will continue to scale and meet the industry's demand for high performance and high capacity memory, significant effort is therefore being invested into the research and development of future memory systems. These novel memory systems are typically explored using hardware simulators. Such an analysis is time consuming and limits the number of design options that can be explored within a practical length of time.

Our approach in this study is the analytical model that estimates the impact of the main memory on the application performance and system power and energy consumption, providing point estimates and error bars. The model is based on memory system profiling and instrumentation of the application execution. It has been evaluated on two actual platforms: Sandy Bridge-EP E5-2670 with four DRAM configurations DDR3-800/1066/1333/1600 on a set of SPEC benchmarks and HPC applications, and Knights Landing Xeon Phi 7230 with DDR4 and 3D-stacked Multi Channel DRAM (MCDRAM), using SPEC benchmarks.

The evaluation results show that the model predictions are very accurate. The average difference from the performance, power and energy measured on the actual hardware is only 2%, 1.1% and 1.7%, respectively. Additionally, we compare the model's performance predictions with simulation results for the Sandy Bridge-EP E5-2670 system with ZSim and DRAMSim2, and our model shows significantly better accuracy over the simulator. The model is also faster than the hardware simulator by three orders of magnitude, so it can be used to analyze production HPC applications, on arbitrarily sized systems.

1.3 Thesis organization

The thesis is divided into seven chapters following the subsequent structure:

- Chapter 2 describes basic DRAM architecture, its organization and operation. We also summarize promising 3D-stacked high-bandwidth memory solutions.

- The experimental environment of our studies is presented in Chapter 3. We describe hardware platforms, HPC benchmarks and applications, instrumentation tools and simulators used to obtain the results.

- In Chapter 4 we present the evaluation and comparison study of a diverse set of mainstream and alternative HPC platforms. Besides a large number of quantitative results, we highlight four memory bandwidth and latency aspects of HPC platform evaluation, which are often overlooked.

- Further in Chapter 5 we analyse memory bandwidth requirements of HPC applications, and show that CPU and memory system bottlenecks are strongly dependent on the number of application processes and granularity of measurements.

- In Chapter 6 we give a preliminary analysis on the performance impact of memory bandwidth and latency. We present our initial estimation on how will improved memory bandwidth of 3D-stacked DRAMs impact performance of HPC applications. We compare theoretical perspective of bandwidth and latency in the case of conventional DDR and 3D-stacked memories, and analyse performance improvement when we increase DDR memory frequency, and therefore bandwidth, by 25%.

- Application performance and system power and energy estimations, based on the analytical model, are given in Chapter 7. We describe in detail the application and memory system profiling, and performance, power and energy models. Afterwards, we evaluate the models on a mainstream Sandy Bridge platform and on an alternative Knights Landing platform. We also compare the performance model with the state-of-the-art ZSim+DRAMSim2 simulators.

- Finally, in Chapter 8 we summarize the conclusions of all thesis contributions and give future research directions.

Cache: a safe place for hiding or storing things.

Webster's New World Dictionary of the American Language, Second College Edition (1976)

2 Background

To provide a better understanding of the modern memory systems, in this chapter we give a description of the DRAM memory system architecture and its operation. We explain how memory is organized from a single DRAM cell level to Dual In-Line Memory Module (DIMM), and how read and write operations are performed in DDR memory systems. Nevertheless, we describe how the concepts of memory bandwidth and memory access latency are interrelated. Together with it, we explain the *Memory Wall* and its impact on system performance. Finally, we give an overview of novel high-bandwidth memory solutions, which could reinforce or even supersede DDR memories in future HPC systems.

2.1 DRAM memory system

The main memory system comprises several components, displayed in Figure 2.1. Memory controller receives memory requests from the CPU, schedules them, and issues commands to memory devices. Although it can be located on-chip or off-chip, in current computer systems it is usually integrated in the platform CPU. Memory bus connects memory controller with the memory devices via several memory channels. In current DRAM DDR systems, each channel carries commands, addresses and data using separate signals. Portion of memory channel that carries data, is usually 64-bit wide in current systems, with additional 8 bits if error-correcting codes (ECC) are used. Memory devices consist of memory chips which integrate the very memory array. They are usually grouped in DIMMs which thus provide wider data-bus and higher memory capacity than individual memory chips. DIMMs are usually located in special slots on computer motherboards, for servicing and future upgrades.

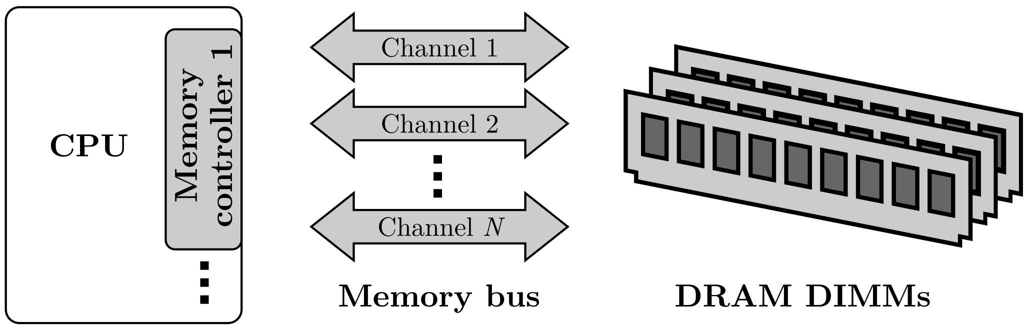


Figure 2.1: Typical main memory system comprises one or more in-CPU memory controllers, per-controller memory bus with N channels and DRAM DIMMs as memory devices.

2.1.1 DRAM organization

The basic building block of DRAM memory is a memory cell. It consists of a transistor-capacitor pair, representing a single bit of memory. These cells are organized into memory arrays, a grid-like structures displayed in Figure 2.2. Each DRAM cell is connected to a *wordline*, which forms horizontal lines of the array in Figure 2.2, and a *bitline*, which forms vertical lines of the memory array. Each DRAM cell can be accessed by specifying its *row address* and *column address.* Once the specified row address turns-on the corresponding *wordline*, capacitor charge from all cells on the selected wordline is transfered to its bitlines. The voltage change on bitlines is further amplified with *sense amplifiers*, referred to as a *row-buffer.* The row-buffer now effectively holds the data of the entire row of the memory array. Afterwards, column address specifies a single column from the row-buffer which goes to the output pins of the chip. This corresponds to the *read* memory access. Similarly, *write* memory access stores the data from the chip pins to the row-buffer, transfers the data from the row-buffer onto the bitlines and further to the cells selected by the wordline. Since the cell capacitor has leakage, it looses charge with time and has to be refreshed periodically in order to preserve the stored information. This corresponds to *DRAM refresh* operation, and it is performed periodically by the memory controller.

DRAM chips have several memory arrays that act in unison or independently. Usually four or more memory arrays is grouped and operating as a unit, so column is effectively several bits wide. If there are four memory arrays, these devices are marked as **x4** implying that the column is 4 bits wide. So, every memory access reads or writes four bits of data through four data pins on the chip package. Currently there are x4, x8, x16 and x32

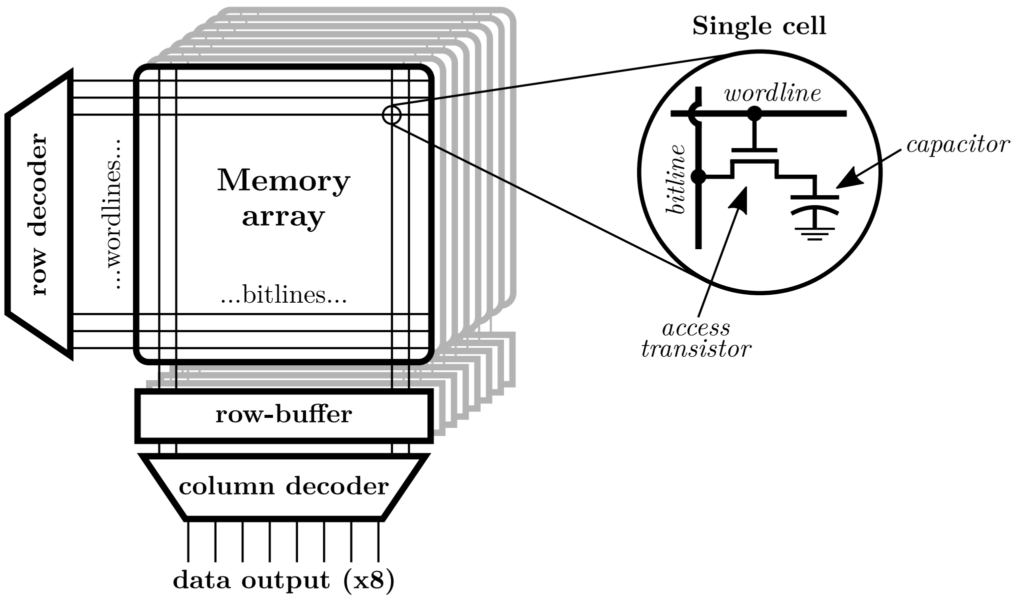


Figure 2.2: Basic DRAM structure, showing memory arrays in a x8 memory device [50]. Single cell is accessed using the corresponding wordline and bitline.

devices on the market. This group of memory arrays that operate in unison is called a *bank*. DRAM chips have multiple banks, increasing the parallelism of accesses since multiple banks can be accessed concurrently. DDR3 memory systems usually have 8 banks, while DDR4 memories increased number of banks to 16. Finally, several memory chips can be grouped to form a *rank*. All chips in a rank are connected to the memory channel and respond to the same DRAM commands. Number of memory chips in a single rank is determined by the number of data bits of each memory chip, which aggregate to form a data-bus of the memory channel. If x8 devices form a rank and memory channel comprises 72 data bits, exactly 9 memory chips form a rank. Each memory DIMM can contain from one to four memory ranks.

Therefore, the physical address of the particular data word is decomposed in the memory controller, specifying memory channel, rank, bank, row and column as a unique location. For example, in the case of a read access, when the address is decomposed and the request is sent over the memory bus, the target cells are accessed and all the chips in the rank output the corresponding data in the same time. Hence, the target data word is present on the memory bus and it can be transfered to the CPU. DDR memories transfer data in bursts, so every access transfers several data words from consecutive columns. For DDR3 and DDR4 memories the length of the burst is 8, so every memory access transfers eight data words, i.e. 64 bytes in total. Because of multiple memory channels, ranks and banks which can be accessed simultaneously, memory requests are *interleaved* by the memory controller to better utilize the memory bus and to achieve higher bandwidth.

2.1.2 DRAM operation

In order to read the specified column from the row-buffer, internal circuitry in the memory chip has to turn-on the wordline which corresponds to the specified row address, transferring the target data from the cells to the row-buffer. This operation is called *row activation*, and it is issued by the memory controller to the corresponding DIMM. Since bitlines in the memory array connect the cells across the vertical dimension, care must be taken not to output more than one cell on the single bitline in the same time, which would invalidate the stored information. To prevent this, at one moment exactly one wordline can be turned-on in the same memory bank, i.e. only one row can be activated at a time. Memory controller takes care of this, since it has the info about all the banks and knows if there is any row already activated among the banks. If another row in the same bank has to be activated, the already activated row has to be closed. This operation of closing the row is named *precharge* operation.

Therefore, we distinguish three scenarios when accessing the target data, each having different latency of access. The first scenario is the case when the target row containing the required data is already activated, and its content is in the row-buffer. It is a *row-buffer hit* access and reading or writing the data in this case has a minimum latency, skipping the row activation step. Therefore, it requires only reading or writing the target data from or to the row-buffer, specified by the column address. The second scenario is the case when there is no activated row in the specified bank, named *row-buffer empty* access. It requires the activation of the specified row and reading or writing the target data, having higher latency than the row-buffer hit. The third scenario is when there is already an activated row in the specified bank and it is referred to as a *row-buffer miss* access. It requires closing the activated row, activating the specified row and reading or writing the data, introducing the highest latency of access.

To better optimize memory accesses, memory controller provides several row-buffer-management policies (sometimes referred to as *paging policies*). The basic ones are *open-page* and *close-page* policies. Open-page policy keeps the row in activated state after reading or writing has finished, until the memory access which targets another row in the same bank appears. It is designed to favor memory accesses to the same row of memory, usually from the memory-access sequence with a high degree of temporal and spatial locality. Since the adjacent accesses target the same row, the access latency is minimal.

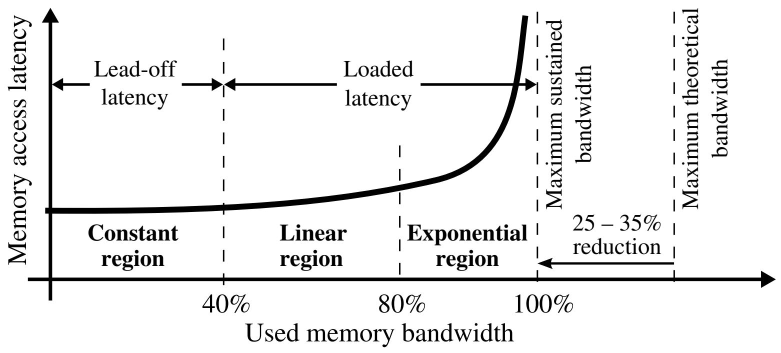


Figure 2.3: Bandwidth–latency curve showing how the memory access latency depends on the used memory bandwidth [10]. It is critical to distinguish between the *lead-off* and *loaded* memory access latency regions.

Close-page policy closes the activated row after every read or write access. It favors accesses to random locations in memory and optimizes memory request patterns with low degrees of access locality. In modern DRAM memory controllers, row-buffer-management policy is usually a dynamic combination of open-page and close-page policies. It usually comprises a countdown timer which controls how long the row remains in active state after accessing it. The timer value is dynamically adjusted based on the spatial and temporal locality of the memory access pattern.

2.2 On memory bandwidth and latency

The memory access latency and used bandwidth are often described as independent concepts, but they are in fact inherently interrelated. It is important to distinguish the *lead-off* and *loaded* memory access latencies, and to understand the connection between memory access latency and used memory bandwidth.

Lead-off memory access latency corresponds to the single-access read latency in an idle system. This latency includes the time spent in the CPU load/store queues, cache memory, memory controller, memory channel and main memory.

Loaded memory access latency corresponds to the read latency in a loaded system. In addition to all timings included in the lead-off latency, the loaded memory latency includes shared-resource contention among concurrent memory requests.

The connection between memory access latency and used bandwidth is given by the bandwidth–latency curve, as illustrated in Figure 2.3. The x

axis shows the application's used memory bandwidth, and the y axis shows the corresponding access latency. This curve has three regions that are limited by the maximum sustainable bandwidth, which is 65–75% of the maximum theoretical bandwidth [10]. In the first region, the application's used bandwidth is low, so there are few concurrent memory requests and contention for shared resources is negligible. Over this region the memory latency is constant and equal to the lead-off latency. In the second region, the application's used bandwidth is between 40% and 80% of sustainable bandwidth, and there are increasing numbers of concurrent memory requests. Contention for shared resources is moderate, and latency grows linearly with the used bandwidth. In the final region, in which the application's used bandwidth is high, contention among concurrent memory requests is severe and memory latency increases exponentially. The memory bandwidth-vs-latency curve illustrated in Figure 2.3 can be described by queuing theory as a *mean system response time as a function of request arrival rate* [51].

It is critical to distinguish between the lead-off and loaded memory access latencies, since the difference between them can be on the order of hundreds of nanoseconds. A fully-stressed memory system therefore introduces a significant loaded latency, which leads to a major performance impact.

2.2.1 Memory Wall

The term was first introduced by Wulf and McKee in 1995. They published a four-page note entitled "Hitting the Memory Wall: Implications of the Obvious" in the (unrefereed) ACM SIGARCH *Computing Architecture News* [11]. The motivation was simple: at the time, researchers were so focused on improving cache designs and developing other latency-tolerance techniques that the computer architecture community largely ignored main memory systems. The article projected the performance impact of the increasing gap between processor cycle time and memory access latency. This gap is illustrated in Figure 2.4.

The study predicted that if the trends held, even with cache hit rates above 99%, relative memory latencies would soon be so large that the processor would essentially always be waiting for memory — which amounts to "hitting the wall". This article was not the first to point out impending problems: Ousterhout had published "Why Aren't Operating Systems Getting Faster As Fast as Hardware?" [53] five years earlier. At the end of 1995, McCalpin demonstrated that current shared-memory High-Performance Computing (HPC) systems could typically sustain only 5–10% of the memory bandwidth needed to keep the floating-point pipelines busy [54], and in 1996 Burger, Goodman, and Kägi pointed out impending pin bandwidth limitations [55].

Various latency-tolerance techniques like out of order execution, wider

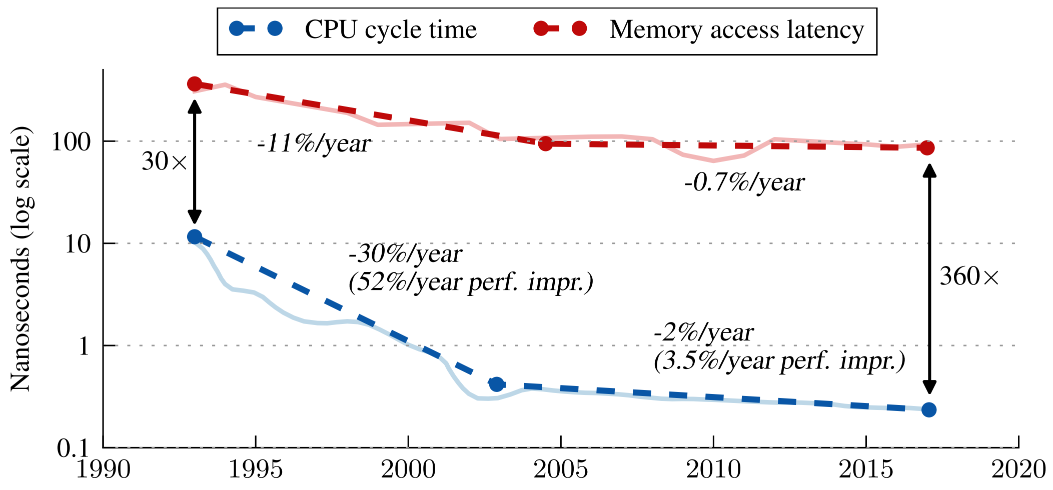


Figure 2.4: Increasing discrepancy between main memory access latency and CPU cycle time, for last 25 years. Recently the downtrend in CPU cycle time decreased to 2%, while single processor performance improves at 3.5% per year [52]. The decrement in memory access latency is less than 1%.

instruction issue, and speculative techniques such as hardware prefetching intent to bridge the processor-memory performance gap. Even in combination, though, such approaches can mask the latency of only so many memory requests, depending on the sizes of the hardware structures. The cost and complexity of implementing larger and larger structures proved prohibitive, and although latency tolerance increased the performance, it did not eliminate the memory wall. Technological evolutions and revolutions notwithstanding, the memory wall has imposed a fundamental limitation to system performance for more than 20 years.

2.3 High-bandwidth memory systems

With continued slowdown in both DRAM density and access time, it is becoming questionable whether DRAM DIMMs will continue to scale and meet the industry's demand for high performance. Significant effort is therefore being invested into the research and development of future memory systems. A promising alternative is a 3D-stacked DRAM. These devices comprise several DRAM dies that are vertically connected with *through silicon vias* (TSVs), which shorten interconnection paths and reduce connectivity impedance. Thus, data can be moved at higher rates with lower energy-per-bit. We summarize two 3D-stacked DRAM products: Hybrid Memory Cube and High Bandwidth Memory.

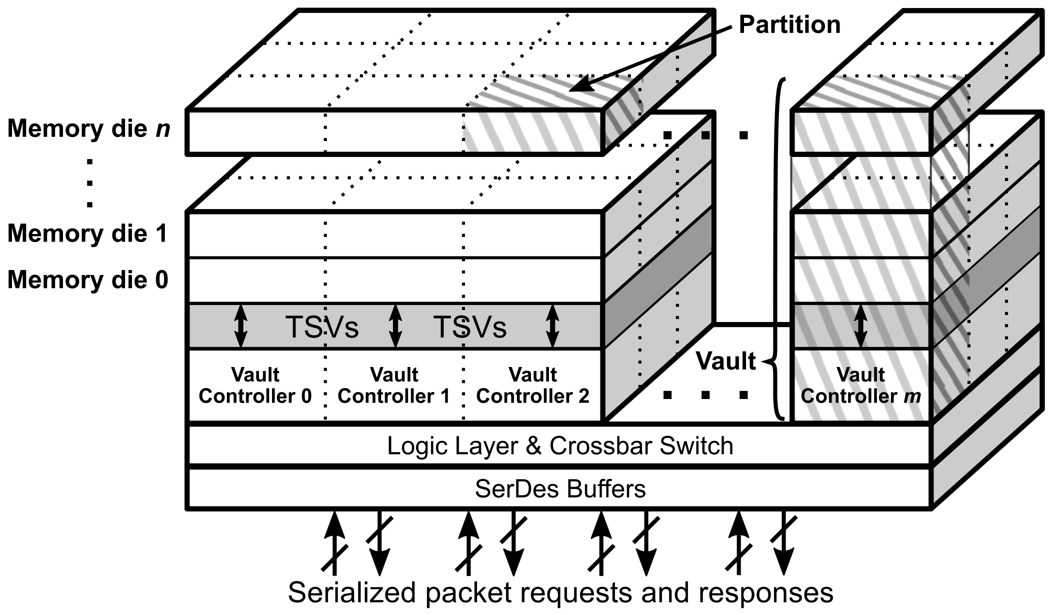


Figure 2.5: The internal structure of a Hybrid Memory Cube (HMC) [56].

2.3.1 Hybrid Memory Cube

The Hybrid Memory Cube (HMC) [21] comprises stacked DRAM dies with a unique organization. Figure 2.5 illustrates the HMC internal structure. Stacked DRAM dies (dies 0 to n) are connected with TSVs and microbumps. Each die is divided into *partitions* vertically grouped into *vaults*. Each vault operates independently through a dedicated *vault controller* resembling the memory controller in DIMM-based systems. Finally, each vault is divided into banks much as in traditional DIMMs. Announced production runs of HMC components are limited to 2 GB and 4 GB devices, while the standards specify capacities of up to 8 GB. The HMC includes a logic layer that redirects requests between off-chip serial interfaces and vault controllers. This logic layer also enables in-memory operations.

A high-speed serial interface connects the HMC to a CPU. The interface has two or four links, with each having four-, eight-, or 16-lane full-duplex serial lines. Lanes support data rates of up to 30 Gb/s, which means that per-device bandwidth can reach 480 GB/s (4 links$\times$16 lanes$\times$2$\times$30 Gbit/s).

In August 2018, Micron Technology, Inc. announced a change in its strategy for high-performance memory solutions, moving away from HMC and focusing on the next-generation of high-performance compute and networking solutions [57].

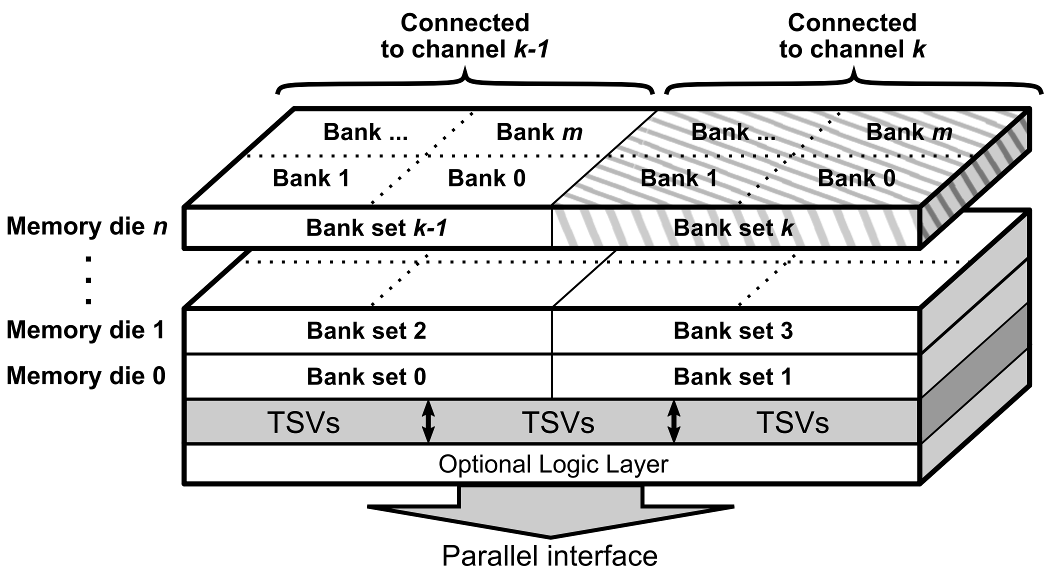


Figure 2.6: The internal structure of High Bandwidth Memory (HBM) [58].

2.3.2 High Bandwidth Memory

High Bandwidth Memory (HBM) [22] is another emerging JEDEC standard. Figure 2.6 shows its internal structure. Like HMC, HBM consists of several 3D-stacked DRAM dies connected with TSVs. The HBM memory specification allows an optional logic base layer that can redistribute signals and implement logic functions. Each die is divided into banks that are grouped and attached to channels. The channels are independent: each accesses an independent set of banks with independent clocks and memory arrays. Similarly to HMC, the standard specifies up to 8 GB devices.

Each HBM stack provides up to eight 128-bit wide channels. A 1024-bit parallel interface merges the channels. Maximum per-channel bandwidth is 32 GB/s, which implies 256 GB/s per device (eight channels$\times$32 GB/s). As with DIMMs, a system can use independent HBM devices to deliver larger overall bandwidth.

2.3.3 Comparison

Both HMC and HBM are based on 3D-stacked DRAM, which increases package density to enable higher per-chip capacity. To reduce channel latency and support high-bandwidth channels, the memory chiplets are placed on a silicon interposer instead of a printed circuit board (PCB). Both, HMC and HMB support a logic layer at the bottom of the DRAM stack; this could support in-memory computation that reduces the amount of data transferred between memory and CPU. Both devices target networking, a domain that traditionally requires high memory bandwidth. HBM also targets GPUs, while HMC targets High-Performance Computing (HPC).

HBM has a DIMM-like system organization: the memory controller is associated with the CPU, and a point-to-point parallel interface links it to the main memory. In contrast, HMC changes the system organization by placing the controller in the memory itself. Unlike DIMM-based architectures, each HMC device can directly connect to four devices via independent serial links. Device chains can thus provide an extended, high capacity memory, or even support a network of CPUs, GPUs and HMCs. In a network-like HMC system, remote HMC accesses that require multi-hop routing may have significantly higher latency, requiring an asynchronous interface. This implies higher variability in memory access times and lower timing determinism in the overall system.

A crossbar switch located in the HMC logic base routes memory requests through a network of HMC devices (see Figure 2.5). The CPU communicates with the HMC using high-level memory requests: it need not be aware of data location (device, vault, bank, row, and column) or memory-device timing parameters. Since the memory controller (i.e., vault controller) resides within the memory device, it can interact with the memory array more efficiently.

Hardware: the parts of a computer system that can be kicked.

Jeff Pesis, TRW Inc. engineer

3 Experimental environment

In this Chapter we explain experimental environment used in our research. We give a high-level view of platforms, workloads, tools and simulators used to obtain the results. Later in further chapters, we give a specific experimental methodology related to the particular study.

3.1 Hardware platforms

Throughout our research we use several HPC platforms. In the evaluation study from Chapter 4 we use a set of three mainstream and five alternative platforms; we also use Intel Sandy Bridge platform in study of bandwidth requirements of HPC applications in Chapter 5 and in modeling study from Chapter 7; in Chapter 7 we also use Intel Knights Landing (KNL) Xeon Phi platform for the additional model evaluation.

3.1.1 Set of mainstream and alternative platforms

For the last decade, the dominant HPC architectures have been Intel architectures such as Nehalem, Sandy Bridge and Haswell. Apart from these, many-core systems, of which Intel's KNL is an example, are becoming popular, while other vendors are also emerging architectures that are promising for HPC. For our evaluation study in Chapter 4, we included mainstream HPC architectures which have been predominantly used in HPC systems so far, as well as alternative architectures which have been recently introduced to the market and are set to be used in future HPC systems. The architectures under study with their most important features and used system software are summarized in Table 3.1. We evaluate and compare three generations of mainstream x86 architectures: Intel Nehalem, Sandy Bridge and Haswell,

Table 3.1: Summary of the most important features and used system software of the platforms under study

	Mainstream architectures			Alternative architectures				
Platforms	**Nehalem X5560**	**Sandy Bridge E5-2670**	**Haswell E5-2698v3**	**Knights Landing Xeon Phi 7250**	**Power8**	**ThunderX**	**X-Gene 2**	**X-Gene 1**
Manuf.	Intel	Intel	Intel	Intel	IBM	Cavium	APM	APM
Arch.	Nehalem	Sandy Bridge	Haswell	2nd gen. MIC	POWER8	ARMv8-A	ARMv8-A	ARMv8-A
Released	2009	2012	2014	2016	2014	2014	2015	2013
Sockets	2	2	2	1	2	2	1	1
Cores per Socket	4	8	16	68	10	48	8	8
CPU freq. [GHz]	2.8	2.6	2.3	1.4	3.49	1.8	2.4	2.4
Out-of-order	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
DP Flops per cycle, per core	4	8	16	32	8	2	2	2
L1i	32kB	32kB	32kB	32kB	32kB	48kB	32kB	32kB
L1d	32kB	32kB	32kB	32kB	64kB	32kB	32kB	32kB
L2	256kB	512kB	256kB	1MB	512kB	16MB	256kB	256kB
L3	8MB	20MB	40MB	/	80MB	/	8MB	8MB
Memory conf. per socket	3 ch. DDR3 1333	4 ch. DDR3 1600	4 ch. DDR4 2133	8 ch. MCDRAM[a] + 6 ch. DDR4 2400	4 ch. DMI 28.8GBps	4 ch. DDR3 1600	4 ch. DDR3 1600	4 ch. DDR3 1600
Memory capacity per node	24GB	32GB	128GB	16GB (MCDRAM) + 192GB (DDR4)	256GB	128GB	128GB	64GB
Operating system (OS)	Ocean OS[b]	SUSE Linux Enterp. Server 11	Ocean OS	Ocean OS	Ubuntu 16.04	Ubuntu 14.04	Ubuntu 14.04	Ocean OS
Compiler	Intel compiler 17.0	Intel compiler 13.0.1	Intel compiler 17.0	Intel compiler 17.0	IBM XL 13.01	GCC 6.1.0	GCC 6.1.0	GCC 4.8.5
MPI implemen.	Intel MPI	Intel MPI	Intel MPI	Intel MPI	Open MPI	Open MPI	Open MPI	Open MPI
Scientific libraries	Intel MKL	Intel MKL	Intel MKL	Intel MKL	ESSL[c]	ARM Perf. Lib.	ARM Perf. Lib.	OpenBLAS

[a] KNL system has been set to *flat mode*, therefore both memories, MCDRAM and DDR4, are exposed as separate NUMA nodes, and the user can choose in which memory the workload executes.
[b] Ocean OS is a customized CentOS 7.2 Linux distribution used in Le Commissariat à l'Énergie Atomique et aux Énergies Alternatives (CEA).
[c] ESSL stands for Engineering and Scientific Subroutine Library.

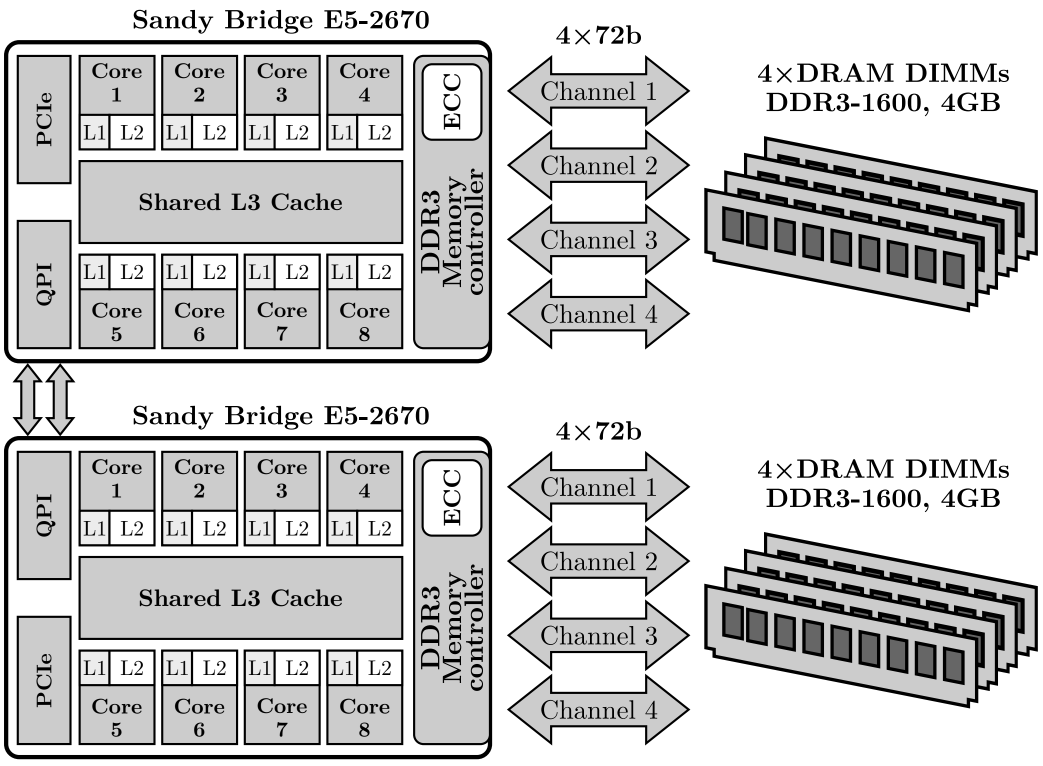


Figure 3.1: Block diagram of a single MareNostrum 3 node [59, 61].

and five alternative architectures: Intel Knights Landing Xeon Phi 7250, IBM Power8, Cavium ThunderX, and Applied Micro (APM) X-Gene 1 and X-Gene 2.

3.1.2 MareNostrum 3 supercomputer

We performed the experiments for HPC applications bandwidth requirements from Chapter 5 using the MareNostrum 3 supercomputer [59]. It is the third version of one of the six Tier-0 (largest) HPC systems in the Partnership for Advanced Computing in Europe (PRACE) [60].

MareNostrum 3 comprises 3,056 compute nodes with peak performance of 1.1 Petaflops. A sigle compute node block diagram is displayed in Figure 3.1. Every node is dual-socket, with each socket containing eight-core Intel Sandy Bridge-EP E5-2670 processor, operating at 3.0 GHz. As in most HPC systems, hyperthreading is disabled. The processors connect to main memory through four channels, each with a single DDR3-1600 DIMM. The theoretical memory bandwidth is 2 sockets×4 channels×64 bits×1.6 GT/s=102.4 GB/s, per node. The maximum sustainable memory bandwidth measured with Triad kernel from STREAM benchmark [62] is 77.86 GB/s, which is 76 % of the

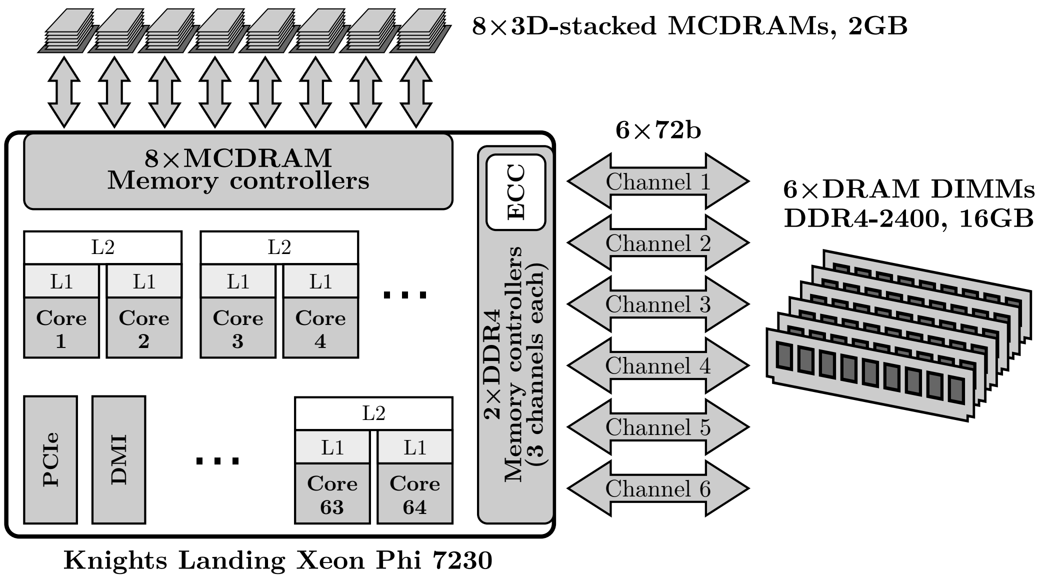


Figure 3.2: Block diagram of a Knights Landing Xeon Phi platform [63, 64].

maximum theoretical value. Regular MareNostrum compute nodes include 32 GB of DRAM memory, i.e., 2 GB per core. The nodes are connected with an InfiniBand FDR-10 (40 Gb/s) interconnect, as a non-blocking two-level fat-tree topology.

3.1.3 Intel Knights Landing Xeon Phi 7230

In Chapter 7 we evaluated our performance model using Intel Knights Landing platform [63, 65]. Figure 3.2 shows the block diagram of a KNL node. It is an alternative platform, with single socket containing Intel Xeon Phi 7230 processor, with 64 cores running at 1.3 GHz. Each core has a local L1 cache while two cores share L2 cache, without an additional third level of shared cache.

KNL combines two types of memory with different memory bandwidths and access latencies: DDR4 DIMMs and 3D-stacked Multi Channel DRAM (MCDRAM). It comprises 96 GB of DDR4-2400 memory through six memory channels, providing 115.2 GB/s of the maximum theoretical bandwidth. MCDRAM is a 3D-stacked high-bandwidth memory [66]. KNL platform comprises eight 2 GB MCDRAM chiplets (stacks), for total of 16 GB MCDRAM memory. MCDRAM has a different organization than DDR memory, and connects to the CPU through high-speed serial interface. It offers superior per-node maximum theoretical memory bandwidth of 480 GB/s, 4.2× higher than the DDR4.

Since it uses two types of memories, the system offers three modes of operation: *cache* mode, *flat* mode and *hybrid* mode. In cache mode, MC-

DRAM is used as another level of cache memory between processor and DDR memory, and it is not visible to the operating system (OS). Flat mode configures MCDRAM and DDR4 as separate NUMA nodes, so both memories are addressable and the user may choose in which memory space the code resides. In hybrid mode, the portion of MCDRAM space is used as the addressable memory, while the rest is used as a cache. In our experiments, we use *flat* mode, and we execute our workloads either in DDR4 or MCDRAM memory.

3.2 HPC workloads

This section details the workloads we used in our analyses. We executed prominent HPC benchmarks, microbenchmarks, benchmark suite and production HPC applications in our experiments.

3.2.1 HPC benchmarks

Two prominent HPC benchmarks we used in our experiments are High-Performance Linpack (HPL) and High-performance Conjugate Gradients (HPCG). Both benchmarks are used to rank supercomputers worldwide.

High-Performance Linpack

The most prominent evaluation of HPC systems constitutes the TOP500 list [2], which uses High-Performance Linpack (HPL) [67] to assess system performance. It measures the sustained floating-point rate (GFLOPs/s) for solving a dense system of linear equations using double-precision floating-point arithmetic. Since the problem is very regular, the achieved performance is quite high, and the performance numbers give a good correction of theoretical peak performance. The linear system is randomly generated, with a user-specified size, so that the user can scale the problem to achieve the best performance on a given system. The documentation recommends setting a problem size that uses approximately 80% of the available memory.

High-Performance Conjugate Gradients

Since HPL stresses only the system's floating point performance, without stressing other important contributors to overall performance, such as the memory subsystem, the community has advocated for a way to evaluate HPC systems that is better correlated with the needs of production HPC applications [68]. High-Performance Conjugate Gradients (HPCG) [69], has been released as a complement to the FLOPs-bound HPL. It is based on an iterative sparse-matrix conjugate gradient kernel with double-precision floating-point values. While HPL can exploit data locality and thus cope with relatively low memory bandwidth, HPCG performance is largely proportional to the available memory bandwidth [70]. HPCG is a good representative of HPC applications governed by differential equations, which tend to have much stronger needs for high memory bandwidth and low latency, and tend to access data using irregular patterns.

3.2.2 Microbenchmarks

We used three microbenchmarks to measure specific performance metrics: double-precision general matrix multiplication (DGEMM), STREAM and LMbench. DGEMM and STREAM are a part of a larger HPC Challenge (HPCC) benchmark suite [71].

DGEMM

DGEMM is a floating-point intensive benchmark that represents the corresponding Level 3 Basic Linear Algebra Subprograms (BLAS) routine. The benchmark calculates the product of dense double precision matrices: $C \leftarrow \alpha A \times B + \beta$. It is used for measuring the sustainable FLOP performance, at the *per-core* or *per-node* level.

STREAM

The STREAM benchmark [62] performs operations on arrays that are several times larger than the last level cache, effectively measuring the system's sustained memory bandwidth. It comprises four kernels: Copy, Add, Scale and Triad. In our analysis, we report the results of the Triad operation, since it is the most similar to kernels used in HPC applications.

LMbench

LMbench [72] is a microbenchmark suite designed to focus attention on the basic building blocks of many common system applications, such as databases, simulations, software development, and networking. It contains several benchmarks which measure performance of different hardware and software components in a system. We used *lat_mem_rd*, the read memory access latency benchmark, in order to measure access latencies of different levels in memory hierarchy. The benchmark reads the input dataset in a random order to mitigate the impact of the data prefetching. By varying the input load size, we measure access latency of all memory hierarchy levels.

The measured latency is reported in nanoseconds per load, and comprises not only the latency of the hardware components (caches, memory controller, main memory), but also the latency of the system software such as virtual-to-physical memory translation.

3.2.3 SPEC CPU2006 benchmark suite

SPEC CPU2006 benchmark suite [73] is SPEC's industry-standardized, CPU-intensive benchmark suite, stressing a system's processor, memory subsystem and compiler. It provides single-thread workloads developed from real user applications, which measure compute-intensive performance across the widest practical range of hardware. The suite includes 12 integer and 17 floating-point benchmarks. Table 3.2 lists the SPEC CPU2006 benchmarks with their application areas. We used reference workload for each SPEC CPU2006 benchmark execution.

3.2.4 UEABS HPC applications

Besides HPC benchmarks, we used production HPC applications from Unified European Application Benchmark Suite (UEABS) [74]. UEABS represents a set of production applications and datasets, from various scientific domains, designed for benchmarking the European HPC systems. It is included in the Partnership for Advanced Computing in Europe (PRACE) [60], for procurement and comparison purposes. Parallelized using the Message Passing Interface (MPI), these applications are regularly executed on hundreds to thousands of cores. We study 9 of 12 applications from UEABS.[1] Here is the brief description about each of the applications used in the thesis:

- ALYA is a computational mechanics code for solving different physics problems: convection-diffusion reactions, incompressible flows, compressible flows, turbulence, bi-phasic flows and free surface, excitable media, acoustics, thermal flow, quantum mechanics and solid mechanics.

- BQCD is a hybrid Monte-Carlo code that simulates Quantum Chromodynamics with dynamical standard Wilson fermions. The computations take place on a four-dimensional regular grid with periodic boundary conditions. The kernel is a standard conjugate gradient solver with even/odd pre-conditioning.

[1] We could not finalize the installations of Code_Saturne and GPAW. The errors have been reported to the application developers. The remaining SPECFEM3D application had problems once the measurement infrastructure was included.

Table 3.2: SPEC CPU2006 benchmarks

Benchmark	Language	Application area
Integer benchmarks		
astar	C++	Path-finding Algorithms
bzip2	C	Compression
gcc	C	C Compiler
gobmk	C	Artificial Intelligence
h264ref	C	Video Compression
hmmer	C	Search Gene Sequence
libquantum	C	Quantum Computing
mcf	C	Combinatorial Optimization
omnetpp	C++	Discrete Event Simulation
perlbench	C	Programming Language
sjeng	C	Artificial Intelligence
xalancbmk	C++	XML Processing
Floating-point benchmarks		
bwaves	Fortran	Fluid Dynamics
cactusADM	C, Fortran	General Relativity
calculix	C, Fortran	Structural Mechanics
dealII	C++	Finite Element Analysis
gamess	Fortran	Quantum Chemistry
GemsFDTD	Fortran	Computational Electromagnetics
gromacs	C, Fortran	Molecular Dynamics
lbm	C	Fluid Dynamics
leslie3d	Fortran	Fluid Dynamics
milc	C	Quantum Chromodynamics
namd	C++	Molecular Dynamics
povray	C++	Image Ray-tracing
soplex	C++	Linear Programming
sphinx3	C	Speech recognition
tonto	Fortran	Quantum Chemistry
wrf	C, Fortran	Weather modeling
zeusmp	Fortran	Fluid Dynamics

- CP2K performs atomistic and molecular simulations of solid state, liquid, molecular and biological systems. It provides a general framework for different methods such as density functional theory using mixed Gaussian and plane waves approach, and classical pair and many-body potentials.

- GADGET is a code for cosmological N-body/SPH simulations on massively parallel computers with distributed memory. GADGET computes gravitational forces with a hierarchical tree algorithm and represents fluids by means of smoothed particle hydrodynamics. The code can be used for studies of isolated systems, or for simulations that include the cosmological expansion of space, either with, or without, periodic boundary conditions. In all these types of simulations, GADGET follows the evolution of a self-gravitating collisionless N-body system, and allows gas dynamics to be optionally included. Both the force computation and the time stepping of GADGET are fully adaptive, with a dynamic range that is, in principle, unlimited. GADGET can therefore be used to address a wide array of astrophysics interesting problems, ranging from colliding and merging galaxies, to the formation of large-scale structure in the Universe. With the inclusion of additional physical processes such as radiative cooling and heating, GADGET can also be used to study the dynamics of the gaseous intergalactic medium, or to address star formation and its regulation by feedback processes.

- GENE is a gyro kinetic plasma turbulence code. Originally used for flux-tube simulations, today GENE also operates as a global code, either gradient- or flux-driven. An arbitrary number of gyro kinetic particle species can be taken into account, including electromagnetic effects and collisions. GENE is, in principle, able to cover the widest possible range of scales, all the way from the system size (where nonlocal effects or avalanches can play a role) down to sub-ion-gyroradius scales (where ETG or micro tearing modes may contribute to the transport), depending on the available computer resources.

- GROMACS performs molecular dynamics, i.e. simulate the Newtonian equations of motion for systems with hundreds to millions of particles. It is primarily designed for biochemical molecules like proteins, lipids and nucleic acids that have a lot of complicated bonded interactions, but since GROMACS is extremely fast at calculating the nonbonded interactions (that usually dominate simulations) many groups also use GROMACS for research on non-biological systems, e.g. polymers.

- NAMD is a widely used molecular dynamics application designed to simulate bio-molecular systems on a wide variety of compute platforms. In the design of NAMD particular emphasis has been placed on scalability when utilizing a large number of processors. The application can read a wide variety of different file formats, for example force fields, protein structure, which are commonly used in bio-molecular science. Deployment areas of NAMD include pharmaceutical research by academic and industrial users. NAMD is particularly suitable when the interaction between a number of proteins or between proteins and other chemical substances is of interest. Typical examples are vaccine research and transport processes through cell membrane proteins.

- NEMO is a state-of-the-art modeling framework for oceanographic research, operational oceanography seasonal forecast and climate studies. Prognostic variables are the three-dimensional velocity field, a linear or non-linear sea surface height, the temperature and the salinity. In the horizontal direction, the model uses a curvilinear orthogonal grid and in the vertical direction, a full or partial step z-coordinate, or s-coordinate, or a mixture of the two.

- Quantum Espresso is an integrated suite of codes for electronic-structure calculations and materials modeling, based on density-functional theory plane waves, and pseudopotentials (norm-conserving, ultrasoft, and projector-augmented wave). Quantum Espresso builds upon newly restructured electronic-structure codes that have been developed and tested by some of the original authors of novel electronic-structure algorithms and applied in the last twenty years by some of the leading materials modeling groups worldwide.

The applications come with input datasets and a recommended range of CPU cores for the experiments. We use the *Test Case A* datasets, which are deemed suitable for Tier-1 systems up to about 1,000 cores [74]. In all experiments, we execute one application process per CPU core. The applications under study with their area of science and input dataset description are listed in Table 3.3.

3.3 Tools

To obtain the measurements in our experiments, we use MPI instrumentation tools and performance measurement tools. We also used CPU and DRAM simulators in order to compare the estimations of our model with simulation results, described in Chapter 7.

3.3.1 Instrumentation and profiling tools

In our analysis of HPC applications we used Extrae and Limpio instrumentation tools, Paraver visualization tool and LIKWID tool for measuring performance counter metrics.

Extrae

Extrae [75] is a dynamic instrumentation package to trace programs compiled and executed with the shared memory model (such are OpenMP and pthreads), the message passing (MPI) programming model or both programming models

Table 3.3: Scientific HPC applications used in the thesis

Name	Area of science	Input problem Size
ALYA	Computational mechanics	27 million element mesh
BQCD[a]	Particle physics	$32^2 \times 64^2$ lattice
CP2K	Computational chemistry	Energy calculation of 1024 waters
GADGET	Astronomy and cosmology	135 million particles
GENE	Plasma physics	Ion-scale turbulence in Asdex-Upgrade
GROMACS	Computational chemistry	150000 atoms
NAMD	Computational chemistry	2×2×2 STM Virus replication
NEMO	Ocean modeling	1/12° global configuration; 4322 × 3059 grid
Quantum Espresso	Computational chemistry	112 atoms; 21 iterations

[a] Quantum Chromo-Dynamics (QCD) is a set of five kernels. We study Kernel A, also called Berlin Quantum Chromo-Dynamics (BQCD), which is commonly used in QCD simulations.

(different MPI processes using OpenMP or pthreads within each MPI process). It takes advantage of multiple interposition mechanisms to add monitors into the application and captures time-stamped events, e.g. entry/leave of a MPI function call. Also, it provides support for gathering additional statistics such as performance counters values at each sampling point. The result is a trace file of a parallel application for a post-mortem analysis.

Limpio

Limpio [76] is a lightweight MPI instrumentation framework. It overrides standard MPI functions and executes instrumentation routines on entry/leave of the selected MPI calls. Users themselves can write and customize the instrumentation routines and invoke external application profiling tools to fit the requirements of the analysis. Limpio can generate application traces of timestamped events that can be visualized by general-purpose visualization tools or libraries. We used Limpio to instrument HPC applications in specific region of interest, i.e., several iterations of the main computational phase.

Paraver

Paraver [77] is a flexible parallel program visualization and analysis tool. It is developed for a qualitative and quantitative analysis of the parallel application behavior by visual inspection. It supports trace filtering and trace visualization in terms of timelines, histograms and a large set of available performance metrics. We used Paraver to display trace files generated by Extrae, and determine the region of interest for HPC applications analysis.

LIKWID

LIKWID ("Like I Knew What Im Doing") [78] is a set of command-line utilities for probing the thread, cache and NUMA topology in multicore and multisocket nodes, enforcing thread-core affinity in a multithreaded applications, measuring performance counter metrics, etc. Currently it supports only x86-based processors. We used it to measure hardware performance metrics, since it easy to install and use, and covers a wide range of core and uncore performance counters of the CPU architectures used in the thesis.

3.3.2 Simulators

In Chapter 7 we compare our performance model with the simulation results. We chose ZSim as a CPU simulator, together with DRAMSim2 for the main memory simulation.

ZSim

ZSim [79] system simulator is developed by researchers from MIT and Stanford University, designed for simulation of large-scale x86-64 systems. It is organized to scale well (almost linearly) with simulated core count. ZSim uses DBT-based instruction-driven core timing models, the bound-weave parallelization algorithm, and a lightweight virtualization for complex workload support. It is also one of the fastest systems simulators available, with speed of hundreds of millions of instructions/second in a modern multicore host.

ZSim was initially designed to simulate Intel Westmere architecture (released in 2008). Since we wanted to simulate Intel Sandy Bridge-EP E5-2670 processor, one of the tasks that we had to perform was to upgrade and validate ZSim for Intel Sandy Bridge processor [80]. The ZSim upgrade was done by following the Intel documentation [81], and it comprised several steps. First, we adjusted the latency of numerous instructions, and added support the for the new x86 vector instruction extensions i.e. AVX, SSE3, that are supported by Sandy Bridge and were not supported by Westmere. We also improved the fusion of the instructions into a single micro-op, and we increased the number of entries in the reorder buffer from 128 (Westmere) to 168 (Sandy Bridge). Finally, the simulated hardware platform comprises a detailed model of Sandy Bridge-EP E5-2670 cache hierarchy [82]. In all three levels of cache memory, we used the Least Recently Used (LRU) cache replacement policy and for the L3 cache level we implemented the slice allocation hash function explained by Maurice et al. [83].

DRAMSim2

DRAMSim2 [84] is a cycle accurate model of a DRAM main memory. It is developed by University of Maryland and it is validated against manufacturer Verilog models. All major components in a modern memory system are modeled as their own respective objects within the source code, including: ranks, banks, command queue, the memory controller, etc. DRAMSim2 can be integrated with various CPU simulators through fairly simple interface.

Pretend that you're Hercule Poirot: Examine all the clues, and deduce the truth by order and method.

LaTeX error message

4 Memory bandwidth and latency aspects in HPC systems evaluation

Increasing diversity of servers with different characteristics and architectures causes their evaluation and comparison in terms of HPC features to become complex and multi-dimensional. In this chapter, we share our experience of evaluating a diverse set of HPC systems, consisting of three mainstream and five alternative architectures. We evaluate the performance and power efficiency using prominent HPC benchmarks, High-Performance Linpack (HPL) and High Performance Conjugate Gradients (HPCG), and expand our analysis using publicly available specialized kernel benchmarks, targeting specific system components. In addition to a large body of quantitative results, we emphasize four usually overlooked aspects of the HPC platforms evaluation, related to memory bandwidth and latency, and share our conclusions and lessons learned. We believe that our analysis will improve the evaluation and comparison of HPC platforms, making a first step towards a more reliable and uniform methodology.

4.1 Introduction

Each year we see a greater variety of HPC systems in the market. In addition to mainstream x86 architectures, emerging architectures based on POWER, ARM, SPARC and others are steadily appearing and catching attention [85]. Instead of hosting a single type of platform, supercomputing centers already provide a diverse set of systems. Making the right choice of architecture is critical, but evaluating and comparing HPC systems is hard.

Our study evaluates and compares three generations of mainstream x86 architectures: Intel Nehalem, Sandy Bridge and Haswell, and five alternative architectures: Intel Knights Landing (KNL), IBM Power8, Cavium ThunderX, and Applied Micro (APM) X-Gene 1 and X-Gene 2. In addition to presenting a large body of quantitative results, we emphasize four usually overlooked aspects of HPC platform evaluation, related to memory bandwidth and latency, and share our conclusions and lessons learnt.

First, we show that a platform's performance and energy-efficiency depend significantly (n-fold) on the characteristics of the target applications. For example, the ThunderX platform has 50% better energy efficiency than Haswell when running memory-bound HPCG, but Haswell shows 3.6× better efficiency for the compute-intensive HPL. We strongly advocate that any comparison between the platforms should start with the performance and energy-efficiency of HPL and HPCG as the boundaries of the compute-intensive and memory-intensive HPC applications. However, most of the previous studies [86, 87, 88] that compare alternative and mainstream HPC platforms do not include these results.

Second, we detect a significant range in the main memory access latency, with a factor of three difference between the fastest and slowest platforms under study (90 ns–285 ns). Since memory access latency has a direct performance impact for many applications [11], it should be minimized in HPC servers, and any increment above about 100 ns should be analysed and justified.

Third, we also analyse the Byte/FLOP ratio and detect a huge difference of up to 21× among the platforms under study. The Byte/FLOP ratio is one of the most important design decisions, and we hope that our results will resurface a discussion on its desired range.

Fourth, our measurements show significant, up to 70% differences between theoretical and sustained FLOPS performance and memory bandwidth, especially for alternative platforms. Therefore, we note the importance of measuring performance using specialized kernel benchmarks rather than relying on theoretical numbers from datasheets, even for the first order evaluation of the system. Also, hopefully these results will motivate further development of the HPC compilers and scientific libraries for alternative platforms.

In summary, given the substantial investment of time and money to deploy an HPC system, it is important to carefully evaluate and compare the available mainstream and alternative architectures. The conclusions of such an analysis depend significantly on the applied methodology, and the previous studies report the findings based on different experimental set-up, statistics of interest and benchmarks. Overall, we believe that this study will improve the evaluation and comparison of HPC platforms, making first steps towards more reliable and uniform methodology.

4.2 Experimental environment

In this section, we explain efforts in evaluation of HPC systems, together with workloads and experimental platforms we used in our analysis.

4.2.1 HPC benchmarks

HPC benchmarks are important for bounding the sustainable performance of different components in a system. In our study, we used prominent HPC benchmarks, HPL and HPCG (see Section 3.2.1). Apart from them, we used DGEMM and STREAM benchmarks from HPCC suite, and memory read latency benchmark from LMbench suite (see Section 3.2.2).

4.2.2 HPC platforms

For our study, we included mainstream HPC architectures which have been predominantly used in HPC systems so far, as well as alternative architectures which have been recently introduced to the market and are set to be used in future HPC systems. We evaluate and compare three generations of mainstream x86 architectures: Intel Nehalem, Sandy Bridge and Haswell, and five alternative architectures: Intel Knights Landing (KNL), IBM Power8, Cavium ThunderX, and Applied Micro (APM) X-Gene 1 and X-Gene 2. The architectures under study with their most important features and used system software are summarized in Table 3.1.

Comparing different HPC architectures under study is challenging. Architectures developed by different vendors essentially have different Instruction Set Architectures (ISAs) and therefore different system software such as compilers and scientific libraries. For each platform, we identified system software that provided the best performance. It has been used as is, and has not been tuned for each of the platforms. Hence, our conclusions should not be understood as a comparison between different hardware (CPUs and memory only), but a comparison of the platforms (systems) that also include the corresponding system software.

To our knowledge, there are no studies which analyze this many platforms, three mainstream and five alternative ones. Unlike some of the previous studies [86, 89, 90] which performed first-order evaluation of the alternative platforms by using their developer kits, all the platforms under study are fully-fledged production servers that could be used in an HPC system. We argue that it is important to compare fully-fledged servers since their performance features and power consumption differ significantly from the corresponding developers kits.

4.2.3 Power measurements

For all platforms under study we measure the power consumption at the server level, which may comprise multiple sockets, as detailed in Table 3.1. The power measurements are performed with on-board or external power meters, that account for the overall server consumption including CPUs, memory, power supply, and so on. We also used power measurements to calculate the power efficiency of the platforms under study for HPL and HPCG benchmarks.

4.3 Results

In this section, we present results from the evaluation of multiple HPC architectures with different benchmarks. We start with the most prominent HPC benchmarks, HPL and HPCG, and later expand the analysis to include other benchmarks, which together give a more complete picture of a system's performance.

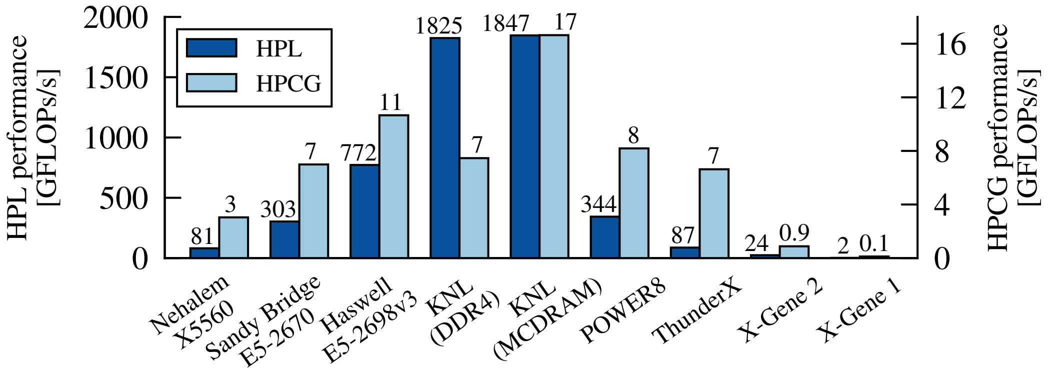
4.3.1 HPL and HPCG benchmarks

This section gives insights on the performance and power efficiency of platforms under study, while executing the HPL and HPCG benchmarks.[1] Figures 4.1a and 4.1b show the HPL and HPCG performance and performance-per-watt of platforms under study. These measurements are obtained with multi-threaded version of the benchmarks that use all the available physical cores.

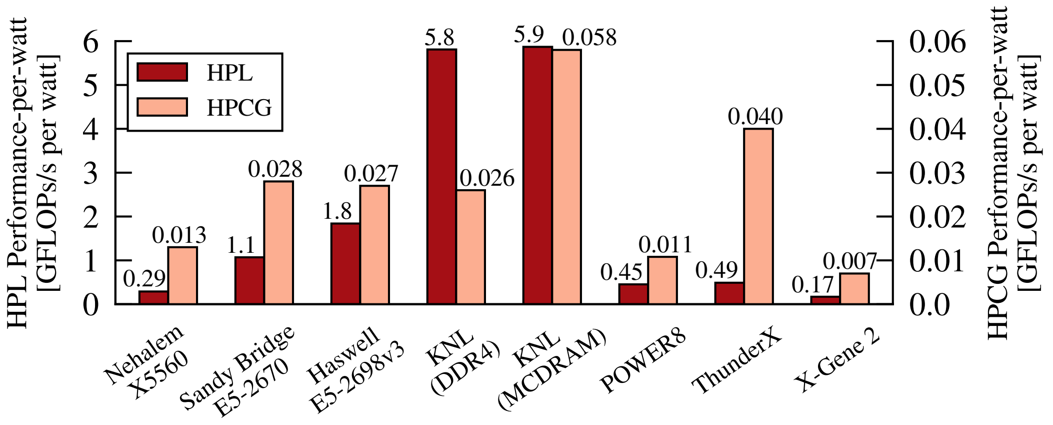
Figure 4.1a lists the performance measurements. The KNL platform shows by far the best performance for HPL, followed by the Haswell and POWER8 servers, which reach 42% and 18% of the KNL's HPL performance. ARM platforms show significantly lower performance. ThunderX, X-Gene 2 and X-Gene 1 deliver 5%, 1.3% and 0.1% of the KNL HPL scores, respectively. The results also show the notable improvement in the HPL performance over the various generations of mainstream platforms, from Nehalem to Sandy Bridge and Haswell.

The HPCG results show a slightly different trend. KNL using MCDRAM is still the highest-ranked platform, followed by Haswell, POWER8, Sandy Bridge and ThunderX. We see that the relative difference in the HPL performance is much higher than the difference in HPCG performance. The Figure 4.1a also indicates that POWER8, Haswell and KNL platforms with

[1] For X-Gene 1 platform, we could not obtain power measurements.



(a) HPL and HPCG performance



(b) HPL and HPCG performance-per-watt

Figure 4.1: The gap in performance between the platforms under study is much lower for HPCG than for HPL. Power efficiency of the alternative platforms is promising for memory-intensive workloads, such as HPCG.

DDRx memory interfaces reached plateau in terms of the HPCG performance, while KNL with MCDRAM provides a huge leap forward.

Figure 4.1b lists the performance-per-watt results for the studied platforms. Mainstream platforms show increasing power efficiency for HPL, with KNL as the best. POWER8, ThunderX and X-Gene 2 show significantly lower energy efficiencies, at just 7.6%, 8.3% and 2.9% of the KNL's performance-per-watt. HPCG power efficiency increases from Nehalem to Sandy Bridge and then stagnates for Haswell and KNL using DDR4. On the other hand, KNL using MCDRAM achieves the highest power efficiency. Alternative platforms show a much lower power efficiency, except for the ThunderX platform, which is the second best, only 31% lower than KNL using MCDRAM.

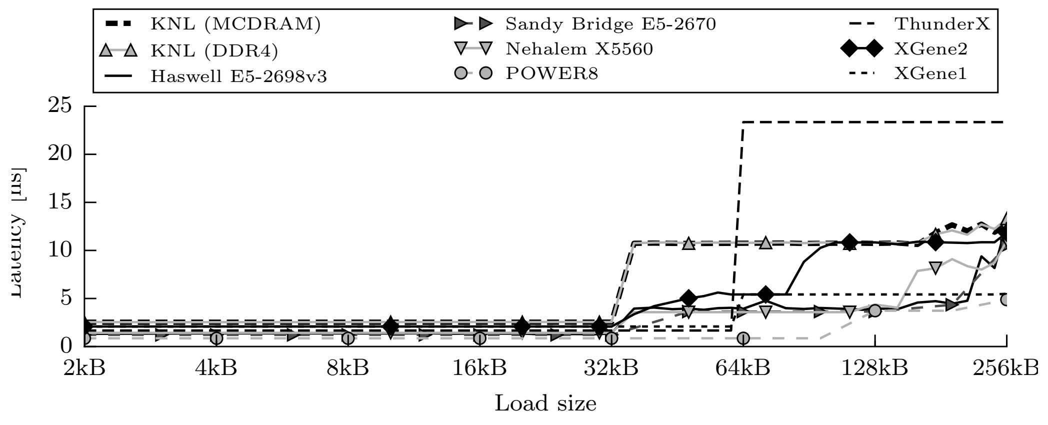
The results show that, regarding power efficiency, it is very important to identify the target application. When targeting floating-point intensive applications (such as HPL), using low-power/low-performance cores seems not to be the best approach for overall energy efficiency. However, when targeting applications with lower processing requirements and higher stress to other resources such as main memory, the ThunderX approach may deliver the energy efficiency, which significantly exceeds the x86 and POWER8 platforms.

4.3.2 Caches and main memory access latency

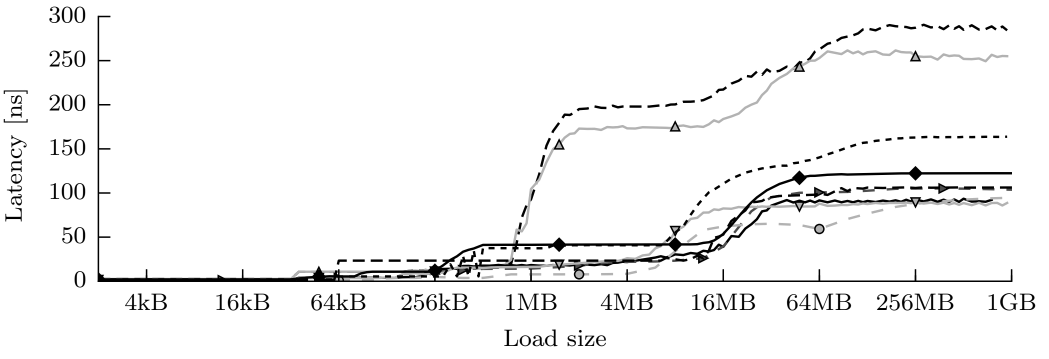
For decades, the memory system has imposed a fundamental limitation on system performance. This is recognized by the HPC community: HPL scores are frequently complemented by HPCG performance; sustained memory bandwidth is one of the main HPC performance metrics [71][91], and high-bandwidth memory solutions caused a lot of interest by the HPC users. However, although the community invests significant effort to understand the memory bandwidth, the cache and main memory latencies are usually overlooked. This is surprising because the memory latency has a direct performance impact, and the memory wall itself was defined in terms of latency, not bandwidth [11].

In this section, we compare the access time of the caches and main memory for platforms under study. The results are plotted in Figure 4.2. The x-axis of the figures shows the input dataset size. In Figure 4.2a the load size ranges from 2 KB to 256 KB, which focuses on the L1 and L2 caches. In Figure 4.2b dataset size reaches up to 1 GB, covering all levels of caches and main memory. Even for the L1 and L2 caches we detect a significant difference in the latencies. At the L1 cache (2–32 KB load) the latency varies from 1.25 ns (Sandy Bridge, Haswell, POWER8) to 2.5 ns for KNL. In the L2 cache (128 KB load), the difference is even more significant, from 3.6 ns (Nehalem, Sandy Bridge, Haswell, POWER8) to 23 ns (ThunderX). The main memory latency (256 MB load) ranges from 90 ns (Nehalem, Haswell and POWER8) and 105 ns (Sandy Bridge, ThunderX) up to 250 ns and 285 ns for KNL using DDR4 and MCDRAM, respectively.

Overall, our measurements show that the cache and main memory latencies can vary significantly among platforms. Mainstream platforms and POWER8 perform well on all memory hierarchy levels. Alternative ARM platforms, ThunderX and X-Gene, have somewhat higher latency. KNL has significantly higher latency especially for the datasets that exceed 1 MB. Since these latencies have a direct performance impact, especially for the workloads with a high rate of dependent memory accesses, they are an important parameter to consider. KNL platform is especially interesting since it incorporates high-bandwidth MCDRAM, based on 3D-integration. While KNL delivers memory bandwidth far superior to any other platform under study, it comes with the cost of the memory access latency that exceeds 3× the latency on mainstream



(a) L1 and L2 cache access latency



(b) L1, L2, L3 and main memory access latency

Figure 4.2: Cache and main memory latency can vary significantly among the studied platforms. KNL memory access latency exceeds 3× the latency on other platforms.

platforms. Finally, it is also important to notice that most of the KNL memory access penalty does not come from the memory device itself. DDR3 and DDR4 modules timing parameters are standardized by JEDEC [92], and the variation between them (in nanoseconds) is negligible. Still, KNL DDR4 access is around 150 ns slower than other platforms. Therefore, the KNL memory access penalty originates mainly from handling the memory request between the last-level cache and the memory device, i.e. from the memory queues and memory controller. It is interesting to see whether future architectures will succeed in incorporating 3D-stacked memory without a significant latency overhead.

4.3.3 Byte/FLOP ratio

Using the node-level measurements of FLOPs and memory bandwidth (measured with DGEMM and STREAM benchmarks), in Figure 4.3 we show the

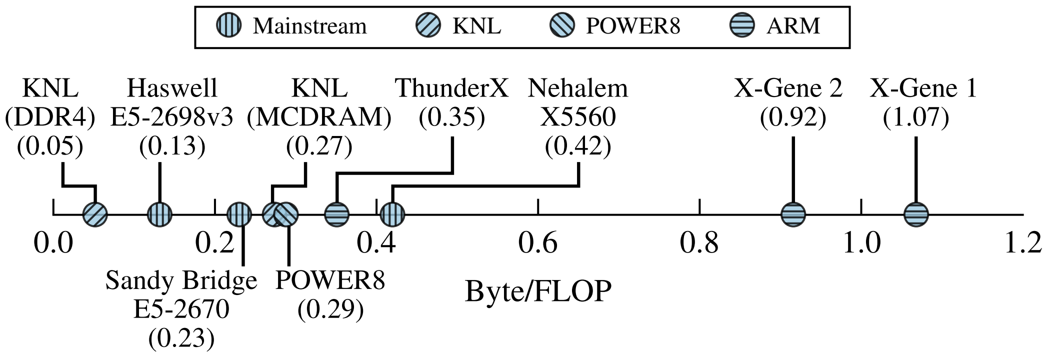


Figure 4.3: Ratio between sustained memory bandwidth and FLOPS of the platforms under study can differ up to 21×.

ratio between sustained memory bandwidth and FLOPS of the platforms under study. Platforms with a low Byte/FLOP ratio are well suited for compute-intensive applications such as HPL. In these platforms, for real applications memory bandwidth may easily become a performance bottleneck. The platforms with a high Byte/FLOP ratio perform well with applications that put a high pressure on memory bandwidth, such as HPCG. In this case, floating-point processing power may limit the performance.

We detect a huge difference in the Byte/FLOP ratio among the platforms under study. The measured Byte/FLOP ratio ranges from 0.05 (KNL-DDR4) to 1.07 (X-Gene 1), a difference of more than 21×. For mainstream HPC systems (Nehalem, Sandy Bridge and Haswell), the Byte/FLOP ratio is significantly below 1, and it has the tendency of decreasing [41], which does not serve well for memory-bound HPC workloads. Also, current DDRx technology cannot keep up with aggressive FLOPs performance increases, so further progress in memory bandwidth relies on high-bandwidth memory solutions based on 3D-integration. In this respect, the KNL platform has a much higher Byte/FLOP ratio using MCDRAM than DDR4. Alternative systems, on the other hand, show a promising ratio, which is higher than mainstream platforms. This is mostly because the sustainable memory bandwidth is currently comparable between mainstream and alternative platforms, while the FLOPs performance of alternative systems is significantly below the mainstream ones. If this ratio keeps up with future developments of emerging platforms, we could see systems that cope better with memory-bandwidth intensive HPC workloads. Since HPC system performance strongly depends on the Byte/FLOP ratio, we advocate for this ratio to be constrained more precisely for HPC systems.

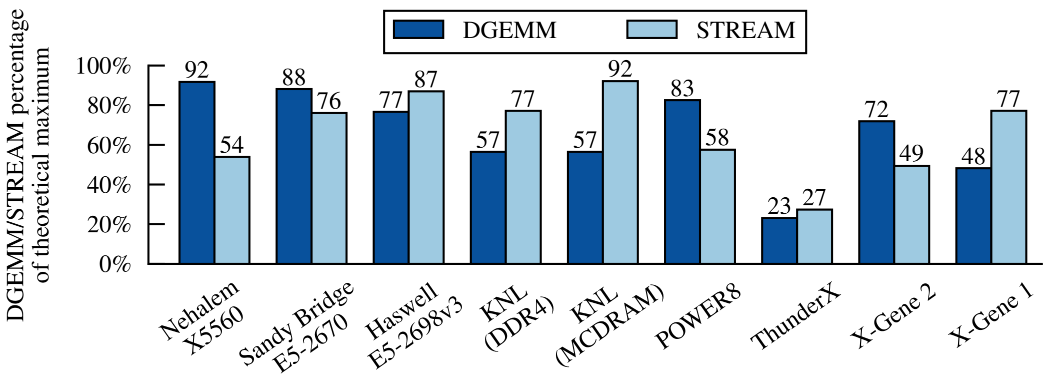


Figure 4.4: Sustained FLOPS and memory bandwidth show significant difference to theoretical maximums, especially for alternative platforms.

4.3.4 Theoretical vs. sustained FLOPs/s and memory bandwidth

As the final step of our analysis, we compare the maximum theoretical FLOPS performance and memory bandwidth from platform datasheets with the sustained values measured using DGEMM and STREAM. This comparison is important because sometimes theoretical numbers are used to compare platforms or estimate large-scale system performance before they are built. Our results, however, show that the differences between theoretical and measured numbers may be significant.

The results are displayed in Figure 4.4. Mainstream HPC systems based on Sandy Bridge and Haswell deliver sustained FLOPS performance and memory bandwidth close to theoretical maximums. Some alternative architectures, however, reach moderate FLOPS and memory bandwidth utilization even when running the DGEMM and STREAM benchmarks. For example, X-Gene 1 and KNL reach only 48% and 56% of the maximum theoretical FLOPS, while X-Gene 2 and POWER8 achieve similar rates for memory bandwidth. An explanation could be that the overall system cannot fully utilize SIMD floating-point execution units or data-transfer mechanisms. By the overall system we include both hardware and system software, including the pipeline, out-of-order (OoO) engine, caches, compilers and scientific libraries. The HPC system software for alternative platforms is still under development; for example, the first math libraries for ARM-based servers were released three years ago [93]. Similar studies confirm that system software stack on alternative platforms is relatively immature, which limits the achievable performance [88, 94, 95]. Finally, ThunderX shows very low FLOPS and memory utilization of 23% and 27%, respectively. In this case, additional problem is the simplicity of the in-order core and poor performance of inter-socket communication.

This analysis has two outcomes. Firstly, we would avoid using maximum theoretical performance even for first-order provisioning or an early evaluation of the HPC system, especially for the alternative platforms. Secondly, for some of the platforms under study, the results also show notable room for performance improvement, which will hopefully motivate further development of compilers and scientific libraries for alternative HPC platforms.

4.4 Related work

In addition to mainstream x86 architectures, emerging architectures based on POWER, ARM, and SPARC are steadily appearing and catching the attention of the HPC community. Although making the right choice of architecture is critical for the HPC infrastructure providers, only few studies evaluate and compare available alternative and mainstream HPC platforms.

The study of Rajovic et al. [86] is the first to analyze the suitability of mobile ARM processors for HPC. The study compares the performance and energy efficiency of development boards using mobile ARM 32-bit SoCs against a laptop with a Intel Sandy Bridge CPU.[2] Based on these measurements, the authors conclude that the performance and energy efficiency of mobile ARM platforms is competitive to the mainstream x86 HPC servers.

Abdurachmanov et al. [94] compare an X-Gene 1 development board with a dual-socket Intel Sandy Bridge server and Intel Xeon Phi PCIe add-on card. The authors compare only the CPU power consumption using on board sensors for X-Gene 1 development kit and Xeon Phi card, and RAPL interface [81] on the Sandy Bridge CPU. The study analyzes performance and energy efficiency of a single benchmark, ParFullCMS, and it concludes that the Sandy Bridge and Xeon Phi CPUs have similar performance that is 2.5× higher than X-Gene 1. Performance-per-watt results position Sandy Bridge as the most efficient platform, followed by X-Gene 1 (approximately 10% lower efficiency) and Xeon Phi (more than 35% lower efficiency compared to Sandy Bridge).

Early evaluation of emerging platforms using developer kits is valuable and needed. However, we argue that energy-efficiency analysis requires measurements on the fully-fledged production servers, as performed in our study.

Rajovic et al. [87] also deploy a prototype cluster with nodes based on mobile ARM 32-bit SoCs and compare it with a production HPC Sandy Bridge cluster. The study also estimates the performance of the potential successor mobile SoCs with advanced ARM cores and embedded GPUs. The

[2] In order to reduce the non-essential power consumption the authors switch off the laptop's screen.

authors conclude that alternative ARM-based systems would offer performance equivalent to mainstream x86 systems, while saving 40% energy, and achieving higher integration density. However, these conclusions are based on two non-trivial HPC application requirements. First, the HPC applications would have to fully utilize the GPUs embedded into emerging SoCs, which is not the case for most current production HPC codes. For the applications that can fully utilize the GPUs, the CPU+GPU emerging systems should be compared with similar (CPU+GPU) mainstream platforms, not with respect to the CPU-only systems. Second, the application should have perfect parallel efficiency and load balancing when scaling-out from strong x86 cores to an approximately 4× larger number of weaker ARM cores. However, scale-out of production HPC applications typically leads to significant performance penalties [17]. Finally, the authors do not consider the performance and energy impact of RAS features (RAS: Reliability, availability and serviceability), such as memory ECC, available in the contemporary HPC systems, and not available on the alternative system under study.

Laurenzano et al. [88] compare the performance, power and energy consumption, and bottlenecks of Sandy Bridge, Atom, Haswell and X-Gene 1 servers. This analysis is based on system measurements with a large number of benchmarks and statistical modeling. The authors conclude that on average, for all the benchmarks under study, the X-Gene 1 and Atom servers have comparable performance, which is significantly below the Haswell and Sandy Bridge systems. Regarding the energy efficiency, Laurenzano et al. measure similar results for the X-Gene 1 and Sandy Bridge, somewhat below the Atom and Haswell servers. For all the platforms under study, the authors perform server-level measurements, but then extract the *power resulting from executing the application* as a subtraction between the server power executing the application and the idle server power. Our position is that using this metric to quantify and compare energy efficiency is misleading and unfavorable for servers with higher energy proportionality, in which power consumption is highly correlated to server performance.

The conclusions of the studies that evaluate and compare emerging and mainstream HPC platforms depend significantly on the methodology and benchmarks used. Still, the related work shows there is no unified approach for this analysis, and that the conclusions are sometimes based on a methodology and assumptions open to discussion. In addition to a large body of quantitative results, our study emphasizes usually-overlooked and important memory bandwidth and latency aspects of the HPC platforms evaluation. We believe this will improve the evaluation and comparison of HPC platforms, making a first step towards a uniform and more reliable methodology.

4.5 Summary

In our study, we perform an extensive analysis of HPC architectures, three mainstream and five alternative ones. To the best of our knowledge this is the first study to include so many platforms. In addition to presenting a large body of quantitative results, we highlight four important features in HPC systems evaluation that require higher attention by the community.

First, we show a platform's performance and energy-efficiency depend significantly (n-fold) on the characteristics of the target application. We strongly advocate that any comparison among platforms should start with measurements using HPL and HPCG, which form the boundaries of compute-intensive and memory-intensive HPC applications.

Second, our results show a huge range of memory access latencies, from 90 ns to 285 ns for the studied platforms. While KNL with MCDRAM has the highest memory bandwidth, it also has the highest memory access latency, due to complex memory controller and its handling of memory requests. Since memory access latency has a direct performance impact any increment above about 100 ns should be analysed and justified.

Third, we detect that the Byte/FLOP ratio can differ by a factor of up to $21\times$ between platforms. While mainstream platforms show a decreasing tendency, alternative platforms trend upwards in this metric. We propose for a community to properly define this ratio for HPC applications, since it has a direct impact on system performance.

Fourth, our results show that sustainable FLOPS performance and memory bandwidth on the alternative platforms can deviate more than 70% from theoretical performance. Therefore, we strongly suggest not relying on theoretical performance, even in a first-order system provisioning. These results will hopefully motivate further development of the compilers and scientific libraries for alternative HPC platforms.

Overall, our study provides a significant body of useful information for HPC practitioners and infrastructure providers. Even more important, we believe it will considerably improve the future evaluations and comparisons of HPC platforms, making a first step towards a more reliable and uniform methodology.

In my opinion this problem of making a large memory available at reasonably short notice is much more important than that of doing operations such as multiplication at high speed.

Alan Turing, mathematician and pioneer of computer science and AI

5 Memory bandwidth requirements of HPC applications

Designing a balanced HPC system requires an understanding of the dominant performance bottlenecks. We highlighted in Chapter 4 that there is as yet no well established methodology for a unified evaluation of HPC systems and workloads that quantifies the main performance bottlenecks. This chapter analyses the methodology of quantifying key performance bottlenecks: FLOPS performance and memory bandwidth congestion, and the implications of scaling-out. We execute seven production HPC applications, together with HPL and HPCG, on a production HPC platform and reach two main conclusions. We show that the results depend significantly on the number of execution processes, which is typically overlooked in benchmark suites that seldom define how many processes should be used. We therefore advocate for guidance in the application suites, on selecting the representative scale of the experiments. Moreover, we find that the average measurements of performance metrics and bottlenecks can be highly misleading. Hence, we propose that the FLOPS performance and memory bandwidth should be represented in terms of the proportions of time with low, moderate and severe utilization. Our analysis shows that this gives much more precise and actionable evidence than the average. Finally, we believe this study offers new guidelines for accurately measuring key performance factors and their impact on overall HPC performance.

5.1 Introduction

Deploying an HPC infrastructure is a substantial investment in time and money, so it is extremely important to make the right procurement decision. Unfortunately, evaluating HPC systems and workloads and quantifying their bottlenecks is hard. There are currently three main approaches. The approach taken by TOP500 and Green500 is to evaluate systems using a prominent HPC benchmark, such as High-Performance Linpack (HPL) [67] or High Performance Conjugate Gradients (HPCG) [69]. Another approach is to measure the sustained performance of the various components in the system using specialized kernel benchmarks, such as HPC Challenge [71]. By design, kernel benchmarks quantify only the sustainable performance of individual system components, so they lack the capability to determine how a real-world production HPC application will behave on the same platform.

The final approach, which is the one taken in this chapter, is to mimic production use by running a set of real HPC applications from diverse scientific fields [96]. We execute seven production HPC applications, together with HPL and HPCG, on a production x86 platform, and we reach two main conclusions. Firstly, we find that HPC application performance and CPU/memory system bottlenecks are strongly dependent on the number of application processes. This is typically overlooked in benchmark suites, which seldom define how many processes should be used. We argue that it is essential that HPC application suites specify narrow ranges on the number of processes, so that the results are representative of real world application use, or that they at least provide some guidelines. Secondly, we find that average values of bytes/FLOP, bytes/s and FLOPs/s can be highly misleading. Our results show that the applications under study have low average FLOPs/s utilization and moderate pressure on the memory bandwidth. However, we identified several applications, such as ALYA and GENE, with a moderate *average* memory bandwidth that spend more than 50% of their computation time in phases where the memory bandwidth bottleneck is severe. We therefore recommend that rather than thinking in terms of average figures, one measures the percentage of time that the utilization of memory bandwidth or FLOPs/s is low (below 40% of sustainable maximum), moderate (40% to 80%) and severe (above 80%). These three figures give a much more precise picture of the application behavior than the average.

In summary, given the substantial investment of time and money to deploy an HPC system, it is important to carefully evaluate HPC architectures. Compared with benchmarks or kernels, system evaluation with HPC application suites can give a more complete picture of the HPC system behavior. However, our results show that it is very important that HPC application suites specify narrow ranges for the number of processes that are representative of real-life application behavior, or at least provide some guidelines so users themselves could determine these ranges for their target platforms. In addition, reporting key application measurements using the average values may conceal bursty behavior, and give a misleading impression of how performance would be affected by changes in the platform's memory bandwidth. We suggest to avoid average figures when evaluating performance or bottlenecks, and instead measure the percentage of time that these figures are low, moderate and severe, with respect to their sustained peak, which gives a more precise picture of the application's or system's behavior.

We hope our study will stimulate awareness and dialogue on the subject among the community, and lead to improved standards of evaluating and reporting performance results in HPC.

5.2 Experimental environment

In this section, we explain the experimental platform, workloads, methodology and tools we used in our analysis.

5.2.1 Experimental platform

The experiments are executed on the MareNostrum 3 supercomputer (see Section 3.1.2). In our experiments we used up to 64 nodes (1,024 processes).

5.2.2 Workloads

HPC benchmarks

For a long time, the High-Performance Linpack (HPL) benchmark (see Section 3.2.1) has been the de facto metric for ranking HPC systems. HPL stresses only the system's floating point performance, without stressing other important contributors to overall performance, such as the memory subsystem. The most prominent evaluation of HPC systems constitutes the TOP500 list [2], which has been criticized for assessing system performance using only HPL [97]. HPCG (see Section 3.2.1) has been released as a complement to the FLOPs-bound HPL. We used both benchmarks in our experiments.

HPC applications

Evaluating HPC systems using benchmarks that target specific performance metrics is not enough to determine the performance of a real-world applica-

Table 5.1: Scientific HPC applications used in the study

Name	Area	Selected no. of processes
ALYA	Computational mechanics	16–1024
BQCD[a]	Particle physics	64–1024
CP2K	Computational chemistry	128–1024
GADGET	Astronomy and cosmology	512–1024
GENE	Plasma physics	128–1024
NEMO	Ocean modeling	512–1024
QE[b]	Computational chemistry	16–256

[a] Quantum Chromo-Dynamics (QCD) is a set of five kernels. We study Kernel A, also called Berlin Quantum Chromo-Dynamics (BQCD), which is commonly used in QCD simulations.
[b] QE stands for Quantum Espresso application. QE does not scale on more than 256 processes.

tion. It is therefore essential to execute production applications on an HPC system to better understand the bottlenecks and constraints experienced by a production HPC application. There are efforts in making suites of HPC applications that could be used in benchmarking purposes, such as NSF [98], NCAR [99] and NERSC Trinity benchmarks [100] in USA, and EuroBen [101], ARCHER [102] and UEABS [74] in Europe. In our evaluation, we used a set of seven UEABS applications (see Section 3.2.4), listed in Table 5.1.[1]

Tools and methodology

In all experiments, we execute one application process per CPU core. The number of processes starts from 16 (a single MareNostrum node) and it increases by powers of two until 1,024 processes. Some of the applications have memory capacity requirements that exceed the available node memory, which limits the lowest number of processes in the experiments, e.g., BQCD cannot be executed with fewer than 64 processes (four nodes). The presented analysis keeps constant the input dataset and varies the number of application processes, which refers to a strong scaling case.[2]

[1] The remaining two applications had problems once the measurement infrastructure was included.

[2] The alternative would be a weak scaling analysis, in which the problem size scales with the number of nodes. Unlike HPL and HPCG, for which the problem size is defined by the user and the input data is generated algorithmically, application benchmark suites include specific input problem data. We are not aware of a production application benchmark suite that has problems suitable for weak scaling analysis. Although some of the UEABS benchmarks are distributed with two input datasets, small and large, they are not comparable so are insufficient for weak scaling analysis [103].

The applications were instrumented with Limpio and Extrae tools (see Section 3.3.1). We used core performance counters [81] to measure FLOPS performance (scalar and vector FLOPS counters) and uncore performance counters [61] to measure memory bandwidth (read and write Column Access Strobe (CAS) commands counters).

We analyze the application behavior at two levels of granularity. First, we plot mean FLOPs and memory bandwidth utilization using end-to-end measurements and averaging the values of all application processes. Second, we analyze the fine-granularity measurements done at the computational burst level. For each computational burst we measure the FLOPs performance and the burst execution time, while memory bandwidth utilization is measured on 1 second time period, since uncore counters do not allow reading on every computational burst. Afterwards, we strip the communication time from the memory bandwidth measurements. Finally, we analyze the cumulative distribution function of the measurements.[3] As we show in this study, these two levels of the analysis can, and often do, actually lead to different conclusions.

5.3 Results

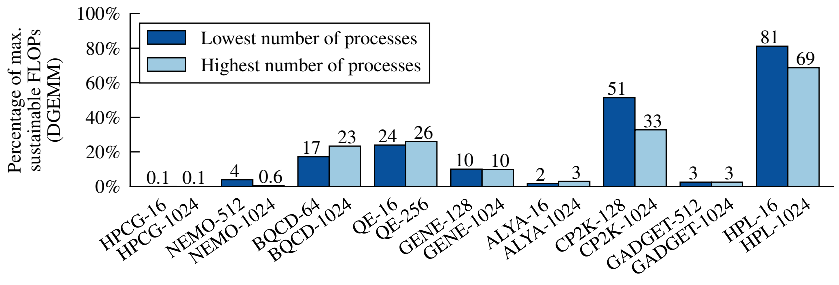
In this section, we analyze the stress of the production HPC applications on the CPU and memory resources, and pay special attention to understand how this stress may change during execution and as the application scales.

5.3.1 Floating-point performance analysis

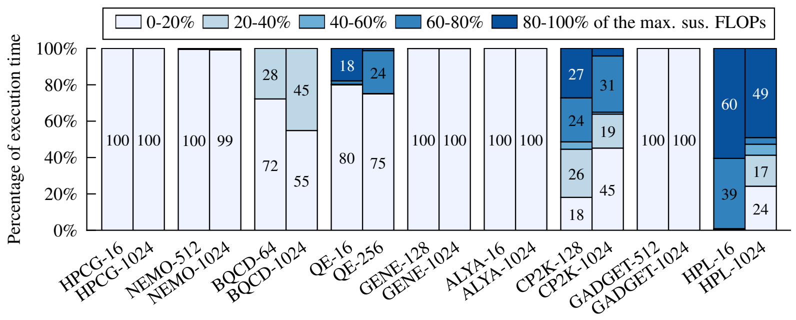
Figure 5.1a plots the average FLOPs/s utilization for different numbers of application processes. The results show that the average FLOPs/s utilization of production HPC applications is fairly low: for most applications it is below 30%, and in the best case it reaches only 51% (CP2K-128 experiment). Figure 5.1b summarizes the distribution of measurements done at computational burst level. We divide the computational burst measurements into five clusters: 0–20%, 20–40%, 40–60%, 60–80% and 80–100% of sustained FLOPs/s, and then plot the portion of execution time represented by each cluster. For example, in the BQCD-64 experiment, 72% of the time the FLOPs/s utilization is between 0 and 20%, while for the remaining 28% of the time it is between 20% and 40%.

Our results show that detailed measurements are indeed needed, and that plotting only average values may hide important information. The

[3] The cumulative distribution function, $y = F(x)$, in this case presents the fraction of samples y that are less or equal to a certain value x.



(a) Average FLOPS utilization



(b) FLOPS utilization on burst granularity

Figure 5.1: Production HPC applications show fairly low FLOPS utilization, both on lowest and highest number of processes.

most obvious case would be the QE-16 experiment. Although the average FLOP utilization is only 24% (Figure 5.1a), the application actually puts extremely high pressure on CPU FLOPs for around 18% of its computation time (Figure 5.1b).

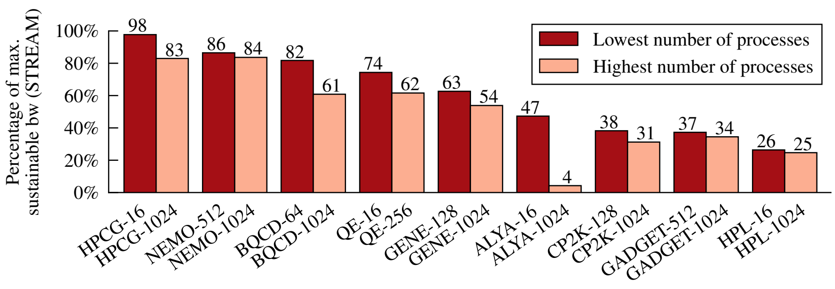
We also analyze changes in the application behavior when executing them using different numbers of processes. Both, average and per-burst measurements indicate significant changes in the application behavior as the applications scale-out[4].

This opens a very important question: Which application behavior is the correct/representative one, i.e. which number should we report?

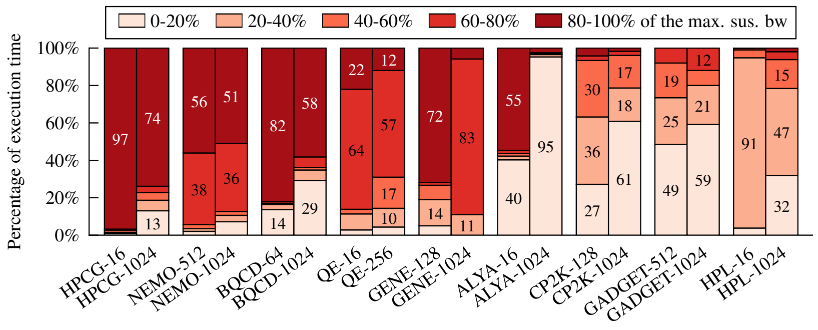
5.3.2 Memory bandwidth analysis

Memory bandwidth has become increasingly important in recent years. Keeping the memory bandwidth balanced with the CPU's compute capabilities,

[4] We remind the reader that we used the official input datasets, and followed the recommendations about the range of CPU processes that should be used in the experiments (see Section 5.2.2).



(a) Average memory bandwidth utilization



(b) Memory bandwidth utilization on burst granularity

Figure 5.2: Contrary to FLOPS, memory bandwidth utilization of production HPC applications is substantial.

within affordable costs and power constraints, has become a key technological challenge. The increasing awareness of this challenge also resulted in the introduction of the HPCG benchmark, as an alternative to HPL. The industry also responded to the growing need for more memory bandwidth, and high-bandwidth 3D-stacked DRAM products are hitting the market. Their manufacturers promise significant performance boosts over standard DDRx DIMMs, although some independent studies doubt whether and to what extent high-bandwidth memory will benefit HPC applications [104].

Memory bandwidth collision can indeed have the strong negative performance impact. When a workload uses more than 40% of maximum sustainable bandwidth, concurrent memory accesses start to collide, which increases memory latency causing performance penalties. Using more than 80% of maximum sustainable bandwidth causes severe collisions among concurrent memory requests; thus memory latency increases exponentially and memory bandwidth becomes a serious performance bottleneck [10].

Figure 5.2 plots the memory bandwidth usage of UEABS applications. The memory bandwidth values are plotted relative to the maximum sustained

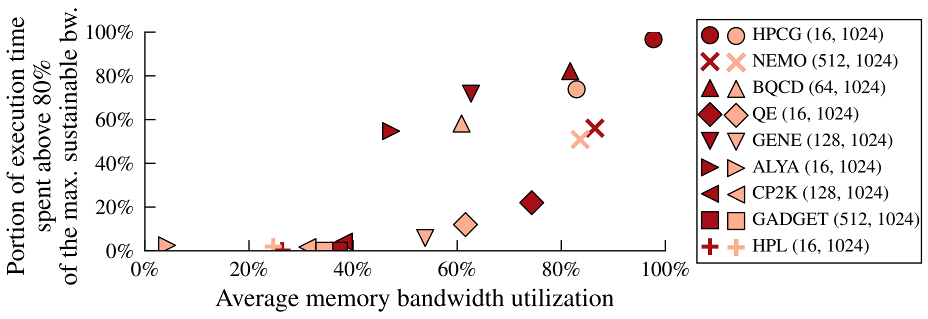


Figure 5.3: Average memory bandwidth can mislead and hide potential bottlenecks. BQCD-1024, GENE-128 and QE-256 have similar average memory bandwidths, however BQCD-1024 and GENE-128 spend significantly more time utilizing more than 80% of max. sustainable bandwidth, which is a serious bottleneck.

memory bandwidth measured by the STREAM benchmark. Again, we plot the results at two levels of granularity: Figure 5.2a plots average utilization over computation time and for different numbers of application processes, while Figure 5.2b shows fine-granularity measurements at the computational burst level. The applications under study show higher utilization of memory bandwidth, than FLOPs performance, even for the average values.

Next we analyze the computational bursts measurements, presented in Figure 5.2b. The chart shows moderate to high memory bandwidth utilization. All the applications under study have segments in which memory bandwidth utilization exceeds 40%, and all but two of them, CP2K and GADGET, spend a significant portion of time with bandwidth utilization above 60% or even 80%.

The computational burst measurements reveal some interesting scenarios, which are more apparent in Figure 5.3. In this figure, the x-axis is the *average* memory bandwidth utilization, as in Figure 5.2a. The y-axis is the proportion of time for which the memory bandwidth utilization is severe; i.e. more than 80% of the sustainable maximum, which corresponds to the darkest shade parts of the bars in Figure 5.2b. Figure 5.3 shows that considering the average memory bandwidth on the x-axis, ALYA-16 and CP2K-128 may seem to be bandwidth insensitive, as their average bandwidths are around 50% and 40% of the sustained bandwidth. However, detailed in-time measurements show that they spent significantly different proportions of the time with severe memory bandwidth utilization: CP2K-128 spends only about 4% of its computation time, but ALYA-16 spends 55% of its computation time, which presents a serious performance penalty. We find a similar situation with BQCD-1024, GENE-128 and QE-1024. These applications all have average memory bandwidth of around 60% of the sustained maximum. Even so, QE-256 spends only 12% of its computation time with severe memory bandwidth utilization (more than 80% of maximum sustained). In contrast, BQCD-1024 and GENE-128 spend 58% and 72% of their computation time, respectively, with severe memory bandwidth utilization.

This is another confirmation that detailed measurements are needed, and that plotting only the average values may be misleading. Applications under study that spend significant amount of their computation time using more than 80% of the sustained bandwidth have a severe performance bottleneck. In these phases of their computation time, the applications would benefit out of increased available memory bandwidth in the system. In our case, ALYA-16, but not CP2K-128, is likely to benefit from higher bandwidth memories. It would reduce the bottleneck and increase the application performance. However, reporting only average values of memory bandwidth cannot point out the necessary details.

Our suggestion would be that memory bandwidth utilization should be defined at least with three numbers — as the percentage of execution time that applications use 0–40%, 40–80% and 80–100% of the maximum sustained bandwidth. This would correspond the portion of the execution time in which the application experiences negligible, moderate and severe performance penalties due to collision on concurrent memory requests.

5.3.3 Discussion

Our analysis emphasizes that HPC application behavior is tightly coupled with the number of application processes. There are two main reasons for this. First, application scaling-out increases the inter-process communication time. To illustrate this, in Figure 5.4 we plot the portion of overall execution

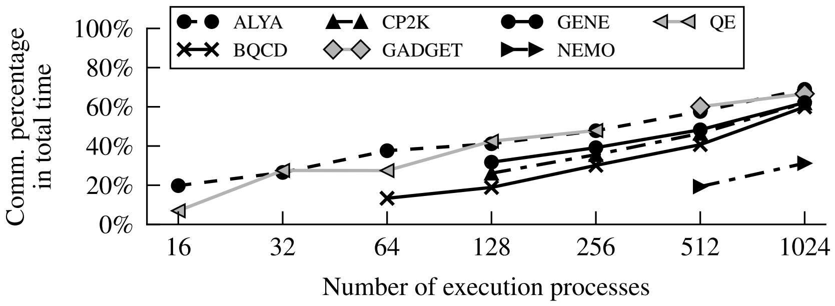


Figure 5.4: Portion of total execution time spent in the inter-process communication for UEABS applications, strong scaling.

time that applications under study spend in inter-process communication.

Even for the low number of application processes, the communication is non-negligible, and as the number of processes increases, it becomes the dominant part of the overall execution time. The higher the portion of time that is spent in communication,the lower the average utilization of FLOPs and memory bandwidth (as detected in Figures 5.1a and 5.2a). Also, in general, the higher the number of processes, the smaller the portion of the input data handled by each process, which changes the effectiveness of cache memory and the overall process behavior (as detected in Figures 5.1b and 5.2b).

HPC application behavior may be known by the application developers, but it is often overlooked in all HPC application suites for benchmarking purposes. State-of-the-art HPC application suites do not strictly define the number of processes to use in experiments. For example, UEABS recommends running the applications with corresponding input datasets on up to approximately 1,000 processes, but the minimum number of processes is not specified. Similarly, other HPC application suites either provide loose recommendations about the number of processes [98, 99, 100, 102] or do not discuss this issue at all [101]. However, it is not surprising that HPC application suites overlooked the problem that application behavior is tightly-coupled with number of application processes. After all, this problem does not exist for single-threaded benchmarks, and it was not detected for HPC benchmarks that put high stress to a single resource, such as HPCG, HPL or HPCC suite.

The essence of benchmarking is to provide representative use cases for characterization and valid comparison of different systems. If the application suite does not provide it, then the results are misleading. Our results show that it is very important that HPC application suites specify narrow ranges for the number of processes that are representative of real-life application behavior, or at least provide some guidelines so users themselves could determine these ranges for their target platforms.

5.4 Related work

There are not many studies that analyse benchmarking methodologies and how to represent evaluation results of HPC systems and applications. Bailey [105] provides common guidelines for reporting benchmark results in technical computing, following his similar summary of misleading claims for reporting results in system evaluation [106]. He points out the possibilities of misleading conclusions and potential biases from using projections and extrapolations, tuning levels, evaluating non-representative segments of the workloads, etc. Nevertheless, he presents several rules and advocates the community to pay attention and avoid the biased results.

Hoefler et al. [107] attempt to define ground rules and guidelines for the interpretation of HPC benchmarking. The authors propose statistical analysis and reporting techniques in order to improve the quality of reporting research results in HPC and ensure interpretability and reproducibility. In their study, they identify several frequent problems and propose rules to avoid them. Their analysis covers methods for reporting the results of speed-up, usage of various means, summarizing ratios, confidence intervals, normalization, usage of various chart techniques, and others.

Sayeed et al. [96] advocate the use of real applications for benchmarking in HPC, and that small benchmarks cannot predict the behavior of the real HPC applications. They discuss important questions, challenges, tools and metrics in evaluating performance using HPC applications. Afterwards, they evaluate the performance of four application benchmarks on three different parallel architectures, and measure the runtime, inter-process communication overhead, I/O characteristics and memory footprint. This way, they show the importance of reporting various metrics, in order to have a better representation of application and system performance. Since they measure these metrics on several numbers of execution processes, the results differ from one execution to another. It is clear from their results that on different numbers of execution processes, different platforms perform better or worse, which can significantly bias the analysis on certain scale of the experiments.

Marjanović et al. [108] explore the impact of input data-set for three representative benchmarks: HPL, HPCG and High-performance Geometric Multigrid (HPGMG). They perform an analysis on six distinct HPC platforms at the node level, and perform scale-out analysis on one of the platforms. Their results show that exploring multiple problem sizes gives a more complete picture of the underlying system performance, than a single number representing the best performance, which is the usual way of reporting the results. They advocate for the community to discuss and propose a method for aggregating these values into a representative result of the system performance.

In our study, we focus on two important aspects of benchmarking with HPC applications: the importance of defining the representative scale of the experiments and measurement granularity in quantifying performance bottlenecks, which are often overlooked by the community. To our knowledge, this is the first study that analyses the importance of a deterministic range for the number of execution processes. We also suggest a simple way to show several values for portions of time spent in different utilizations of certain metric. It does not require additional executions or special evaluation infrastructure, yet it gives much better representation of application behavior and clearer focus on its bottlenecks.

5.5 Summary

A clear understanding of HPC system performance factors and bottlenecks is essential for designing an HPC infrastructure with the best features and a reasonable cost. Such a perception can only be achieved by closely analysing existing HPC systems and execution of their workloads.

When executing production HPC applications, our findings show that HPC application performance metrics strongly depend on the number of execution processes. We argue that it is essential that HPC application suites specify narrow ranges on the number of processes, for the results to be representative of a real-world application use. Also, we find that average measurements of performance metrics and bottlenecks can be highly misleading. Instead, we suggest that performance measurements should be defined as the percentage of execution time in which applications use certain portions of maximum sustained values.

Overall, we believe this study offers new guidelines for accurately measuring key performance factors and their impact on overall HPC performance.

The Fast drives out the Slow even if the Fast is wrong.
William Kahan, professor Emeritus of Mathematics,
and of E.E. & Computer Science

6 First steps on the performance impact of memory bandwidth and latency

First defined more than two decades ago, the memory wall remains a fundamental limitation to system performance. Innovations in 3D-stacking technology enable DRAM devices with much higher bandwidths than traditional DIMMs. The first such products hit the market, and some of the publicity claims that they will break through the memory wall. Here we summarize our preliminary analysis and expectations of how such 3D-stacked DRAMs will affect the memory wall for a set of representative HPC applications. We conclude that although 3D-stacked DRAM is a major technological innovation, it cannot eliminate the memory wall.

6.1 Introduction

In 1995, Wulf and McKee published a four-page note entitled "Hitting the Memory Wall: Implications of the Obvious" in the (unrefereed) ACM SIGARCH *Computing Architecture News* [11]. The motivation was simple: at the time, researchers were so focused on improving cache designs and developing other latency-tolerance techniques that the computer architecture community largely ignored main memory systems. The article projected the performance impact of the increasing speed gap between processors and memory, referred to as the *Memory Wall* (see Section 2.2.1). There were other articles pointing-out to the problems of main memory, however the memory wall note seemed to strike a nerve where the others did not, though, and it inspired a considerable backlash.

One set of arguments maintained that latency-tolerance techniques like out of order execution, wider instruction issue, and speculative techniques such as hardware prefetching would bridge the processor-memory performance gap. Even in combination, though, such approaches can mask the latency of only so many memory requests — the exact numbers depend on the sizes of the hardware structures. The cost and complexity of implementing larger and larger structures proved prohibitive, and although latency tolerance pushed the memory wall back, it did not save us.

Another set of arguments maintained that new memory technologies like Intelligent RAM (IRAM) [109] and Rambus Direct DRAM (RDRAM) [110] would eliminate the memory wall. In spite of years of hype, embedded DRAM (i.e., IRAM, or eDRAM) did not appear in commercial products for another five years, and then it was only cost-effective in special-purpose platforms like game consoles [111, 112]. eDRAM would not appear in general purpose processor chips for another decade [113, 114]. Rambus modified the DRAM interface and introduced narrow, high-speed channels that supported a peak bandwidth of 1.6 Gbytes/s — a significant improvement over other memory devices at the time — but taking full advantage of RDRAM capabilities required memory controller modifications that would introduce both design and verification costs. Intel released the Pentium 4 with all RDRAM (but without an optimized memory controller), but these systems came at higher cost and offered little or no performance gain. Subsequent products [115, 116] moved to DDR (double data rate) devices [117]. Rambus and others continue to deliver high-performance DRAM technology, but they have not (yet) removed the memory wall — latency remains limited by the speed of the underlying storage technology, and high bandwidth does not necessarily reduce latency.

Technological evolutions and revolutions notwithstanding, the memory wall has imposed a fundamental limitation to system performance for more than 20 years. 3D-stacking technology now enables DRAM devices that support much higher bandwidths than traditional DIMMs, and the first commercial products such are the Hybrid Memory Cube (HMC) [21] and High Bandwidth Memory (HBM) [22], appeared (their descriptions is given in Section 2.3). Some of the publicity surrounding these promising new devices suggests that they will break through the wall.

Here we summarize our analysis and expectations of how 3D-stacked DRAMs will affect the memory wall for a particular set of HPC applications. Recall that the memory wall was defined in terms of latency, not bandwidth. Higher bandwidth *may* lower average latency, provided that our applications offer sufficient memory-level parallelism (MLP) [118] and that CPU architectures can exploit it. But higher bandwidth by itself cannot guarantee better performance. How well we can exploit available bandwidth ultimately depends on the inherent MLP in our targeted workloads.

6.2 Latency vs. Bandwidth

Although memory latency and bandwidth are often described as independent concepts, they are inherently interrelated. We describe their relation in Section 2.2. Nonetheless, we examine the likely impact of 3D devices on them, in turn.

6.2.1 Memory access latency

When analyzing memory access latency, it is important to distinguish the *lead-off* and *loaded* memory access latencies (see Section 2.2). 3D-stacked DRAM does not change the time that memory requests spend in the CPU. DRAM technology determines time spent accessing the storage array, so that time is unlikely to decrease significantly. Placing 3D-stacked memory devices on a silicon interposer instead of a PCB reduces time spent in the memory channel. However, higher memory system complexity could *increase* memory access latency. For example, HMC memory systems include serial links, serializing/deserializing logic, and more complex memory controllers on both the CPU and memory sides. Multi-hop memory accesses require routing and multiple channel access. Overall, 3D-stacked DRAM barely reduces request time in an idle system. In fact, the first memory system implementations with 3D-stacked DRAM devices still locate them on the PCB [119, 120], so lead-off memory access latency will probably increase.

Figure 6.1 compares bandwidth-latency curves of a conventional and an emerging memory system to characterize the impact of 3D-stacked DRAMs on loaded memory access latency. For the conventional memory system, we analyze a four-channel DDR3 memory running at 1066 MHz frequency. The maximum theoretical system bandwidth is 68.2 GB/s (four channels$\times$frequency $\times$bytes-per-transfer). For the emerging memory system, we analyze an HMC device. Note that our conclusions also apply to other 3D-stacked DRAMs. Lack of access to real hardware and to details of on-CPU memory controllers and in-memory logic (controller and request routing) complicates estimating the HMC bandwidth-latency curve. For purposes of our analysis, we estimate that the constant-latency region of the curve will reach at least 25% of the maximum theoretical bandwidth, i.e., 80 GB/s.

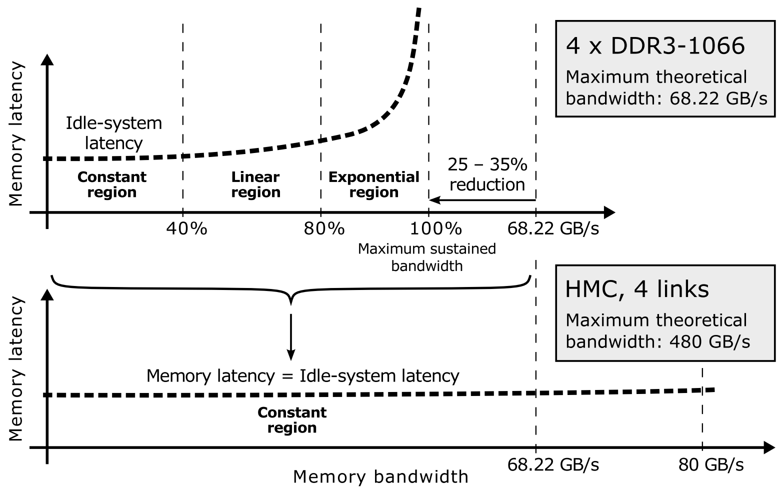


Figure 6.1: Bandwidth-latency curves of DDR3 and HMC systems.

Latency curve transition from conventional DDR3 to the HMC is illustrated in Figure 6.1. The constant-latency region of the HMC (up to 80 GB/s) exceeds the maximum theoretical bandwidth of the conventional system (68.2 GB/s). Thus, it covers all three regions of the DDR3 system — constant, linear, and exponential. If an application is in the constant-latency region of the DDR3 system (i.e., when memory access latency corresponds to lead-off latency), upgrading the memory to the HMC will not reduce memory access latency, nor will it improve overall performance. If the application is in the linear or exponential regions in the DDR3 system, a significant portion of its memory access latency comes from collisions between concurrent memory requests. In this case, the bandwidth upgrade may reduce contention, which could reduce memory access latency and improve performance.

6.2.2 Memory bandwidth

We also analyze whether high-bandwidth memories will increase effective memory bandwidth — what applications actually use. According to Little's Law [51], effective application bandwidth is directly proportional to the number of outstanding memory requests and inversely proportional to memory access latency:[1]

$$effective\ bandwidth \propto \frac{outstanding\ memory\ requests}{memory\ access\ latency} \tag{6.1}$$

[1] Instead of equality as originally used in Little's Law, we use *proportional to* ("$\propto$") to avoid converting between units that quantify memory bandwidth (GB/s) and memory access latency (CPU cycles or nanoseconds).

Properties of the application (such as the portion of memory accesses in the overall instruction mix and data and control dependencies) and the CPU (such as core count, out-of-order issue, speculative execution, branch prediction, and prefetching) determine the number of outstanding memory requests. 3D-stacked DRAM will not change these parameters, and thus we expect that the number of outstanding memory requests to remain roughly the same. Therefore, effective application bandwidth with emerging 3D-stacked DRAM systems will increase only if memory access latency is reduced.

6.2.3 Summary

3D-stacked DRAM devices will significantly increase available memory bandwidth. How well applications will exploit that higher bandwidth, though, ultimately depends on the workload's memory-level parallelism (MLP). For bandwidth-hungry applications that are in the linear or exponential regions of the bandwidth-latency curve, the bandwidth upgrade will reduce contention among concurrent memory requests, reduce memory access latency, and improve performance. Lower memory access latency will also increase effective application bandwidth. However, 3D-stacked DRAM cannot reduce lead-off memory access latency. Memory access latency, and thus effective bandwidth and overall performance, will not improve for applications for which lack of MLP limits effective bandwidth.

6.3 Experimental environment

We conduct a preliminary evaluation of our analysis for a set of HPC applications running on a production system. This section describes our hardware platform and applications along with the methodology we use in the study.

6.3.1 Hardware platform

We conduct all experiments on a dual-socket Sandy Bridge-EP E5-2620 server. Each socket contains six cores operating at 2.3 GHz. We execute experiments with and without the supported two-way hyperthreading. When hyperthreading is disabled at the operating system level the platform appears to have 12 virtual CPUs (two sockets $\times$ six cores). With hyperthreading enabled, the OS sees twice as many. We fully utilize the hardware platform in all the experiments, i.e., we execute either 12 or 24 application processes. Each Sandy Bridge processor accesses main memory through four channels, and each channel connects to am 8 GB DDR3 DIMM, which gives 64 GB total server memory. We use a single server (not a large-scale HPC cluster), because memory bandwidth measurements require root privileges (see Section 6.3.3).

We initially set the memory frequency to 1066 MHz, which makes the theoretical maximum memory bandwidth 68.2 GB/s. In order to analyze the performance impact (improvement) of higher memory bandwidth, we then increase the frequency to 1333 MHz (through the BIOS setup at boot time), which increases bandwidth by 25%. This change has no impact on memory latency, though: memory operation latencies are still limited by the DRAM technology. When memory frequency increases (i.e., the duration of a memory cycle decreases), memory commands take more cycles, so memory operation latencies remain practically the same.

6.3.2 HPC applications

We analyze a set of four large-scale production HPC applications from UEABS (see Section 3.2.4): ALYA, GROMACS, NAMD, and Quantum Espresso (QE). We could not run the remaining applications because the input dataset sizes exceed the main memory capacity of our hardware platform. Apart from HPC applications, we used two widely-used HPC benchmarks, HPL and STREAM (see Sections 3.2.1 and 3.2.2).

6.3.3 Methodology

Memory bandwidth: We calculate the *maximum theoretical bandwidth* of the system based on the specification of the hardware platform under study — by multiplying memory frequency (1066 MHz or 1333 MHz) by the memory data bus width and number of memory channels. We measure *sustainable memory bandwidth* with the STREAM benchmark (a common approach [91]). We measure effective bandwidth via the Intel Performance Counter Monitor (PCM) library, which provides routines to access the memory controller performance counters.[2] In all experiments, we report total memory bandwidth, i.e., read and write memory traffic of all four channels.

Performance: All applications under study report their performance in their output files. For Quantum Espresso, ALYA, GROMACS, NAMD, and HPL, performance corresponds to the number of elements that are processed in a time unit, which is directly proportional to the number of floating point operations per second. Performance of the STREAM benchmark corresponds to the sustainable memory bandwidth.

[2] Access to memory controller (uncore) performance counters requires root privileges.

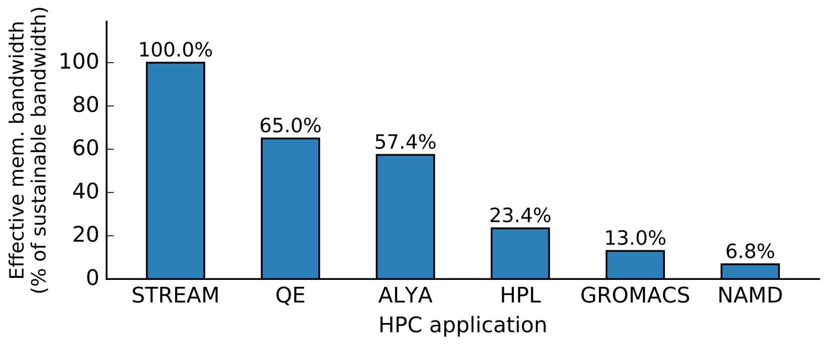


Figure 6.2: DDR3-1066: Portion of the maximum sustainable bandwidth (STREAM) that HPC applications actually use.

6.4 Results

First, we analyze the system without hyperthreading running with a memory frequency of 1066 MHz. The maximum theoretical memory bandwidth of this configuration is 68.2 GB/s, and the maximum sustainable memory bandwidth is 54.1 GB/s. Figure 6.2 shows the application memory bandwidth relative to the maximum sustainable bandwidth. STREAM, Quantum espresso (QE), and ALYA use a significant portion of the maximum sustainable bandwidth, and thus we expect that increasing available memory bandwidth will improve performance for these applications by increasing their effective memory bandwidth. On the other hand, HPL, GROMACS, and NAMD use a small portion of the sustainable memory bandwidth — 23.4%, 13%, and 6.8%, respectively. We expect no significant performance improvements for these applications when available memory bandwidth increases.

In Figure 6.3, we quantify the impact of a 25% memory bandwidth increase on application performance. Performance improves by 14.7%, 8.5%, and 3.7% for STREAM, QE, and ALYA, respectively. This improvement clearly correlates with the memory bandwidth that the applications use in the baseline system configuration with DDR3-1066 memory. For HPL, GROMACS, and NAMD, performance improves negligibly if at all — from 0% (HPL) to 1.6% (NAMD). Effective memory bandwidth follows the same trend.

We repeat the experiments with hyperthreading enabled in order to understand the impact on effective memory bandwidth.[3] For all applications but HPL, the results change insignificantly from those in Figures 6.2 and 6.3. HPL changes memory behavior with hyperthreading: it becomes bandwidth-

[3] The OS views the platform as 24 virtual CPUs: 2 sockets$\times$6 cores$\times$2, so the workloads comprise 24 application processes.

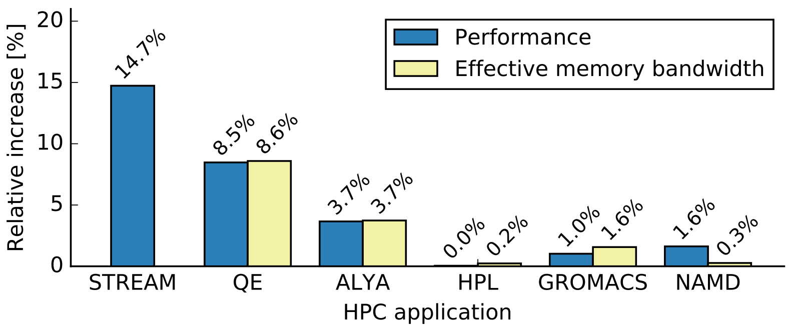


Figure 6.3: Performance improvement and effective memory bandwidth increase due to 25% memory bandwidth increment.

hungry — in the configuration with 1066 MHz memory, it uses 54% of the maximum sustainable bandwidth. When memory frequency increases to 1333 MHz, against our expectations, performance does not improve, nor does effective memory bandwidth increase.

To understand why HPL does not benefit from the 25% bandwidth increase, we use a *roofline model* [91] to correlate the application performance with its *operational intensity*, or the number of the floating point operations (FLOPS) that it executes per byte of data transferred between the CPU and main memory. Figure 6.4 shows roofline models for our two systems, with 1066 MHz and 1333 MHz main memory, and the position of the HPL benchmark in each of them. The x axis of the figure shows the application operational intensity, and the y axis shows application performance in GFLOPS/s. The sloped line shows the maximum sustainable memory bandwidth (as measured by the STREAM benchmark); this determines the upper performance bound for applications with low operational intensity. The horizontal line shows maximum sustainable performance in GFLOPS/s (determined by the DGEMM routine [121]).

Our roofline model shows that even though HPL uses a significant portion of the available bandwidth, it touches the horizontal line of the chart, meaning that it is clearly limited by the GFLOPS/s that the CPUs can sustain. Increasing the memory frequency from 1066 MHz to 1333 MHz increases sustainable memory bandwidth and raises the inclined part of the roofline chart. However, it has no impact on the horizontal GFLOPS/s performance limit, and therefore does not improve HPL performance.

To summarize, even when increasing available memory bandwidth mitigates collisions in the memory system, other parts of the system (processing units or interconnect) can still limit system performance. This result empha-

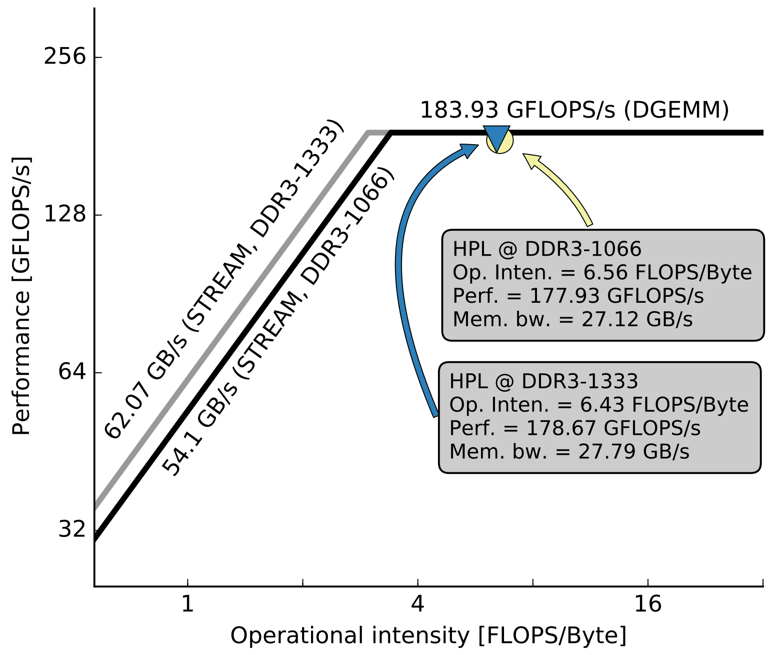


Figure 6.4: Position of HPL application on platform roofline model, with DDR3-1066 and DDR3-1333 memory configurations.

sizes the importance of building balanced computer systems that properly exploit the benefits of novel high-bandwidth memory solutions.

6.5 Looking forward

How well applications will exploit the higher bandwidth provided by emerging 3D-stacked DRAMs ultimately depends on the workload's memory-level parallelism. For high-MLP applications, the bandwidth upgrade will reduce contention among concurrent memory requests, reduce memory latency, and improve performance. However, 3D-stacked DRAMs cannot reduce lead-off memory access latency. Thus, they will not improve the performance of applications with limited MLP.

Further, we are unlikely simply to replace conventional DIMMs with the 3D-stacked DRAMs. Higher prices will prevent memories composed only of 3D devices and may even limit adoption. Instead, we are likely to see main memories that include both 3D devices and conventional DIMMs. Thus, in the best case, the system will still be bandwidth-limited, and it will often be latency-limited. So in contrast to the publicity surrounding 3D DRAMs, they are unlikely to break through the memory wall — at best, they move it.

Building balanced, high-performance systems will require us to design CPUs and memory controllers that can exploit the new devices for high-MLP application domains. The logic layers in the HMC and HBM offer interesting possibilities for in-memory processing and sophisticated memory controller functionality. 3D-stacked DRAM is certainly an interesting technological innovation. Finding a way to use this innovation to build high-performance systems, however, will take time — and the extent of its adoption will likely come down to cost.

This simulation is not as the former.

Malvolio, Act II, scene V
of Shakespeare's Twelfth Night

7 Memory system evaluation: Modeling system performance and energy without simulating the CPU

The approaching end of DRAM scaling and expansion of emerging memory technologies is motivating a lot of research in future memory systems. Novel memory systems are typically explored by hardware simulators that are slow and often have a simplified or obsolete model of the CPU.

This study presents an analytical model that quantifies the impact of the main memory on application performance and system power and energy consumption. The model is based on memory system profiling and instrumentation of an application execution on a baseline memory system. The model outputs are the predicted performance, power and energy consumption on the target memory. The model is evaluated on two actual platforms: Sandy Bridge-EP E5-2670 and Knights Landing Xeon Phi 7230 platforms with various memory configurations. The evaluation results show that the model predictions are accurate, typically with only 2% difference from the values measured on actual hardware. We plan to release the model source code and all input data required for memory system and application profiling. The released package can be seamlessly installed and used on high-end Intel platforms.

7.1 Introduction

The memory system is a major contributor to the deployment and operational costs of a large-scale high-performance computing (HPC) cluster [3, 7, 8], and in terms of system performance it is one of the most critical aspects of the system's design [10, 11]. For decades, most server and HPC cluster memory systems have been based on DRAM DIMMs. However, it is becoming questionable whether DRAM DIMMs will continue to scale and meet the industry's demand for high performance and high capacity memory. Significant effort is therefore being invested into the research and development of future memory systems.

Novel memory systems are typically explored using hardware simulators. System simulation is, however, time consuming, which limits the number of design options that can be explored within a practical length of time. Also, although memory simulators are typically well validated [84, 122], current CPU simulators have various shortcomings, such as simplified out-of-order execution, an obsolete data prefetcher and a lack of virtual-to-physical memory translation, all of which can make a huge difference between the simulated and actual memory system, in terms of behavior and performance.

This study proposes an analytical model that quantifies the impact of the memory on the application performance and system power and energy consumption, providing point estimates and error bars. The model is based on memory system profiling and instrumentation of an execution of the application. It has been evaluated on two actual platforms: Sandy Bridge-EP E5-2670 with four DRAM configurations DDR3-800/1066/1333/1600, and Knights Landing (KNL) Xeon Phi 7230 with DDR4 and 3D-stacked MCDRAM. The evaluation results show that the model predictions are very accurate — the average difference from the performance, power and energy measured on the actual hardware is only 2%, 1.1% and 1.7%, respectively.

We also compare the model's performance predictions with simulation results for the Sandy Bridge-EP E5-2670 system with ZSim [79, 80] and DRAMSim2 [84], and our model shows significantly better accuracy over the simulator. The model is also faster than the hardware simulators by *three orders of magnitude*, so it can be used to analyze production HPC applications, on arbitrarily sized systems. Additionally, the method is based on profiling of the application's memory behavior, so it does not require detailed modeling of the CPU as it already takes account of the real (and not publicly disclosed) data prefetcher and out-of-order engine. Therefore, it can be used to model various platforms as long as they support the required application profiling. The model was initially developed for the Sandy Bridge platform, and later we evaluated it for the KNL server. Adjustment of the model to the KNL system was trivial, as it required changes to only a few hardware parameters, such as, for example the reorder buffer size.

We will release the model source code as open source. The release will include all model inputs and outputs and evaluation results for the case study that is used in the rest of this study. The package includes the memory system profiles, CPU parameters, application profiles and memory power parameters, as well as the power, performance and energy outputs from the model and the measurements on the baseline and target platforms. The released model is ready to be used on high-end Intel platforms, and we would encourage the community to use it, adapt it to other platforms, and share their own evaluations.

7.2 On memory bandwidth and latency

The connection between memory access latency and used bandwidth is given by the bandwidth–latency curve, as illustrated in Figure 2.3 and described in Section 2.2. We repeat the essential information here. The curve in Figure 2.3 has three regions that are limited by the maximum sustainable bandwidth, which is 65–75% of the maximum theoretical bandwidth [10]. In the first region, the application's used bandwidth is low, so there are few concurrent memory requests and contention for shared resources is negligible. Over this region the memory latency is constant and equal to the lead-off latency. In the second region, the application's used bandwidth is between 40% and 80% of sustainable bandwidth, and there are increasing numbers of concurrent memory requests. Contention for shared resources is moderate, and latency grows linearly with the used bandwidth. In the final region, in which the application's used bandwidth is high, contention among concurrent memory requests is severe and memory latency increases exponentially.

It is critical to distinguish between the lead-off and loaded latencies, since the difference between them can be on the order of hundreds of nanoseconds. A fully-stressed memory system therefore introduces a significant loaded latency, which leads to a major performance impact.

7.3 Model overview

This section summarizes the main idea behind the presented analytical models and it describes the model inputs and outputs.

7.3.1 The idea: Moving between memory curves

The main idea of this study is that we can understand the effect of changing the memory system by understanding how the application moves from one bandwidth–latency curve to another. We illustrate this idea using the DDR4 and MCDRAM memories on Intel's Knight's Landing platform. This platform has two memory systems, so there are two bandwidth–latency curves, shown

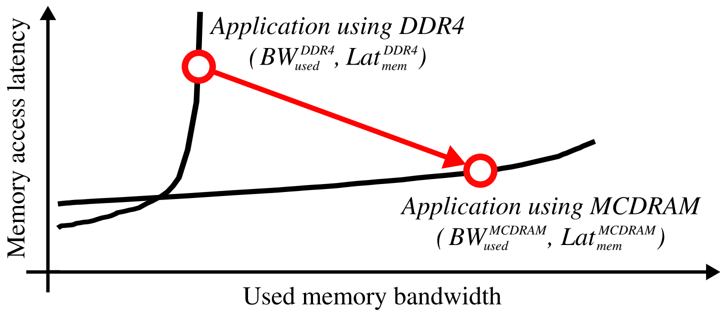

Memory access latency
Application using DDR4
(BW_{used}^{DDR4}, Lat_{mem}^{DDR4})
Application using MCDRAM
(BW_{used}^{MCDRAM}, Lat_{mem}^{MCDRAM})
Used memory bandwidth

Figure 7.1: High-level view of the transition from DDR4 to high-bandwidth MCDRAM memory on the KNL platform.

together on the same plot in Figure 7.1.[1] When used bandwidth is high (towards the right of the figure), MCDRAM is clearly better, but when used bandwidth is low (towards the left), DDR4 has lower latency due to its lower lead-off latency.

When an application (or application phase) executes on the DDR4 main memory, it will be positioned at some point on the DDR4 curve; e.g. point $(BW_{used}^{DDR4},\ lat_{\mathrm{mem}}^{DDR4})$ illustrated in Figure 7.1. Analogously, when the same application is executed on the MCDRAM memory, it will be positioned at some point on the MCDRAM curve, e.g. $(BW_{used}^{MCDRAM},\ lat_{\mathrm{mem}}^{MCDRAM})$. We see that the application in Figure 7.1 benefits from running on the MCDRAM through a lower memory latency (MCDRAM point is lower) and a higher used bandwidth (MCDRAM point is to the right).

This idea, of moving between bandwidth–latency curves, is central to the performance, power and energy model presented in this chapter.

7.3.2 Model inputs

Figure 7.2 gives a high-level overview of the whole process of performance, power and energy estimation. The model inputs, shown towards the left of the figure, are the detailed bandwidth–latency curves, measured for the *baseline memory system* and the *target memory system*, parameters for the CPU (which is the same for both memory systems), as well as the application profiles on the baseline memory system. These inputs can all be easily obtained on mainstream platforms and it is becoming increasingly possible to obtain them on emerging platforms. The outputs from the model will be the predicted performance, power and energy consumption on the target memory system.

[1] Figure 7.1 shows a simplified bandwidth–latency curve, as discussed in Section 7.2. Detailed curves are given in Figure 7.3 in Section 7.4.

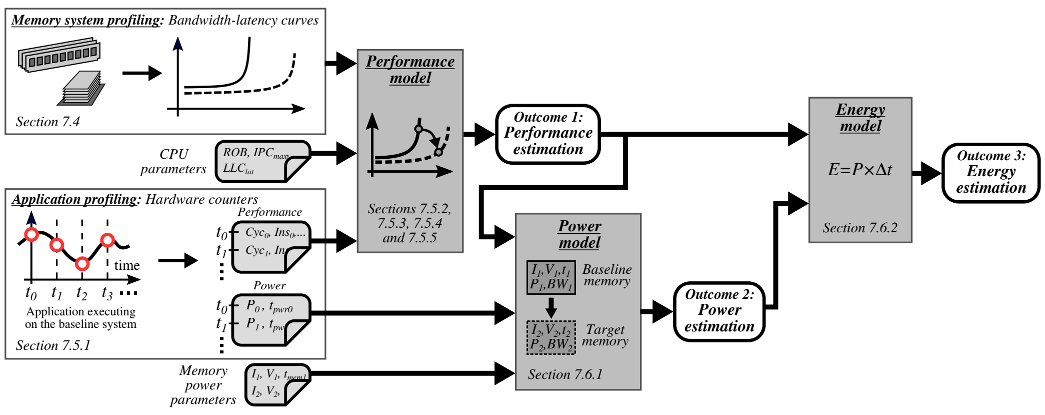

Memory system profiling: Bandwidth-latency curves
Section 7.4
CPU parameters
ROB, IPC_{max}, LLC_{lat}
Application profiling: Hardware counters
Performance
t_0 Cyc_0, Ins_0,...
t_1 Cyc_1, In
time
t_0 t_1 t_2 t_3 ...
Application executing on the baseline system
Section 7.5.1
Power
t_0 P_0, t_{pwr0}
t_1 P_1, t_{pw}
Memory power parameters
I_1, V_1, t_{mem1}
I_2, V_2,
Performance model
Sections 7.5.2, 7.5.3, 7.5.4 and 7.5.5
Outcome 1: Performance estimation
Power model
I_1,V_1,t_1 P_1,BW_1 Baseline memory
I_2,V_2,t_2 P_2,BW_2 Target memory
Section 7.6.1
Outcome 2: Power estimation
Energy model
$E=P\times\Delta t$
Section 7.6.2
Outcome 3: Energy estimation

Figure 7.2: Diagram of the whole process of performance, power and energy estimation. The cross-references indicate which section describes which part of the estimation process.

Memory system profiling is done via bandwidth–latency curves, for the baseline and target memory systems, along the lines outlined in Section 7.2. The precise method for obtaining these curves is given in Section 7.4, which describes the memory profiling microbenchmarks and their outputs.

CPU parameters are needed, alongside the application profiling (see below), to characterize the relationship between memory system latency and execution time. This relationship is dependent on the processor's ability to hide memory latency by overlapping memory accesses with independent instructions. As detailed in Section 7.5.3, the performance model therefore requires some basic parameters of the processor under study: re-order buffer (ROB) capacity, miss information status holding register (MSHR) capacity and minimum theoretical cycles-per-instruction (CPI).

Application profiling is done on the baseline memory system, and consists of executing the application and profiling it using hardware performance counters. *Application performance profiling* obtains the number of CPU cycles, number of instructions, number of last-level cache (LLC) misses and the read and write memory bandwidths. *Application power profiling* measures the total power consumption using integrated or external power measurement infrastructure, and memory-related power parameters using performance counters. These parameters include row-buffer access statistics, number of page activations and page misses, and number of cycles in power-down states: active standby, precharge power-down and self-refresh.

Since the application's behavior changes over time, application profiling is done by sampling over regular time intervals, which we refer to as segments. The overall outcome of application profiling is two time-stamped trace files, needed for performance and power consumption, respectively. Further details on application profiling are given in Section 7.5.1.

Memory power parameters characterize the baseline and target memory systems, in terms of the power consumption in various operational modes, idle state and power-down states, as well as the energy consumption for various operations such as read and write transfers, row buffer hits and misses. These figures are typically provided by the memory device manufacturers [123].

7.3.3 Performance, power and energy estimation

The right-hand side of Figure 7.2 gives an overview of the whole process of performance, power and energy estimation. Since application profiling involves collecting a trace over the program's execution, the performance and power models are ran for each segment (time interval) in the trace. This gives the predicted execution time, power and energy consumption of each segment. Summing over time gives the final execution time and energy for the whole application. The application's average power demand is total energy divided by total execution time.

The **Performance model** reads the application performance information from the profiling trace-file and determines the application's position on the bandwidth–latency curve for the baseline memory system. As described in detail in Sections 7.5.2, 7.5.3 and 7.5.5, the model then estimates the application's position on the memory bandwidth–latency curve for the target memory system, and uses it to predict the application's performance on the target memory system.

The **Power model** estimates the power consumption of the target memory system using the application performance profiling and the memory power parameters. Finally, the **Energy model** is done based on the output of the performance and power models. The detailed description and evaluation of the power and energy models are presented in Sections 7.6.1 to 7.8.2.

7.3.4 Model source codes and case study

We plan to release the model source code as open source. The release includes all model inputs and outputs and all evaluation results for the case study that is used in the rest of this analysis. The package includes the memory system profiles, CPU parameters, application profiles and memory power parameters, as well as the power, performance and energy outputs from the model and the measurements on the baseline and target platforms.

7.4 Memory system profiling

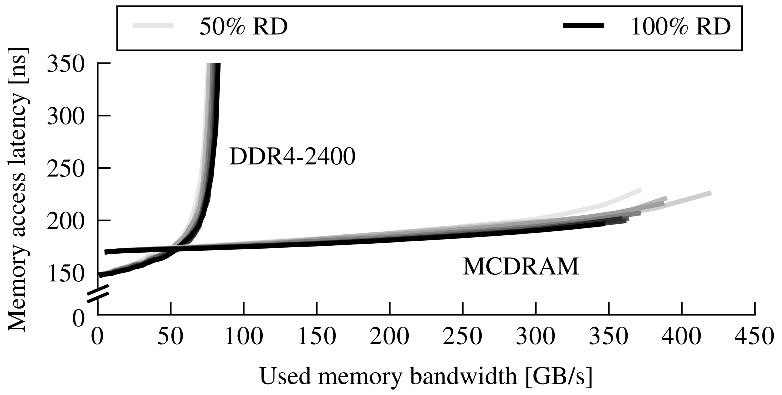
The baseline and target memory systems are characterized using bandwidth–latency curves measured on a real platform. This is straightforward for mature technologies, but for emerging memory devices that are not yet available in off-the-shelf servers, the bandwidth–latency curve can be measured on a developer board with a prototype of the new device [119], or alternatively it can be provided by the manufacturer.

The bandwidth–latency curve is determined using a pointer-chasing microbenchmark designed to measure latency [124]. It executes constant number of data-dependent load instructions that traverse randomized memory locations, so we exclude the effect of prefetcher and ensure that all the loads go to the main memory. By measuring clock cycles necessary for the benchmark execution, we can derive the latency of a single instruction, i.e., single load memory access. Pointer-chasing microbenchmark is running concurrently with a derivative of the STREAM benchmark [62] that was modified to vary the load on the memory system, for various ratios of read and write accesses.

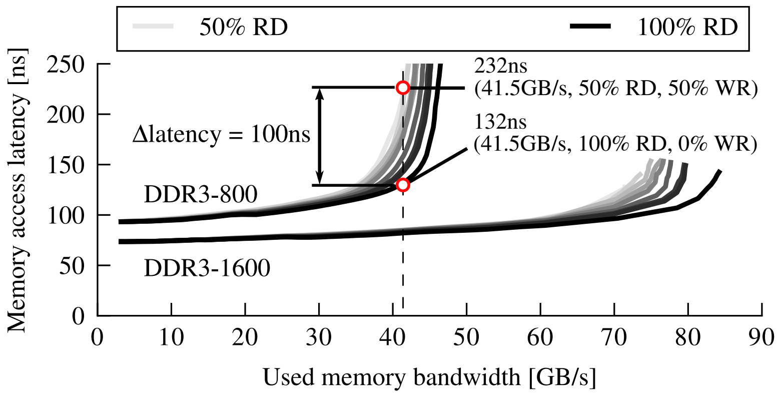
Although Section 2.2 plots a single bandwidth–latency curve, in reality a single memory system has a family of curves. This is because the memory bandwidth on the x axis combines into a single metric the aggregate bandwidth of reads and writes, even though they are fundamentally different operations.[2] The main reason for distinguishing reads and writes is that write requests introduce additional delays not required by memory reads [50]: *Write Recovery time* or t_{WR} is a minimum delay between the end of a write and the next precharge command, and *Write To Read delay time* or t_{WTR} is a minimum time interval between a memory write and a consecutive read. So, increasing the proportion of write requests reduces the sustainable bandwidth and increases the loaded latency.

As an example, Figure 7.3 shows the measured bandwidth–latency curves for the Knights Landing and Sandy Bridge platforms, as the proportion of reads is varied between 50% and 100%. The lightest curves correspond to 50% reads and the darkest curves correspond to 100% reads. Instead of the single bandwidth–latency curve per memory system that was illustrated in Figure 2.3, we now see a family of curves. When the used memory bandwidth is low or moderate, the read fraction has negligible impact on the memory access latency, i.e. within the constant and linear regions, the bandwidth–latency curves practically overlap. As the stress to the memory system increases, however, the read fraction starts to have a significant impact on latency.

[2] In addition to the fractions of reads and writes, the bandwidth–latency curve measures the loaded memory latency, which depends on other parameters, such as the row-buffer hit rate. This analysis is a part of ongoing work.



(a) Knights Landing platform with a DDR4-2400 and MCDRAM.



(b) Sandy Bridge platform with DDR3-800/1066/1333/1600. DDR3-1066 and DDR3-1333 are excluded to improve the visibility.

Figure 7.3: Bandwidth–latency curves for the platforms under study. Memory access latency with respect to used memory bandwidth *cannot* be approximated with a single curve — as the used memory bandwidth increases, memory traffic read/write composition makes a significant latency impact.

For example, in Figure 7.3b, at an aggregate bandwidth of 41.5 GB/s, the (read) latency with 100% reads is 132 ns, but the read latency with 50% reads and 50% writes is 232 ns, an increase of 76% and 100 ns. In general, for all experiments we did, shown in Figure 7.3, curves with a higher percentage of writes (lighter curves) are located higher (at higher latency) on the chart.

Recently Clapp et. al. [125] also did a preliminary analysis of bandwidth–latency curves for different memory frequencies (DDR3-1333 and DDR3-1600) and different read-to-write ratios (3:1 and 2:1). Based on an analysis of four curves, the authors conclude that it is sufficient to use a single, generic memory bandwidth–latency curve for different frequencies and memory traffic compositions. Our analysis is based on numerous measurements on a wide range of DDR3, DDR4 and MCDRAM frequencies, with fine-grain changes in the read-to-write ratio. Our findings show that different memory frequencies have fundamentally different bandwidth–latency curves with different shapes and different lead-off and maximum memory access latencies. We also show that the read-to-write ratio may have a significant impact on memory access latency. Directly contrary to the conclusion of Clapp et. al. [125], our study shows that the relationship between bandwidth and latency cannot be approximated with a single curve. To the best of our knowledge this is the first study of memory system read latency that considers how the latency depends on the fractions of reads and writes.

In conclusion, the memory system profiling must quantify the impact of the read fraction on the read latency. Across the benchmarks used in the study (see Section 7.7.2), the read fraction varied between 50% and 100%. We therefore profile the memory system by measuring several bandwidth–latency curves, as the read fraction is varied over this range.

7.5 Performance model: Detailed description

This section presents the analytical model that predicts the application's performance. This is done starting from the baseline and target memory system characterization via the bandwidth–latency curves, obtained as described in Section 7.4. The first step is to determine, for a given application, the relationship between memory system read latency and performance. We start, in Section 7.5.1, by outlining the application characteristics that must be measured on the baseline system. Then, in Section 7.5.2, we introduce the problem with a simple case, that of an in-order processor. Next, in Section 7.5.3, we analyse a complex out-of-order processor. Section 7.5.4 completes the analysis of out-of-order processor performance as a function of latency. Finally, in Section 7.5.5, we explain how the model combines this latency–performance characterization with the bandwidth–latency curves to obtain the estimate, with error bars, of the application performance on the target memory system.

7.5.1 Application profiling

As outlined in Section 7.3.2, the application's execution is divided into segments at regular time intervals. For each segment, we measure, using performance measuring counters, the number of cycles, number of instructions, read last-level cache (LLC) misses, used memory bandwidth, and the overall fraction of reads. These parameters and the notation used in the study are listed in Table 7.1.

Table 7.1: Performance model input parameters

Input parameter	Symbol
Number of cycles	Cyc_{tot}
Number of Instructions	Ins_{tot}
Application read LLC misses	$Miss_{\mathrm{LLC}}$
Used memory bandwidth for total traffic	$BW^{(1)}_{\mathrm{used}}$
Fraction of reads in total traffic	$Ratio_{\mathrm{R/W}}$

The used memory bandwidth, BW_{used}, and fraction of reads, $Ratio_{\mathrm{R/W}}$, are measured at the memory system, i.e. after the LLC. Both these figures include all accesses, whether issued by the application or the prefetcher, since both types of accesses appear the same to the memory system. In contrast, $Miss_{\mathrm{LLC}}$ only includes LLC read misses issued by the application. This parameter is used to estimate how the memory read latency impacts application performance, and only application read misses have a direct impact on performance.

In order to determine the duration of the sampling interval, we analyzed the tradeoff among the measurement overhead, trace-file size and model accuracy. An interval of 1 s was selected because it provided high accuracy (see Section 7.8) while introducing negligible measurement overhead of below 1%. With this sampling interval, the trace-file size of the benchmarks used in the study is in the range of hundreds of megabytes, which is acceptable.

7.5.2 In-order processors

This section derives the relationship between latency and performance for a simple in-order CPU. In the interest of helping the reader to follow the formulas, we start by summarizing the model's inputs and outputs in Table 7.2.

Our analysis distinguishes between *Memory access latency*, *Memory access penalty* and *LLC miss penalty.* **Memory access latency**, Lat_{mem}, is the number of CPU cycles necessary for a single load instruction that reads data from the main memory. It is measured as part of the memory system profiling and given in the memory bandwidth–latency curve. **Memory access penalty**, Pen_{mem}, is the difference between the latency of a main memory access and the latency of an LLC hit. The values of Lat_{mem} and Pen_{mem} are inputs to the model. The values for the baseline memory system are found by looking up the application's used bandwidth, measured on the baseline memory system. The values for the target memory system are generated

Table 7.2: Notation used in formulas: In-order processors

Description	Symbol
Inputs (in addition to Table 7.1)	
Memory access latency from bandwidth–latency curve	Lat_{mem}
Memory access penalty (Lat_{mem} minus LLC hit latency)	Pen_{mem}
Intermediate outputs	
Single LLC miss penalty (number of CPU stall cycles)	$Stalls_{\text{LLC}}$
Application cycles-per-instruction	CPI_{tot}
CPI component in the case of perfect LLC	CPI_0
CPI component due to LLC misses penalties	CPI_{LLC}
Outputs	
Application instructions-per-cycle ($1/CPI_{\text{tot}}$)	IPC_{tot}

as explained in Section 7.5.5. Finally, **LLC miss penalty**, $Stalls_{\text{LLC}}$, is calculated by the model as the average number of cycles for which the CPU pipeline is stalled because of each LLC miss.

We start by partitioning the application cycles-per-instruction, CPI_{tot}, into two components [52, 126, 127, 125]: $CPI_{\text{tot}} = CPI_0 + CPI_{\text{LLC}}$. The first component, CPI_0, is the application's CPI for the hypothetical case of a 100% LLC hit rate. This component is not affected by the memory access latency. The second component, CPI_{LLC}, is due to execution stalls due to the LLC misses. Once we know the number of stall cycles to be attributed to each LLC miss, we can calculate its value as [52, 128]:

$$CPI_{\text{LLC}} = \frac{Miss_{\text{LLC}} \times Stalls_{\text{LLC}}}{Ins_{\text{tot}}} \tag{7.1}$$

We use the superscripts (1) and (2) to distinguish between the baseline an target memory systems, respectively:

$$\begin{aligned} \text{Baseline memory:}\quad & CPI_{\text{tot}}^{(1)} = CPI_0^{(1)} + CPI_{\text{LLC}}^{(1)} \\ \text{Target memory:}\quad & CPI_{\text{tot}}^{(2)} = CPI_0^{(2)} + CPI_{\text{LLC}}^{(2)} \end{aligned} \tag{7.2}$$

Since the memory access latency does not affect CPI_0, we have $CPI_0^{(1)} = CPI_0^{(2)}$. Therefore, $CPI_{\text{tot}}^{(2)}$ from Eq. 7.2 can be expressed as:

$$CPI_{\text{tot}}^{(2)} = CPI_{\text{tot}}^{(1)} + \left(CPI_{\text{LLC}}^{(2)} - CPI_{\text{LLC}}^{(1)}\right) \tag{7.3}$$

Next, we assume that a change in the memory system, which for this section is a change in the read access latency, does not change the number of instructions, Ins_{tot}, or the application's memory access pattern. This is a reasonable assumption for applications or computational kernels that do not use busy-waiting or dynamic scheduling. A change in the memory access latency may affect the timeliness of the prefetcher, but it should not consistently affect its coverage or accuracy; in any case, we found this effect to be small.[3] We therefore also assume that the LLC miss rate is unaffected by the change in memory latency. In summary we conclude that we do not need superscripts (1) or (2) on Ins_{tot} and $Miss_{\text{LLC}}$. We can therefore substitute Eq. 7.1 into Eq. 7.3 to obtain:

$$CPI^{(2)}_{\text{tot}} = CPI^{(1)}_{\text{tot}} + \frac{Miss_{\text{LLC}}}{Ins_{\text{tot}}} \times \left(Stalls^{(2)}_{\text{LLC}} - Stalls^{(1)}_{\text{LLC}}\right) \tag{7.4}$$

We have not yet assumed an in-order processor, so all the above equations are also true for out-of-order processors. For an in-order processor we make the single observation that LLC misses directly lead to the pipeline stalls, i.e.: $Stalls_{\text{LLC}} = Pen_{\text{mem}}$. Substituting this into Eq. 7.4 gives:

$$CPI^{(2)}_{\text{tot}} = CPI^{(1)}_{\text{tot}} + \frac{Miss_{\text{LLC}}}{Ins_{\text{tot}}} \times \left(Pen^{(2)}_{\text{mem}} - Pen^{(1)}_{\text{mem}}\right) \quad \text{for an in-order processor} \tag{7.5}$$

Eq. 7.5 is important because it shows that the difference in application CPI for different memory systems, $CPI^{(2)}_{\text{tot}}$ and $CPI^{(1)}_{\text{tot}}$, can be calculated based on the corresponding memory access penalties, $Pen^{(2)}_{\text{mem}}$ and $Pen^{(1)}_{\text{mem}}$. Finally, by replacing $IPC = 1/CPI$, the application performance on the target memory system is calculated as:

$$IPC^{(2)}_{\text{tot}} = \frac{1}{\frac{1}{IPC^{(1)}_{\text{tot}}} + \frac{Miss_{\text{LLC}}}{Ins_{\text{tot}}} \times \left(Pen^{(2)}_{\text{mem}} - Pen^{(1)}_{\text{mem}}\right)} \quad \text{for an in-order processor} \tag{7.6}$$

7.5.3 Out-of-order processors

The analysis for out-of-order (OOO) processors is more complex because following an LLC miss the processor can continue executing independent instructions without immediately being stalled. In consequence, the number

[3] We measured the number of prefetches per instruction on our Sandy Bridge evaluation platform (Section 7.7.1). The overall difference between DDR3-800 and DDR3-1600 memory configurations across all benchmarks is less than 5%.

of stalls per LLC miss is no longer equal to the full memory access penalty, and it is typically strictly lower than it: $Stalls_{\text{LLC}} < Pen_{\text{mem}}$. In order to handle this inequality it is necessary to introduce the additional symbols given in Table 7.3.

Table 7.3: Notation used in formulas: Out-of-order processors

Description	Symbol
Inputs	
Instructions in reorder buffer	Ins_{ROB}
Size of miss information status holding register (MSHR)	$MSHR$
Minimum CPI, equal to reciprocal of maximum IPC	CPI_{min}
Intermediate outputs	
Number of execution cycles overlapped with LLC miss stalls, due to OOO mechanism	Cyc_{ooo}
Number of instructions executed during LLC miss stalls, due to OOO mechanism	Ins_{ooo}
Cyc_{tot} component in the case of perfect LLC	Cyc_0
Memory level parallelism: Number of concurrent LLC misses (memory accesses)	MLP

Isolated LLC miss

We first consider an isolated LLC miss. When an LLC miss occurs (isolated or not), the corresponding instruction must wait for data from memory, but the CPU pipeline continues issuing and executing independent instructions. Execution may halt, however, before the LLC miss is resolved, for two reasons. First, instruction issue may stop because the instruction window has filled with instructions, all of which are dependent, directly or indirectly, on the instruction waiting for data from main memory. Second, instruction commit may stop because the reorder buffer (ROB) has filled with instructions that cannot be committed until after the waiting instruction has itself been committed.[4] Both cases are illustrated in Figure 7.4. The upper part of the figure shows a timeline indicating whether instruction execution has halted, while the lower part of the figure shows snapshots of the ROB occupancy before an isolated LLC miss, while the processor is waiting for data, and after the data has been received. Immediately following the LLC miss, the ROB is occupied with a certain number of instructions. The ROB begins to fill, as the processor executes independent instructions. At some point, either there

[4] They cannot be committed because OOO processors commit instructions in-order.

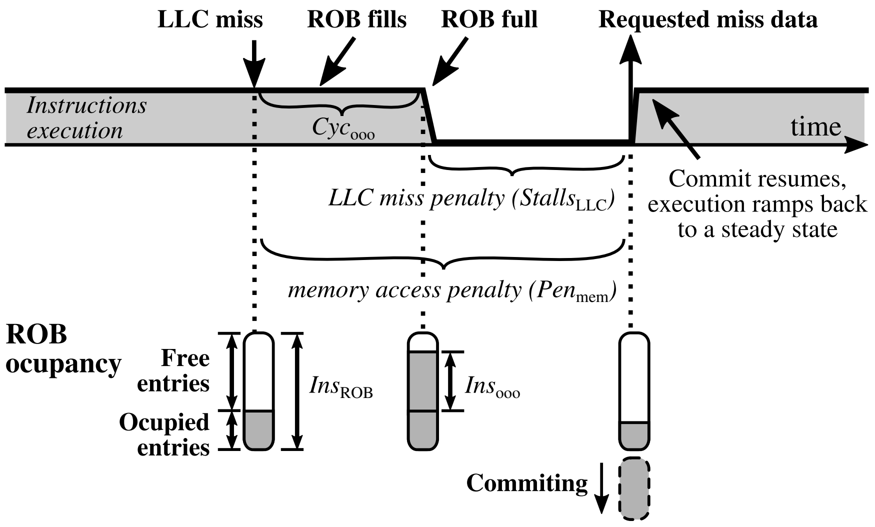


Figure 7.4: In OOO processors, LLC misses overlap with the execution of the instructions independent of the missing data. The overlap depends on the number of independent instructions in the instruction window and number of free entries in ROB [127].

are no more independent instructions or the ROB becomes full. In either case instruction execution will stall. Once the LLC miss has been resolved and the LLC miss data are available, the instructions waiting for the data can be executed and committed. This allows the instructions in the ROB to be committed, so issuing and execution of new instructions can resume.

In Figure 7.4, the period after the LLC miss in which the processor is executing new independent instructions is labeled as Cyc_{ooo}.So, the number of stall cycles is equal to the memory access penalty minus Cyc_{ooo} [52, 127]:

$$Stalls_{\text{LLC}} = Pen_{\text{mem}} - Cyc_{\text{ooo}} \quad \text{for an isolated miss} \tag{7.7}$$

If execution is immediately halted following the LLC miss, then Cyc_{ooo} would equal zero, and the LLC miss penalty would equal the memory access penalty, as for the in-order case in Section 7.5.2.

The number of independent instructions that are executed during this period, of Cyc_{ooo}, is referred to as Ins_{ooo}, and indicated in the lower half of Figure 7.4. The connection between Cyc_{ooo} and Ins_{ooo} requires knowledge of the CPI over the period. Our analysis partitions the application execution in sampling segments of 1 second (detailed in Section 7.5.1), and considers there to be a steady average execution rate. So, during these execution segments the CPI equals its average rate of CPI_0, as detailed in Table 7.2 and the corresponding text. Therefore, Cyc_{ooo} can be calculated as:

$$Cyc_{\text{ooo}} = CPI_0 \times Ins_{\text{ooo}} \quad \text{for an isolated miss} \tag{7.8}$$

The factor of Ins_{ooo} depends on the number of independent instructions and the number of free instruction slots in the ROB. It is analyzed in detail in the next section.

Ins_{ooo} estimation

State-of-the-art architectures do not incorporate counters that can be used to measure the value of Ins_{ooo}. We therefore calculate bounds on its value and incorporate these bounds into the model's error estimate. We use the platform-specific parameters and the application's measured CPI.

The Ins_{ooo} lower bound is trivial: $Ins_{\text{ooo}} \geq 0$, since OOO execution may stop immediately after the LLC miss and continue being stalled until the requested miss data arrives.

The Ins_{ooo} upper bound is calculated as the lower of two constraints. The first is the reorder buffer size, Ins_{ROB}, which corresponds to the maximum number of instructions that can be stored in the ROB. The first bound is summarized as [127]:

$$Ins_{\text{ooo}}^{\text{max1}} = Ins_{\text{ROB}} \tag{7.9}$$

The ROB size is a characteristic on the target architecture. In our study we analyze two architectures, as described in Section 7.7. In Sandy Bridge EP-2670 CPU the ROB comprises 168 entries, while in Intel Knights Landing Xeon Phi 7230 it has 72 entries.

The second upper bound is determined by the maximum number of instructions that can be executed during the LLC miss. In this scenario, the whole Pen_{mem} is covered by the OOO execution, so Cyc_{ooo} would equal Pen_{mem}. Therefore, since Ins_{ooo} is calculated as Cyc_{ooo}/CPI_0 (Eg. 7.8), we can combine the two equations to find that the maximum number of instructions that the processor will execute in this time is Pen_{mem}/CPI_0. Since the second upper bound assumes that OOO execution covers the whole memory access penalty, there cannot be any stalls due to LLC misses, i.e., CPI_{LLC} would equal 0. Therefore, since the application's overall CPI is defined to be $CPI_{\text{tot}} = CPI_0 + CPI_{\text{LLC}}$, it must be (in this case) that CPI_{tot} equals CPI_0. Combining these facts the final form of the second upper-bound, given in terms of inputs to the model:

$$Ins_{\text{ooo}}^{\text{max2}} = Pen_{\text{mem}} \times \frac{Ins_{\text{tot}}}{Cyc_{\text{tot}}} \tag{7.10}$$

The overall upper bound on Ins_{ooo} is the minimum of the two limits:

$$Ins_{\text{ooo}}^{max} = \min\left(Ins_{\text{ROB}}, \quad Pen_{\text{mem}} \times \frac{Ins_{\text{tot}}}{Cyc_{\text{tot}}}\right) \tag{7.11}$$

Since the value of Ins_{ooo} can be anywhere between its bounds we consider Ins_{ooo} to be a free parameter and perform a sensitivity analysis when calculating other dependent parameters.

Overlapping LLC misses: Impact of memory level parallelism.

Previously, in Section 7.5.3, specifically in Eq. 7.7, we considered the case of an isolated read LLC miss. This section now considers the general case, in which after an LLC miss occurs, and while the corresponding instruction is waiting for data from memory, the CPU pipeline generates one or more additional LLC misses. This situation is illustrated in Figure 7.5. Any stall cycles that occur should be counted once per group of overlapping LLC misses rather than once per LLC miss, which was the case for Eq. 7.7. The number of concurrent LLC misses is typically known as the memory level parallelism, and is denoted MLP [118, 128]. The penalty per LLC miss is therefore given by the number of stall cycles divided by MLP [127, 128]:[5]

$$\begin{aligned} Stalls_{\text{LLC}} &= \frac{1}{MLP} \times (Pen_{\text{mem}} - Cyc_{\text{ooo}}) \\ &= \frac{1}{MLP} \times (Pen_{\text{mem}} - CPI_0 \times Ins_{\text{ooo}}) \end{aligned} \tag{7.12}$$

Karkhanis et al. [127] analyze this in detail and show that this is correct independently of the moment in which second, third, or subsequent LLC misses occur, as long as they occur within the Cyc_{ooo} interval. If MLP equals 1, then the above equation becomes identical to Eq. 7.7; so Eq. 7.12 covers both cases, that of isolated and overlapping LLC misses.

Current processors cannot directly measure MLP, so it must be estimated based on the parameters that are available. We derive lower and upper bounds on MLP and a point estimate.

The MLP lower and upper bounds can be computed starting from the equation $CPI_{\text{tot}}^{(1)} = CPI_0 + CPI_{\text{LLC}}^{(1)}$, then substituting $CPI_{\text{LLC}}^{(1)}$ from Eq. 7.1 and $Stalls_{\text{LLC}}^{(1)}$ from Eq. 7.12:

[5] LLC misses can also overlap with front-end miss events such as instruction cache miss, branch misprediction, etc. These overlaps, however, tend to be rare leading to an insignificant performance impact [129].

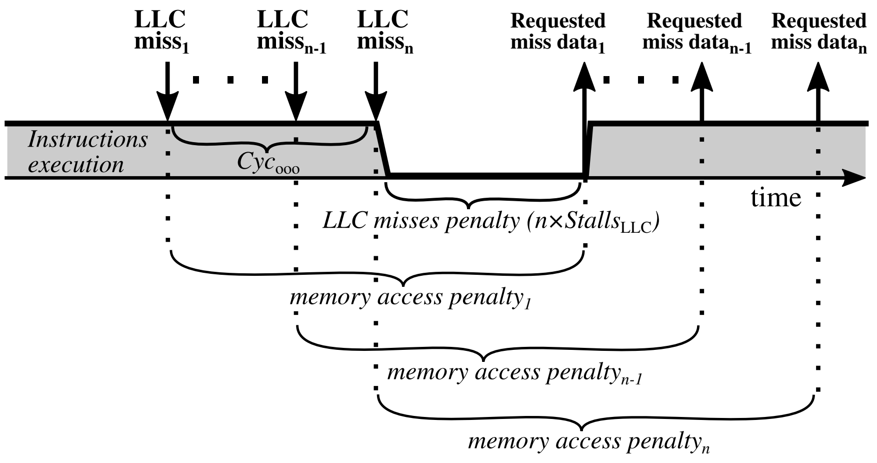

LLC miss$_1$
LLC miss$_{n-1}$
LLC miss$_n$
Requested miss data$_1$
Requested miss data$_{n-1}$
Requested miss data$_n$
Instructions execution
Cyc_{ooo}
time
LLC misses penalty ($n \times Stalls_{LLC}$)
memory access penalty$_1$
memory access penalty$_{n-1}$
memory access penalty$_n$

Figure 7.5: Handling overlapping LLC misses in an OOO processor: the penalty of a single miss is divided by a number of concurrent LLC misses [127].

$$CPI^{(1)}_{\text{tot}} = CPI_0 + \frac{Miss_{\text{LLC}} \times \left(Pen^{(1)}_{\text{mem}} - CPI_0 \times Ins_{\text{ooo}}\right)}{Ins_{\text{tot}} \times MLP} \tag{7.13}$$

Rearranging to isolate MLP and writing as a function to make clear which values are unknown gives:

$$MLP(Ins_{\text{ooo}}, CPI_0) = \frac{\frac{Miss_{\text{LLC}}}{Ins_{\text{tot}}} \times (Pen^{(1)}_{\text{mem}} - CPI_0 \times Ins_{\text{ooo}})}{CPI^{(1)}_{\text{tot}} - CPI_0} \tag{7.14}$$

This equation expresses MLP, which we want to know, in terms of Ins_{ooo}, the free variable that we will vary later, and CPI_0, which is unknown but can be bounded. The lower bound on CPI_0 is CPI_{min}, the reciprocal of the processor's *highest* theoretical IPC. The upper bound on CPI_0 is $CPI^{(1)}_{\text{tot}}$, since CPI_0 was defined to be one (of two) components contributing to $CPI^{(1)}_{\text{tot}}$. Now that CPI_0 is bounded, and assuming a value of Ins_{ooo}, it is possible to use Eq. 7.14 to obtain the range of potential values of MLP, either via a sweep on CPI_0 between its lower and upper bounds or using differential calculus.

A second upper bound on MLP is the size of the Miss Information Status Holding register (MSHR) [130]. The MSHR is the hardware structure that keeps information about in-flight cache misses, so they can be resolved once the corresponding data arrives. Its size is CPU-specific; e.g. it is 10 for Sandy Bridge [82] and 12 for KNL [131].[6]

The *MLP* point estimate is derived by assuming that the applica-

[6] In state-of-the-art processors, the MSHR is usually referred to as *Line Fill Buffer* (Intel), *Linefill Buffer* (ARM), *Load Miss Queue* (POWER), *Movein buffer* (SPARC), etc.

tion's behavior is uniform (in a sense to be clarified below) over the sampling segment (described in Section 7.5.1). Specifically, we assume that the number of LLC misses per instruction is homogeneous across the time segment, in which case it must equal $Miss_{\text{LLC}}/Ins_{\text{tot}}$. In the period between the LLC miss and the arrival of its data, the processor executes Ins_{ooo} instructions, so with a constant rate of LLC misses, the total number of *additional* LLC misses is $\frac{Miss_{\text{LLC}}}{Ins_{\text{tot}}} \times Ins_{\text{ooo}}$. The value of Ins_{ooo} is a free parameter, as described in Section 7.5.3, so the value being calculated here is a function of that parameter. In order to account for the first LLC miss, which has not yet been counted, the point estimate for the total number of LLC misses, as a function of Ins_{ooo}, to which the stall cycles must be attributed is:

$$\widehat{MLP}(Ins_{\text{ooo}}) = \frac{Miss_{\text{LLC}}}{Ins_{\text{tot}}} \times Ins_{\text{ooo}} + 1 \tag{7.15}$$

Note that $\widehat{MLP}(Ins_{\text{ooo}})$ is a point estimate for MLP based on the available information. If the point estimate is outside the valid range, between the lower and upper bounds described above, then it is corrected to lie in the range.

7.5.4 Performance as a function of latency

This section completes the analysis of out-of-order processor performance as a function of latency. We start by repeating Eq. 7.4, which gives the predicted CPI in terms of $Stalls_{\text{LLC}}$:

$$CPI^{(2)}_{\text{tot}} = CPI^{(1)}_{\text{tot}} + \frac{Miss_{\text{LLC}}}{Ins_{\text{tot}}} \times \left(Stalls^{(2)}_{\text{LLC}} - Stalls^{(1)}_{\text{LLC}}\right) \tag{7.4 again}$$

As remarked at the beginning of Section 7.5.3, in comparison with an in-order processor, an out-of-order processor has a more complex expression for $Stalls_{\text{LLC}}$, and this was given in Eq. 7.12:

$$Stalls_{\text{LLC}} = \frac{1}{MLP} \times (Pen_{\text{mem}} - CPI_0 \times Ins_{\text{ooo}}) \tag{7.12 again}$$

Finally we replace the MLP parameter in this equation with the point estimate in Eq. 7.15:

$$\widehat{MLP}(Ins_{\text{ooo}}) = \frac{Miss_{\text{LLC}}}{Ins_{\text{tot}}} \times Ins_{\text{ooo}} + 1 \tag{7.15 again}$$

In fact, as explained in Section 7.5.3, this value should be restricted to lie between the lower and upper bounds given in that section. This is done in the released source code, but for the sake of clarity we consider the more common case for which it is not necessary.

Combining Eq. 7.4, Eq.7.12 and Eq.7.15, and assuming that Ins_{tot}, $Miss_{\text{LLC}}$, CPI_0, Ins_{ooo} and MLP do not change when moving from one memory system configuration to another as detailed in Section 7.5.3, then $CPI^{(2)}_{\text{tot}}$ can be calculated as:

$$CPI^{(2)}_{\text{tot}} = CPI^{(1)}_{\text{tot}} + \frac{Pen^{(2)}_{\text{mem}} - Pen^{(1)}_{\text{mem}}}{Ins_{\text{ooo}} + Ins_{\text{tot}}/Miss_{\text{LLC}}} \tag{7.16}$$

This equation is written in terms of the memory access penalty, Pen_{mem}, but at the system level, outside a detailed analysis of a particular processor's pipeline, only Lat_{mem} is relevant. Recall that Pen_{mem} was defined to be the memory access latency, Lat_{mem} minus the cost of an LLC hit. We note, therefore, that the expression $Pen^{(2)}_{\text{mem}} - Pen^{(1)}_{\text{mem}}$ is equal to $Lat^{(2)}_{\text{mem}} - Lat^{(1)}_{\text{mem}}$. Taking account of this and rewriting in terms of the IPC instead of the CPI gives:

$$IPC^{(2)}_{\text{tot}} = \frac{IPC^{(1)}_{\text{tot}}}{1 + IPC^{(1)}_{\text{tot}} \times \frac{Lat^{(2)}_{\text{mem}} - Lat^{(1)}_{\text{mem}}}{Ins_{\text{ooo}} + Ins_{\text{tot}}/Miss_{\text{LLC}}}} \tag{7.17}$$

The various values in Eq. 7.17, $IPC^{(1)}_{\text{tot}}$, Ins_{tot}, and $Miss_{\text{LLC}}$ are known because they were measured on the baseline memory configuration. All other inputs to the model, such as Ins_{ROB}, $MSHR$, CPI_{min} (see Table 7.3) appear in the upper and lower bounds of Ins_{ooo}.[7]

Eq. 7.17 is plotted in Figure 7.6. The x axis is the target system memory latency, $Lat^{(2)}_{\text{mem}}$, and the y axis is the predicted IPC, $IPC^{(2)}_{\text{tot}}$. Eq. 7.17 is a function of the independent parameter Ins_{ooo}, which we cannot measure or calculate exactly. We bounded its value in the previous section, and varying it between the lower and upper bounds gives the family of curves shown in the figure. Note that the case of $Ins_{\text{ooo}} = 0$ corresponds to an in-order processor. This can be seen by comparing Eq. 7.17 and Eq. 7.6.

As indicated on the figure, when the target memory latency is the same as the baseline memory latency, $Lat^{(1)}_{\text{mem}}$, the model correctly "predicts" the measured IPC to be that of the baseline system, $IPC^{(1)}_{\text{tot}}$.

It is easy to be misled by Figure 7.6. For instance, a decrease in the memory latency by a fixed value, e.g. reducing the lead-off load penalty by 10 ns, is **not** equivalent to simply moving by 10 ns to the left on the x axis. This is because, as seen in the figure, such a change will result in an increase

[7] Or the upper and lower bounds of MLP which are not in Eq. 7.17 but considered in the full model.

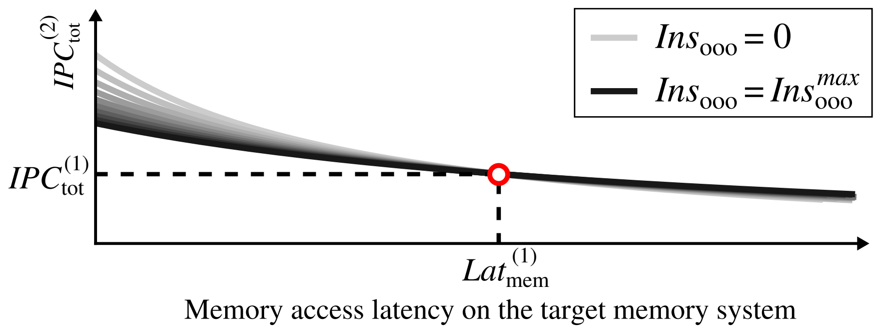


Figure 7.6: Performance as a function of the memory access latency. The different curves arise by varying the unknown parameter Ins_{ooo} within its bounds. Please read the text before interpreting this figure.

in the IPC, which will itself cause an *increase in the used memory bandwidth*, for reasons explained in the next section. This increase in used bandwidth will cause a movement to the right in the memory's bandwidth–latency curve and therefore *increase the memory system latency*, counterbalancing the original decrease in memory system latency. It is to this problem that we turn to in the next section.

7.5.5 Performance estimation — the ultimate step

This section completes the performance model. We start from the bandwidth–latency curves described in Section 7.4, which give the loaded memory access latency as a function of used memory bandwidth. We then combine these curves with the analysis in Section 7.5.4, which gives application performance as a function of the loaded memory latency. Doing so, in the right way, gives a prediction of the performance on the target memory system, with error bars.

The solution will be explained through Figure 7.7. The x axis is the used memory bandwidth and the y axis is the memory access latency. We show the measured bandwidth–latency curves for the Knights Landing platform, exactly as in Figure 7.3a. As before, the lightest curves correspond to 50% reads and 50% writes, and the darkest curves correspond to 100% reads. Now, however, since we are analysing a specific application segment, the proportion of reads is known (it is $Ratio_{R/W}$), so we know which curve from the family to select. The selected curve, which corresponds to $Ratio_{R/W}$, is shown as a dashed white curve.

We now turn to Figure 7.6, and use the latency–performance plot to construct a latency–*bandwidth* plot, i.e. to find the used memory *bandwidth* as a function of the memory access latency. This is because the total number

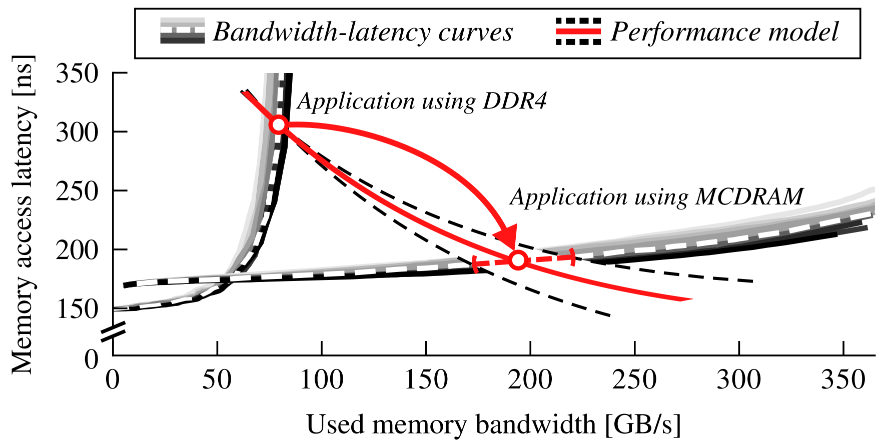


Figure 7.7: Graphical interpretation of a performance estimation as a merged solution of Sections 7.4 and 7.5.4.

of memory accesses performed over the application's execution is a constant, since it is the sum of the application's total number of read LLC misses and the prefetcher's total number of accesses, both of which are argued to be constant in Section 7.5.2. The total number of accesses can be evaluated, for the baseline memory system, by multiplying bandwidth by time, giving $BW^{(1)}_{\text{used}} \times (Cyc^{(1)}_{\text{tot}}/Freq_{\text{CPU}})$. Dividing this expression by the execution time on the target memory system gives:

$$BW^{(2)}_{\text{used}} = \frac{BW^{(1)}_{\text{used}} \times (Cyc^{(1)}_{\text{tot}}/Freq_{\text{CPU}})}{(Cyc^{(2)}_{\text{tot}}/Freq_{\text{CPU}})} = \frac{BW^{(1)}_{\text{used}}}{IPC^{(1)}_{\text{tot}}} \times IPC^{(2)}_{\text{tot}} \tag{7.18}$$

We therefore obtain a plot of bandwidth vs latency simply by multiplying the value on the y axis of Figure 7.6 by the factor $BW^{(1)}_{\text{used}}/IPC^{(1)}_{\text{tot}}$. The axes now match those in Figure 7.7 except they are transposed. We therefore transpose the bandwidth vs. latency plot, by swapping the x and y axes, and superimpose it onto Figure 7.7. This gives the family of lines for the performance model.

When the application runs on a memory system, it must be located on the memory system's bandwidth–latency curve *and* on one of the performance model curves that was just added. It must therefore be located on the intersection of these curves, as indicated on Figure 7.7. For the baseline memory system, we find that all performance model curves intersect the bandwidth–latency curve in the same place, at the bandwidth measured on the real system.

Each pair of bandwidth–latency and performance model curves will intersect in exactly one place. There cannot be more than one intersection because the memory system's bandwidth–latency curve is increasing (as a function of latency) whereas the application's performance model curve is decreasing (as a function of latency). In addition, there must be an intersection point, since the application's curve decreases from a very high latency necessary to get a small used memory bandwidth whereas the memory's bandwidth–latency curve increases to a very high latency close to the maximum sustainable bandwidth.

In summary, we start from the target memory system's bandwidth–latency curve and Eq. 7.17, which defines a family of performance model curves. We perform a sweep of the valid range for Ins_{ooo}, and for each value, find the intersection of its performance model curve with the target bandwidth–latency curve. To find this intersection we use the bisection method. This point gives a bandwidth on the y axis, which can be converted to an IPC by rearranging Eq. 7.18. Taking the minimum, maximum and average of these IPC values, as Ins_{ooo} varies, gives the minimum, maximum and point estimate for the performance prediction. Recall that the whole discussion so far is related to a single segment (time period) of the application. Performing these steps and averaging over time provides the final point estimate for the application run, plus error bars.

7.5.6 Novelties of the presented analytical model

As our analytical model is based on the CPI stack analysis widely used for performance modeling [52], it is important to emphasize the contributions of our work beyond the previous studies.

Computation of the CPI component that corresponds to the execution stalls due to the LLC misses (Eq. 7.1) is described by previous studies [128, 125, 52]. Hennessy and Patterson [52] and Karkhanis and Smith [127] also analyze execution of independent instructions Ins_{ooo} after the LLC miss (Eq. 7.7) and define some of the Ins_{ooo} bounds (Eq. 7.9). Finally, the MLP and its impact on the CPI stack analysis (Eq. 7.12). are also well explored by the community [127, 128, 132, 125, 133].

CPI stack, Ins_{ooo} and MLP are the foundation of various analytical models that quantify the performance impact of the main memory latency [127, 128, 132, 133]. The previous analytical models however have one great challenge: they require detailed application profiling, which can be performed only with hardware simulators. The main objective of our work is to avoid the use of the simulators, and to develop an analytical model based *only* on the parameters that can be obtained or derived from performance counters measurements on actual platforms. This requires a novel approach to the MLP estimate, presented in Eq. 7.13–7.15. In addition to this, we also present additional

Ins_{ooo} bounds in Eq. 7.10 and Eq. 7.11. Finally, to the best of our knowledge, this is the first study that combines analysis of the application performance as a function of the memory access latency (Sec 7.5.4) with the bandwidth–latency curves (Sec 7.5.5) and it proves that this analysis leads to a unique solution.

7.6 Power and energy modeling

Apart from performance, power and energy demand are important system constraints. In modern HPC systems, the memory subsystem contributes 10–16% of the total server power consumption [134]. It is therefore valuable to quantify the trade-offs in power and energy consumption due to the change of the memory system.

Similarly to performance model, we develop the power and energy models for the Sandy Bridge E5-2670 server.Based on the application profiling and memory power parameters, these models predict the variation of the system power and energy consumption due to the change of the memory systems. Detailed description of power and energy models are given in Sections 7.6.1 and 7.6.2.

7.6.1 Power modeling

We analyse the difference in total system power consumption when we move from baseline to target memory system. In our study, we assume that when the memory system changes, the biggest impact on total system power consumption is the change of the main memory power consumption. Hence, we focus on modeling the power consumption of the memory subsystem and consider that power consumption of the rest of the system does not change [135]. Estimating power and energy consumption requires measurements of the total platform power consumption, ratio of time spent in memory power-down states (active standby, precharge power-down and self-refresh states in our system) and row-buffer access statistics (rate of page hits and page misses). Table 7.4 summarizes the symbols used in the formulas for power and energy estimation.

To calculate the components of memory power consumption, we use Micron's guide for calculating power consumption of DDR3 memory systems [123]. Apart from the parameters in Table 7.4, Micron's power consumption guide requires *IDD* currents, memory system voltage and DIMM timing parameters, which are detailed in DIMMs documentation [136].[8] The power model was

[8] During the evaluation on different memory frequencies (DDR3-800/1066/1333/1600) we used the same DIMMs. Each memory frequency, however, uses different timing and *IDD* current parameters.

Table 7.4: Notation used in formulas: Power modeling

Description	Symbol
Input parameters	
Total platform power consumption	P_{tot}
Percentage of time spent in active standby state	t_{act}
Percentage of time spent in precharge power-down state	t_{ppd}
Percentage of time spent in self-refresh state	t_{sr}
Percentage of row-buffer hits	p_{hit}
Percentage of row-buffer misses	p_{miss}
Used memory bandwidth for read/write traffic	$BW_{\text{used}}^{rd/wr,(1)}$
Intermediate outputs	
Total memory power	P_{mem}
Power consumption of the rest of the system, apart from the memory	P_{rest}
Operational memory power	P_{op}
Background memory power	P_{bg}
Memory power in active standby state	P_{act}
Memory power in precharge power-down state	P_{ppd}
Memory power in self-refresh state	P_{sr}
Total memory read/write operations power	$P_{\text{rd/wr}}$
Power of memory refresh operations	P_{ref}
Duration of sampling segment	T_{sample}
Number of read/write memory accesses on a sampling segment	$N_{\text{access}}^{rd/wr}$
Energy on termination resistors for a single read/write memory access	$E_{\text{term}}^{rd/wr}$
Single memory read/write access energy	$E_{\text{access}}^{rd/wr}$
Energy of a read/write row buffer miss access	$E_{\text{miss}}^{rd/wr}$
Energy of a read/write row buffer hit access	$E_{\text{hit}}^{rd/wr}$

not developed for the KNL server, because we lacked the reliable MCDRAM power parameters. The estimation of these power parameters is part of ongoing work.

Micron's power consumption guide defines how to calculate power consumption of individual read or write memory accesses and DRAM power-down states. In our power analysis, we begin from total platform power consumption and expand its components down to these basic DRAM operations. We start by dividing the total platform power consumption into two components, power of the memory system and power of the rest of the platform apart from the memory [135]: $P_{\text{tot}} = P_{\text{mem}} + P_{\text{rest}}$. As in the analysis of the performance in Section 7.5, we use the superscripts (1) and (2) to distinguish between the P_{tot} and its components in the baseline and target memory systems, respectively:

$$\begin{aligned} \text{Baseline memory:} \quad & P^{(1)}_{\text{tot}} = P^{(1)}_{\text{mem}} + P^{(1)}_{\text{rest}} \\ \text{Target memory:} \quad & P^{(2)}_{\text{tot}} = P^{(2)}_{\text{mem}} + P^{(2)}_{\text{rest}} \end{aligned} \tag{7.19}$$

Since we assume that the power of the rest of the system stays the same, we can write $P^{(2)}_{\text{rest}} = P^{(1)}_{\text{rest}}$.[9] Therefore, total platform power consumption in the target memory system can be expressed as:

$$P^{(2)}_{\text{tot}} = P^{(1)}_{\text{tot}} + (P^{(2)}_{\text{mem}} - P^{(1)}_{\text{mem}}) \tag{7.20}$$

In further analysis we focus on the P_{mem}. It comprises two components, background power P_{bg} and operational power P_{op}:

$$P_{\text{mem}} = P_{\text{bg}} + P_{\text{op}} \tag{7.21}$$

Background power accounts for the current state of the memory system. There are several possible states of the memory system, depending on which power-down states are used. On our experimental system, there are three supported memory power states: *active standby*, *precharge power-down* and *self-refresh*. They differ in terms of power consumption and the transition latency to the active state. In *active standby state*, the memory device consumes the highest power P_{act} but executes the commands immediately, without any latency penalty. *Precharge power-down* state consumes power P_{ppd}, which is less than P_{act}, with a moderate latency penalty. In *self-refresh* mode, memory consumes the least power P_{sr}, but has a significant latency penalty for coming back to active standby mode. Multiplying each of these powers with the corresponding time share spent in each of them (t_{act}, t_{ppd} and t_{sr}, respectively),[10] and summing these products gives the background power P_{bg}:

$$P_{\text{bg}} = t_{\text{act}} \times P_{\text{act}} + t_{\text{ppd}} \times P_{\text{ppd}} + t_{\text{sr}} \times P_{\text{sr}} \tag{7.22}$$

Operational power presents the sum of power consumptions while reading or

[9] Quantifying the impact of a change from baseline to target memory system on P_{rest} is a part of ongoing work.

[10] Please note that $t_{\text{act}} + t_{\text{ppd}} + t_{\text{sr}} = 1$.

writing the data, plus the power consumption of the refresh operations:

$$P_{\text{op}} = P_{\text{rd}} + P_{\text{wr}} + P_{\text{ref}} \tag{7.23}$$

Components P_{rd} and P_{wr} present power consumptions of all read or write memory accesses, respectively, on a sampling interval. Expanding these components further leads to the power consumptions of individual read or write memory accesses, which can be calculated using the Micron's power consumption guide. However, using the power consumption of individual read or write memory accesses to calculate P_{rd} and P_{wr} is not trivial. These individual reads or writes are interleaved and overlapped in time. In order to sum the powers of individual reads or writes, we have to know their distribution on intervals which are the orders of magnitude of 1 ns, which is infeasible in current hardware platforms.

To mitigate this problem we calculate the energy of individual read or write memory access. Using energy instead of power implies that we do not have to know the distribution of memory accesses on a sampling segment to calculate the sum of energies of all the individual reads or writes. This way, we calculate P_{rd} and P_{wr} in two steps. First, we sum the energies of all the individual reads or writes on the sampling segment. Second, we divide this cumulative energy from the previous step with the duration of the sampling segment. If an energy of a single read or write memory access is $E^{rd/wr}_{\text{access}}$ and there are $N^{rd/wr}_{\text{access}}$ number of reads or writes on a sampling segment T_{sample}, we can write:

$$P_{\text{rd}} = \frac{\sum_{i=1}^{N^{rd}_{\text{access}}} E^{rd}_{\text{access,i}}}{T_{\text{sample}}} = \frac{E^{rd}_{\text{access}} \times N^{rd}_{\text{access}}}{T_{\text{sample}}} \tag{7.24}$$

$$P_{\text{wr}} = \frac{\sum_{i=1}^{N^{wr}_{\text{access}}} E^{wr}_{\text{access,i}}}{T_{\text{sample}}} = \frac{E^{wr}_{\text{access}} \times N^{wr}_{\text{access}}}{T_{\text{sample}}} \tag{7.25}$$

Number of reads or writes on the sampling segment can be measured with memory bandwidth hardware counters. A single read or write memory access transfers the amount of data defined as width of the memory bus (64 bits) multiplied by number of bursts (8), so 8 Bytes × 8 bursts = 64 Bytes of data. Therefore, number of reads or writes equals the total read or write traffic during the sampling segment, divided by the size of a single memory access:

$$N^{rd}_{\text{access}} = \frac{BW^{rd}_{\text{used}} \times T_{\text{sample}}}{64\text{ B}} \tag{7.26}$$

$$N^{wr}_{\text{access}} = \frac{BW^{wr}_{\text{used}} \times T_{\text{sample}}}{64\ \text{B}} \tag{7.27}$$

Using Eq. 7.26 and 7.27 with Eq. 7.24 and 7.25, we get:

$$P_{\text{rd}} = E^{rd}_{\text{access}} \times \frac{BW^{rd}_{\text{used}}}{64\ \text{B}} \tag{7.28}$$

$$P_{\text{wr}} = E^{wr}_{\text{access}} \times \frac{BW^{wr}_{\text{used}}}{64\ \text{B}} \tag{7.29}$$

Energy per single memory access accounts for read or write operation with its sub-operations. It also includes the energy on termination resistors, which are common in DDR devices. Our Sandy Bridge experimental platform uses *adaptive open-page policy* [50, 137], therefore performed sub-operations depend whether the target row in memory array was open or closed when accessing it. If the target row was open, it is a *row-buffer hit* access and it consumes only the energy for reading or writing the data. When the target row is closed, there are two scenarios. The first scenario is that there is no other opened row in the same bank and initially the energy accounts for opening a row and reading or writing the data. This row will be eventually closed after a time-out, so precharge energy should be added afterwards. The second scenario is that if there is an opened row (which is not the target one) in the same bank, it has to be closed first. It accounts the energy for precharging and closing the opened row, activating the target row and reading or writing the data. Both scenarios include same sub-operations from the energy point of view and we consider them as *row-buffer miss* case in further analysis. Since we measure the ratio of row-buffer hits p_{hit} and row-buffer misses p_{miss}[11] with regard to total number of accesses, the energy per memory access can be represented as:

$$E^{rd}_{\text{acc}} = E^{rd}_{\text{hit}} \times p_{\text{hit}} + E^{rd}_{\text{miss}} \times p_{\text{miss}} + E^{rd}_{\text{term}} \tag{7.30}$$

$$E^{wr}_{\text{acc}} = E^{wr}_{\text{hit}} \times p_{\text{hit}} + E^{wr}_{\text{miss}} \times p_{\text{miss}} + E^{wr}_{\text{term}} \tag{7.31}$$

The energy parameter E_{miss} represents the row-buffer miss energy, including opening the row, sending or receiving the data and precharging the row. E_{hit}

[11] Note that $p_{\text{hit}} + p_{\text{miss}} = 1$.

represents the row-buffer hit energy and it includes sending or receiving the data. E_{term} represents the energy on termination resistors. These parameters are calculated using Micron's power consumption guide.

Now that we have all the power components, we can include all of them from Equations 7.21 to 7.31 into Eq. 7.32 and calculate $P^{(2)}_{\text{tot}}$ on the target memory system. Before this step, we have to make two assumptions. First assumption is that the ratio of time spent in memory power-down states stays the same on baseline and target memory system. Hence, $t^{(2)}_{\text{act}} = t^{(1)}_{\text{act}}$, $t^{(2)}_{\text{ppd}} = t^{(1)}_{\text{ppd}}$ and $t^{(2)}_{\text{sr}} = t^{(1)}_{\text{sr}}$. Second assumption is that row-buffer access statistics does not change from baseline to target memory system. So, $p^{(2)}_{\text{hit}} = p^{(1)}_{\text{hit}}$ and $p^{(2)}_{\text{miss}} = p^{(1)}_{\text{miss}}$. These assumptions are reasonable, since we assume that Ins_{tot}, $Miss_{\text{LLC}}$, CPI_0, $Ratio_{\text{R/W}}$ and memory access pattern do not change from baseline to target memory system (detailed in Section 7.5.2). So, the final solution for $P^{(2)}_{\text{tot}}$ on the target memory system is given in the Eq. 7.32.

$$
\begin{aligned}
P^{(2)}_{\text{tot}} &= P^{(1)}_{\text{tot}} + (P^{(2)}_{\text{mem}} - P^{(1)}_{\text{mem}}) = P^{(1)}_{\text{tot}} + \big(P^{(2)}_{\text{bg}} + P^{(2)}_{\text{op}} - (P^{(1)}_{\text{bg}} + P^{(1)}_{\text{op}})\big) \\
&= P^{(1)}_{\text{tot}} + P^{(2)}_{\text{bg}} - P^{(1)}_{\text{bg}} + P^{(2)}_{\text{op}} - P^{(1)}_{\text{op}} \\
&= P^{(1)}_{\text{tot}} + P^{(2)}_{\text{bg}} - P^{(1)}_{\text{bg}} + P^{(2)}_{\text{ref}} - P^{(1)}_{\text{ref}} + P^{(2)}_{\text{rd}} - P^{(1)}_{\text{rd}} + P^{(2)}_{\text{wr}} - P^{(1)}_{\text{wr}} \\
&= P^{(1)}_{\text{tot}} + \overbrace{t_{\text{act}} \times (P^{(2)}_{\text{act}} - P^{(1)}_{\text{act}}) + t_{\text{ppd}} \times (P^{(2)}_{\text{ppd}} - P^{(1)}_{\text{ppd}}) + t_{\text{sr}} \times (P^{(2)}_{\text{sr}} - P^{(1)}_{\text{sr}})}^{P^{(2)}_{\text{bg}} - P^{(1)}_{\text{bg}} \quad \text{(Eq. 7.22)}} \\
&+ (P^{(2)}_{\text{ref}} - P^{(1)}_{\text{ref}}) + \overbrace{\frac{1}{64\text{B}} \times \big(BW^{rd,(2)}_{\text{used}} \times (E^{rd,(2)}_{\text{hit}} \times p_{\text{hit}} + E^{rd,(2)}_{\text{miss}} \times p_{\text{miss}} + E^{rd,(2)}_{\text{term}})\big)}^{P^{(2)}_{\text{rd}} \quad \text{(Eq. 7.28 and Eq. 7.30)}} \\
&\quad \underbrace{- \frac{1}{64\text{B}} \times \big(BW^{rd,(1)}_{\text{used}} \times (E^{rd,(1)}_{\text{hit}} \times p_{\text{hit}} + E^{rd,(1)}_{\text{miss}} \times p_{\text{miss}} + E^{rd,(1)}_{\text{term}})\big)}_{P^{(1)}_{\text{rd}} \quad \text{(Eq. 7.28 and Eq. 7.30)}} \\
&+ \overbrace{\frac{1}{64\text{B}} \times \big(BW^{wr,(2)}_{\text{used}} \times (E^{wr,(2)}_{\text{hit}} \times p_{\text{hit}} + E^{wr,(2)}_{\text{miss}} \times p_{\text{miss}} + E^{wr,(2)}_{\text{term}})\big)}^{P^{(2)}_{\text{wr}} \quad \text{(Eq. 7.29 and Eq. 7.31)}} \\
&\underbrace{- \frac{1}{64\text{B}} \times \big(BW^{wr,(1)}_{\text{used}} \times (E^{wr,(1)}_{\text{hit}} \times p_{\text{hit}} + E^{wr,(1)}_{\text{miss}} \times p_{\text{miss}} + E^{wr,(1)}_{\text{term}})\big)}_{P^{(1)}_{\text{wr}} \quad \text{(Eq. 7.29 and Eq. 7.31)}}
\end{aligned}
\tag{7.32}
$$

7.6.2 Energy modeling

Once we have the performance and the power consumption estimations, we can estimate the total system energy consumption with the target memory system. In general, energy is defined as the integral of power over time:

$$E_{\text{tot}} = \int_0^{t_{\text{tot}}} P_{\text{tot}}(t)\mathrm{d}t$$

In our experiments, we measure and analyse power consumption on sampling segments of 1 s. Hence, we represent total energy from our experiments in a discrete form:

$$E_{\text{tot}} = \sum_{i=1}^{N} P_{\text{tot,i}} \times \Delta t_{\text{i}} \tag{7.33}$$

The parameter Δt_{i} is the duration of the sampling segment, and equals ${\Delta t_{\text{i}}}^{(1)} = 1\,\text{s}$ on a baseline memory system (detailed in Section 7.5.1). During ${\Delta t_{\text{i}}}^{(1)}$ segment, $Ins_{\text{tot,i}}^{(1)}$ instructions are executed. As we mentioned in Section 7.5.2, we assume that Ins_{tot} does not change from baseline to target memory system, therefore $Ins_{\text{tot,i}}^{(1)} = Ins_{\text{tot,i}}^{(2)}$. However, duration of the corresponding time interval ${\Delta t_{\text{i}}}^{(2)}$ on the target memory system is not the same as ${\Delta t_{\text{i}}}^{(1)}$. This implies that ${\Delta t_{\text{i}}}^{(2)}$ on the target memory system is inversely proportional to the estimated performance improvement:

$$\Delta t^{(2)} = \frac{IPC_{\text{tot,i}}^{(1)}}{IPC_{\text{tot,i}}^{(2)}} \times {\Delta t_{\text{i}}}^{(1)} \tag{7.34}$$

Finally, total energy on the target memory system is:

$$E_{\text{tot}}^{(2)} = \sum_{i=1}^{N} P_{\text{tot,i}}^{(2)} \times \Delta t_{\text{i}}^{(2)} = \sum_{i=1}^{N} P_{\text{tot,i}}^{(2)} \times \frac{IPC_{\text{tot,i}}^{(1)}}{IPC_{\text{tot,i}}^{(2)}} \times {\Delta t_{\text{i}}}^{(1)} \tag{7.35}$$

Parameter $P_{\text{tot,i}}^{(2)}$ can be calculated using the Equation 7.32, and $IPC_{\text{tot,i}}^{(2)}$ can be calculated using the performance model from Section 7.5.

7.7 Experimental environment and methodology

In this section, we present the hardware platforms and benchmarks used in the model evaluation. We also list the tools used for the application profiling

Table 7.5: The most important features of experimental platforms

Platforms	**Sandy Bridge E5-2670**	**Knights Landing Xeon Phi 7230**
Sockets	2	1
Cores per socket	8	64
CPU freq. [GHz]	3.0	1.3
L1i, L1d	32 kB, 32 kB	32 kB, 32 kB
L2	512 kB	1 MB
L3	20 MB	/
Memory conf. per socket	4 chann. DDR3-800/1066/1333/1600	8 chann. MCDRAM 6 chann. DDR4-2400
Memory capacity	64 GB	16 GB MCDRAM 96 GB DDR4

and server power measurements. Finally, we summarize the main steps of the model evaluation process.

7.7.1 Hardware platforms

We evaluate the model on Sandy Bridge-EP E5-2670 and Knights Landing Xeon Phi 7230 platforms (see Sections 3.1.2 and 3.1.3). The most important features of the platforms are summarized in Table 7.5.

The Sandy Bridge-EP server is a representative of mainstream high-performance computing (HPC) servers, and it is still in use, especially in smaller Tier-0 systems [2]. In the server under study we were able to setup four memory frequencies: DDR3-800, DDR3-1066, DDR3-1333 and DDR3-1600, and we used these configurations to evaluate our performance, power and energy models.

The Intel KNL platform [65] is an emerging platform that combines two types of memory with different memory bandwidths and access latencies: DDR4 DIMMs and 3D-stacked MCDRAM [65]. Since it uses two types of memories, the system offers three modes of operation: *cache* mode, *flat* mode and *hybrid* mode. In our experiments, we use *flat* mode, in which the DDR4 and MCDRAM are configured as separate NUMA nodes, and we execute our workloads either in DDR4 or MCDRAM memory.

7.7.2 Benchmarks

We evaluated our model on a set of SPEC CPU2006 benchmarks (see Section 3.2.3) and scientific HPC applications from UEABS (see Section 3.2.4).

We choose four applications: ALYA, representative of the computational mechanics codes, and GROMACS, NAMD, and Quantum Espresso (QE), computational chemistry applications. The remaining UEABS applications could not be executed because their input dataset sizes exceed the main memory capacity of our hardware platforms.

In all the experiments on Sandy Bridge platform, we fully utilize the available 16 CPU cores:[12] we execute 16 copies of each SPEC CPU2006 benchmark, or 16 application processes for each UEABS applications. The KNL platform comprises 16 GB of the MCDRAM, which is insufficient to execute any of the UEABS applications. Also, for each SPEC CPU2006 benchmark, we had to determine the maximum number of instances whose cumulative memory footprint fits into MCDRAM. In the SPEC charts, this number of benchmark instances is specified in parentheses after the benchmark name.

In order to quantify the level of stress that our workloads put to the memory system, we measure their memory bandwidth. Figure 7.8 shows the bandwidth utilization, relative to the maximum sustained memory bandwidth,[13] for the platforms under study. When reporting the model evaluation in Section 7.8, we will emphasize the results for the high-bandwidth benchmarks with over 50% of the used memory bandwidth. These benchmarks are the most affected by the changes in the memory system, and the most challenging to model because they are located in the linear and exponential regions of the memory bandwidth–latency curves.

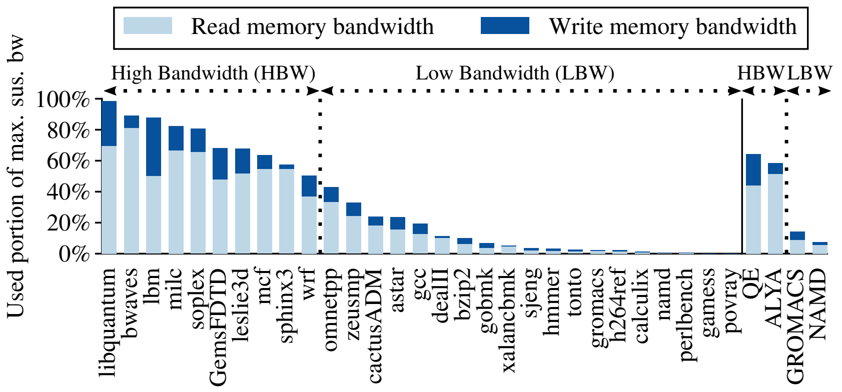
7.7.3 Tools and methodology

Application profiling requires measurements of the CPU cycles, instructions, LLC misses, read and write memory bandwidths, as well as the row-buffer access statistics, number of page activations and page misses, and number of cycles spent in memory power-down states. All these inputs are measured by the hardware counters and the LIKWID performance tool suite (see Section 3.3.1). The counters used in the study are widely available in mainstream HPC servers [61, 138].

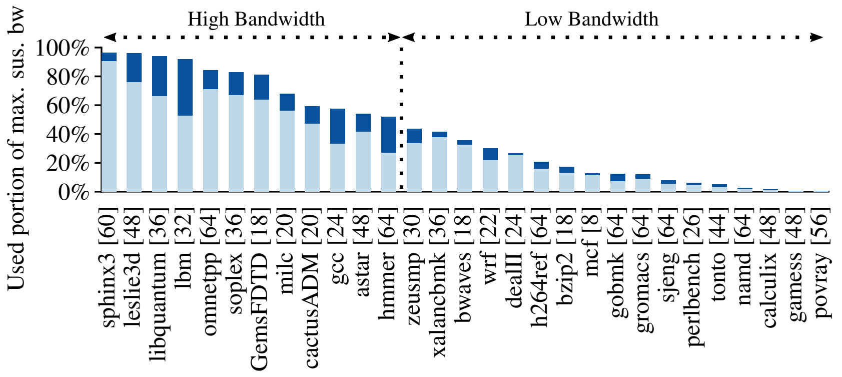
The UEABS applications mimic large-scale production HPC workloads; they have large input data-sets and can scale up to thousands of processes. Extracting the UEABS execution segments representative of the production runs is not trivial [139] and requires specialized HPC profiling and visualiza-

[12] Although the processors support hyper-threading at the core level, this feature is disabled, as in most HPC systems.

[13] The maximum sustainable memory bandwidth of our experimental platforms is measured with STREAM benchmark (see Section 3.2.2).



(a) Sandy Bridge E5-2670. Fully utilized server: 16 SPEC CPU2006 instances or 16 UEABS MPI processes.



(b) Knights Landing Xeon Phi. The MCDRAM capacity limits the number of the benchmarks instances, specified in the square brackets.

Figure 7.8: The workloads under study show a wide range of memory bandwidth utilization, and different ratios of the Read and Write memory traffic.

tion tools, such as Limpio, Extrae and Paraver, all of which were used in this study and described in Section 3.3.1.

We used a Yokogawa WT230 [140] power meter to measure the server power consumption. It measures the voltage and current at the power plug to calculate the server power consumption. The measurements were sampled on one second time period. The energy consumption was calculated by summing the power consumption over the execution time.

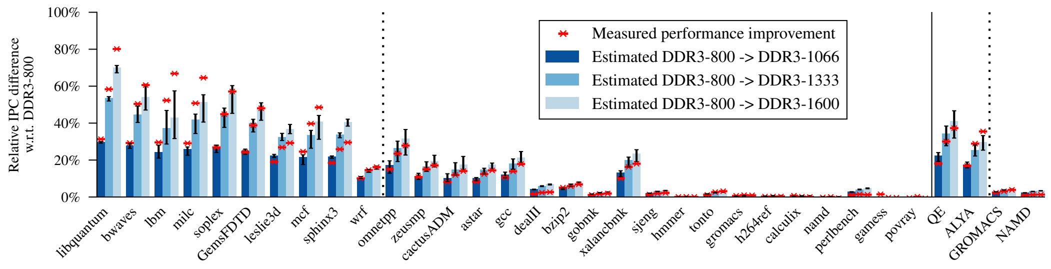


Figure 7.9: Sandy Bridge, DDR3-800→1066/1333/1600: Changing the DRAM frequency has a significant performance impact. Performance model estimations are precise, with low error bars, and accurate, with small difference from the values measured on the actual hardware.

7.8 Evaluation

The model evaluation is done in four steps. First, we execute a benchmark on the *baseline memory system*, e.g., Sandy Bridge server with DDR3-800. In this run, we measure the benchmark performance, power and energy consumption, and collect all the hardware counters needed for the model prediction. Second, we use the model to *estimate* the benchmark performance, power and energy on the *target memory configuration*, e.g., DDR3-1600. Third, we change the platform memory configuration from the baseline to the target memory, e.g., from DDR3-800 to DDR3-1600. This requires changing the BIOS settings for the Sandy Bridge, and changing the execution NUMA node for the Knights Landing platform. Finally, we *execute* the benchmark on the *target memory system*, measure the *actual* performance, power and energy consumption, and compare them with the values estimated by the model in Step 2.

7.8.1 Sandy Bridge: DDR3-800 → 1066/1333/1600

Performance results for the Sandy Bridge server are displayed in Figure 7.9. For each benchmark we plot three sets of bars, one per each DRAM frequency change: DDR3-800→1066/1333/1600. As described in Section 7.5.3, since we could not determine the exact value of the Ins_{ooo} parameter in the experimental platform, we performed a sensitivity analysis on this parameter — we changed the Ins_{ooo} values from 0 to Ins_{ooo}^{max}, and did the performance estimate for each of them. The solid bars correspond to the mean performance estimate, while the error bars show the lowest and highest estimated performance. In addition to the estimated values, we plot the performance improvement measured on the actual platform, marked with a cross marker for each experiment.

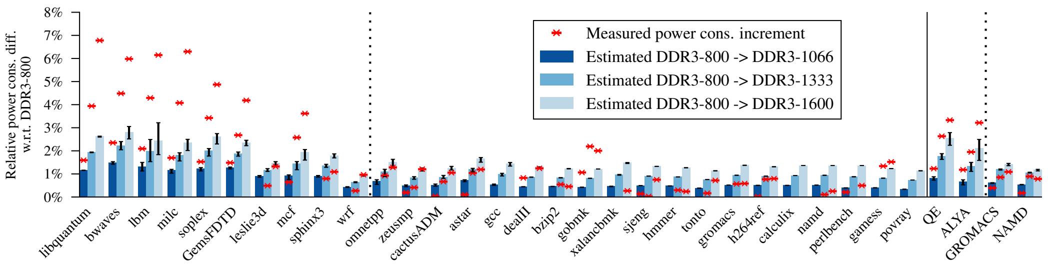


Figure 7.10: Sandy Bridge, DDR3-800→1066/1333/1600: DRAM frequency has a minor impact on the overall server power consumption. Increasing DRAM frequency from DDR3-800 to DDR3-1600 (100% increment) causes average power increment of only 2%.

First, we can see that the error bars, i.e., ranges of the estimated performance are narrow. Across the high-bandwidth benchmarks, the average width of the error bars is only 2.5%, 5.4% and 7.2%, for DDR3-1066/1333/1600 respectively. Across the low-bandwidth benchmarks, the average width of the error bars is even lower, at 1.1%, 1.8% and 2.1%. This means that, although for the architectures under study we cannot determine the precise value of Ins_{ooo} parameter, performance estimation based on the sensitivity analysis leads to narrow ranges of the estimated performance. Second, the model predictions are highly accurate. The average difference from the performance measured on the actual hardware for DDR3-1066/1333/1600 frequencies is 1.8%, 3.9% and 5.3% for the high-bandwidth benchmarks, and it drops to just 1%, 1.3% and 1.7% over the low-bandwidth benchmarks. Finally, the presented results show that the DRAM frequency increase indeed has a significant performance impact. For example, increasing the DRAM frequency from the baseline DDR3-800 to the target DDR3-1600 causes average performance improvement of 22%, and it reaches 80% for the *libquantum* benchmark. Therefore it is important to understand the relation between the available memory bandwidth and the overall application performance, which is the main objective of our work.

System power

Next we present the evaluation results for system-level estimation of power and energy consumption. Figure 7.10 shows the estimated and measured system power. As in Figure 7.9, for each benchmark we plot three sets of bars, one per DRAM configuration: DDR3-800 → 1066/1333/1600. Also, as in all previous evaluation charts, we plot the model point estimate (solid bars), estimation bounds (error bars), and the power consumption measured on the actual server (cross markers).

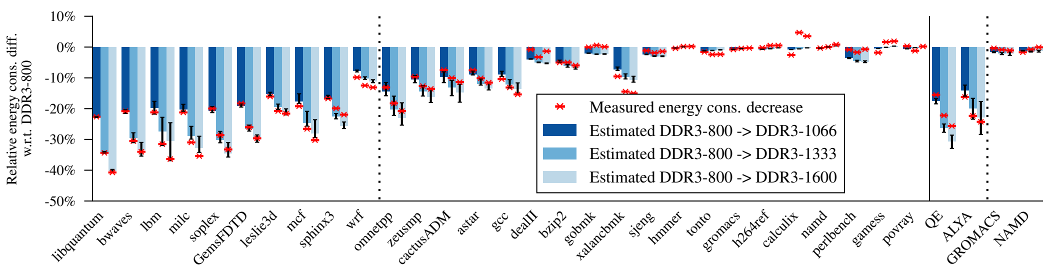


Figure 7.11: Sandy Bridge, DDR3-800→1066/1333/1600: DRAM frequency has a significant impact on the system energy consumption. Energy predictions of our model are precise, with low error bars, and accurate, with small difference from the actual values measured on real hardware.

Although some of the high-bandwidth benchmarks such as *libquantum* to *GemsFDTD* show a moderate prediction error of 3–4%, in general the power prediction error of the model is small, below 1% on average. The most important finding of the results presented in Figure 7.10 is that the significant increment in the DRAM frequency causes very small change in the overall server power consumption. For example, increasing the DRAM frequency from DDR3-800 to 1600 (100% increment) causes average power increment of only 2%. Even if we focus on the high-stress memory benchmarks, the power increment is still below 5%. This is expected since, as assumed in our power model, when changing the DRAM frequency the most important impact on total system power consumption is the change of the main memory power consumption. Although the relative change of the memory power itself could be significant, this is still a small portion of the overall server power [134].

Energy consumption

Estimated and measured changes in the system energy consumption when increasing the DRAM frequency from DDR3-800 to DDR3-1066/1333/1600 are given in Figure 7.11. The results show that the DRAM frequency has a significant impact on the overall energy consumption. For example, increasing the DRAM frequency from DDR3-800 to DDR3-1600 leads to average energy savings of 13% and reaches 41% savings for the *libquantum* benchmark. The results also show that the model energy predictions are precise, with narrow error bars, and accurate, with average prediction error of below 2%.

The presented results and findings are expected. Our previous results showed that increasing the DRAM frequency causes significant execution time reductions, and only few percent server power increment. Since, energy is the integral of the power consumption over the application execution time,

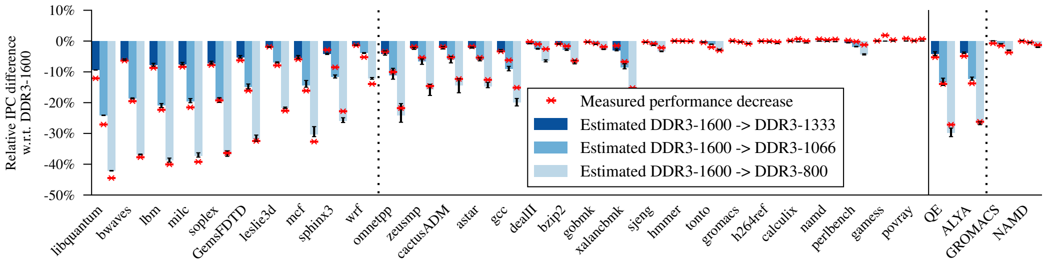


Figure 7.12: Sandy Bridge, DDR3-1600→1333/1066/800: Decreasing the DRAM frequency significantly decreases performance, especially for high-bandwidth workloads. Performance predictions of the model are precise and accurate.

it is reasonable to expect significant energy savings. Also, it is anticipated that the energy predictions are precise and accurate, since they are derived from the performance and power estimations.

7.8.2 Sandy Bridge: DDR3-1600 → 1333/1066/800

In all previous experiments, we analyzed the impact of the DRAM frequency increment. In this section, evaluate the model performance, power and energy predictions when the DRAM frequency is decreased, from DDR3-1600 to DDR3-1333/1066/800.

Figure 7.12 shows the measured and estimated system performance. As one may expect based on our previous results, reducing the DRAM frequency has a significant performance impact. Precision and the accuracy of the model is high, with average error of below 2%. Estimations and measured difference of the server power consumption are displayed in Figure 7.13. As expected based

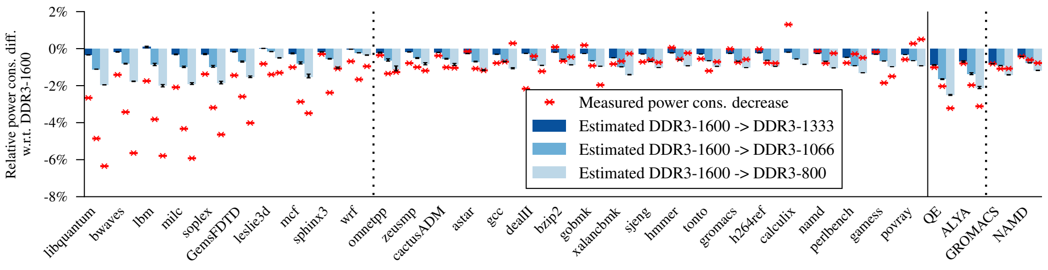


Figure 7.13: Sandy Bridge, DDR3-1600→1333/1066/800: Changing the DRAM frequency has a minor impact on the overall server power consumption. Decreasing the DRAM frequency from DDR3-1600 to DDR3-800 causes average power decrement of only 2%.

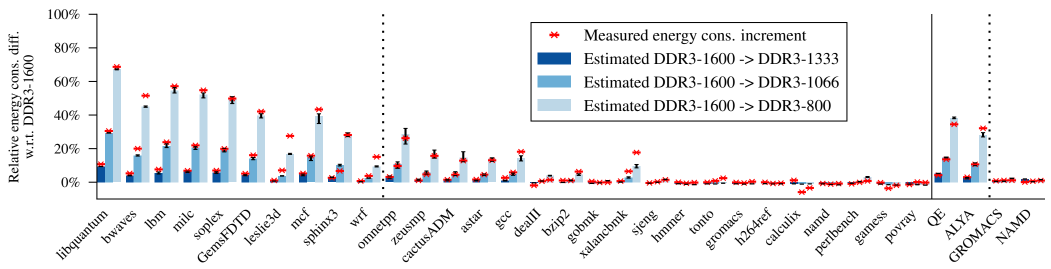


Figure 7.14: Sandy Bridge, DDR3-1600→1333/1066/800: Changing the DRAM frequency has a significant impact on the energy consumption. Energy predictions of the our model are precise, with low error bars, and accurate, with small difference from to the values measured on the actual hardware.

on the previously-presented results (see Section 7.8.1), significant decrement in available memory bandwidth causes small changes in overall server power consumption. As in Figure 7.10, some of the high-bandwidth benchmarks, show a moderate prediction error of up to 4%. Analysis of this error is a part of ongoing work. Still, the power prediction error of the model is small, below 1% on average. Finally, Figure 7.14 shows the estimated and measured energy consumption. Reducing the DRAM frequency leads to significant increment in the overall energy consumption. The average energy increment is 19%, and it ranges up to 69% for the *libquantum* benchmark. The energy predictions of the our model are precise, with low error bars, and accurate, with small difference from to the values measured on the actual hardware.

Overall, the results presented in this section show that the model shows high precision and accuracy when predicting the impact of the decreasing DRAM frequency, i.e., that the model can be used to predict system performance, power and energy consumption in both scenarios, increasing and decreasing of the available memory bandwidth.

7.8.3 Step further. Sandy Bridge: DDR3-1600 → DDR3-1866/2133

The main focus of the previous results sections was the evaluation of the model predictions w.r.t. performance, power and energy measurements from the actual hardware. In this section, we demonstrate how the model can be used to explore new memory configurations that cannot be configured in the existing hardware. We illustrate this capability by exploring the behavior of the Sandy Bridge platform with DDR3-1866 and DDR3-2133 memory systems. DDR3-1866 and DDR3-2133 are existing memory configurations, but they cannot be used with the Sandy Bridge system because they exceed

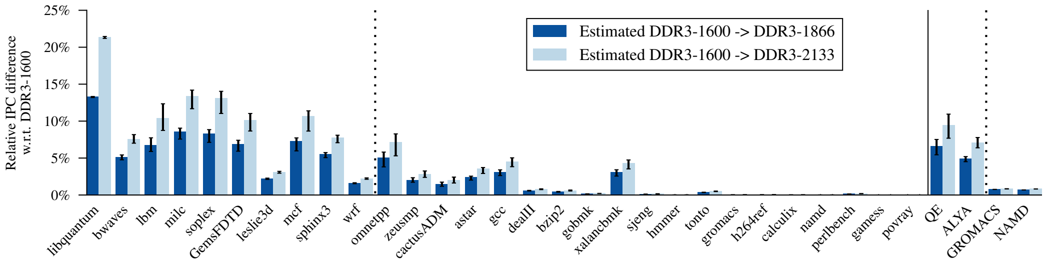


Figure 7.15: Sandy Bridge, DDR3-1600→1866/2133: Increasing DRAM frequency beyond the limits of the current system to DDR3-1866 and DDR3-2133 significantly benefits performance, up to 13% and 21% for high-bandwidth benchmarks.

the maximum DRAM frequency supported by the processor.

Figure 7.15 shows the performance estimations for a hypothetical case of the DDR3-1866 and DDR3-2133 integrated into the Sandy Bridge platform under study. The results are plotted relative to the DDR3-1600, which is the highest DRAM frequency available in the actual system. The average estimated performance improvement of the DDR3-1866 and DDR3-2133 is 3% and 4.5%, respectively, and it reaches 13% and 21% for the high-bandwidth benchmarks, respectively. Power consumption estimates are displayed in Figure 7.16. As expected, the power increase is small, between 1% and 2%. Finally, Figure 7.17 shows the estimated energy consumption. Increasing the memory frequency to DDR3-1866 and DDR3-2166 reduces the energy consumption by 2.6% and 3.3% on average, and up to 10% and 16% for the high-bandwidth benchmarks.

The most important outcome of this analysis, however, are not the estimated performance, power and energy improvements, but the demonstration how easy it is to explore memory configurations that are beyond the limits

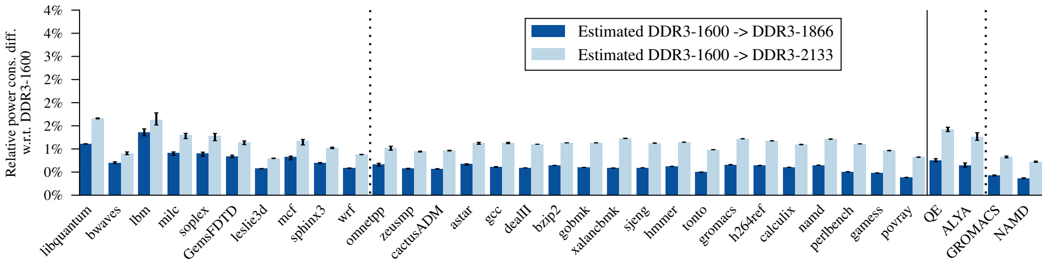


Figure 7.16: Sandy Bridge, DDR3-1600→1866/2133: Increasing the DRAM frequency beyond the limits of the current system to DDR3-1866 and DDR3-2133 causes average power increment between 1% and 2%.

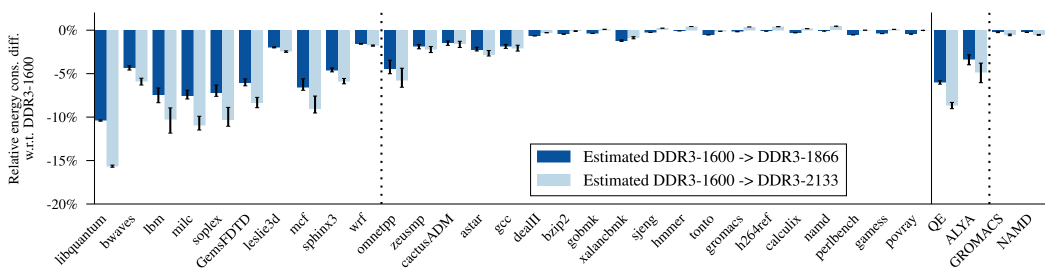


Figure 7.17: Sandy Bridge, DDR3-1600→1866/2133: Increasing DRAM frequency beyond the limits of the current system to DDR3-1866 and DDR3-2133, impacts energy consumption of the whole system significantly, saving up to 10% and 16% for high-bandwidth benchmarks.

of the existing system. Application execution can be instrumented on the CPU under study running with an existing memory system, and performance prediction on the new memory systems can be performed as soon as the memory bandwidth–latency curves of the target memory system are available. Power and energy predictions additionally requires only the memory power parameters detailed in Section 7.6. All the required memory system parameters are publicly available for most of the existing and emerging memories, and can be provided by the memory manufacturers for the technologies in an early development stage.

7.8.4 Knights Landing: DDR4-2400 → MCDRAM

Figure 7.18 shows the estimated and measured performance improvement of the KNL platform with high-bandwidth MCDRAM with respect to the DDR4-2400 memory. Again, our model shows high precision. Actually, the width of the estimation error bars is only 0.7%, which is significantly smaller from the Sandy Bridge system. The main reason for this is the 72-instructions reorder buffer in the KNL platform, compared to 168 instructions in the Sandy Bridge. The smaller the reorder buffer, the smaller the range of the Ins_{ooo} sensitivity analysis, and therefore the more precise performance prediction.

The MCDRAM provides 4.2-fold higher bandwidth over DDR4, which leads to significant performance improvement for the high-bandwidth benchmarks, up to 212% improvement for the *leslie3d*. However, the MCDRAM also has a 23 ns higher lead-off latency (see Figure 7.3a), that penalizes benchmarks with low and moderate memory bandwidth requirements. Actually, for *bzip* and *mcf*, execution on the MCDRAM leads to performance loss. For the *mcf* benchmark, this performance loss reaches a non-negligible 9%.

Despite of the wide range of the performance variation, between −9% and

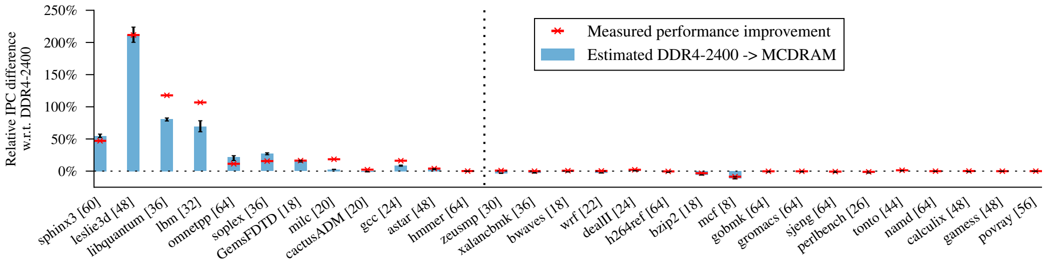


Figure 7.18: Knights Landing, DDR4-2400→MCDRAM: Despite of the wide range of the performance variation, between −9% (*mcf*) and 212% (*leslie3d*), the model shows high accuracy. Smaller KNL reorder buffer leads to a smaller range of the Ins_{ooo} sensitivity analysis, and therefore more precise the performance prediction, i.e., smaller error bars.

212%, when moving from DDR4 to the MCDRAM, the model shows high accuracy. The difference between the model performance estimates and the measurements on the actual hardware is 7% for high-bandwidth workloads, and drops down to 1.6% for the low-bandwidth benchmarks. Also, the model predictions accurately distinguish between the benchmarks that significantly benefit from the MCDRAM, and the benchmarks that show negligible performance improvements or even performance loss. This confirms that the model properly considers both segments of the memory bandwidth–latency curves (Figure 7.3a): the constant latency segment, close to the lead-off memory latency, and the exponential segment, close to the memory bandwidth saturation point.

7.8.5 Knights Landing: MCDRAM → DDR4-2400

Next, we analyze the model performance predictions on the KNL server when moving from the high-bandwidth MCDRAM to DDR4-2400. The results are plotted in Figure 7.19. As already discussed in Section 7.8.4 performance prediction when switching between MCDRAM and DDR4 is challenging. The MCDRAM provides 4.2× higher bandwidth, but it also has 23 ns higher lead-off latency. Proper performance estimate, therefore, requires accurate modeling of both extremes of the memory bandwidth–latency curve (Figure 7.3a): the constant latency region close to the lead-off memory access latency, and the exponential region close to the memory bandwidth saturation point.

As one may expect, the results in Figure 7.19 show that moving from MCDRAM to DDR4 penalizes the high-bandwidth benchmarks. The measured performance loss reaches up to 68% for the *leslie3d*. For the benchmarks with moderate or low memory bandwidth requirements, however, moving to the DDR4, leads to a small performance impact, or even performance

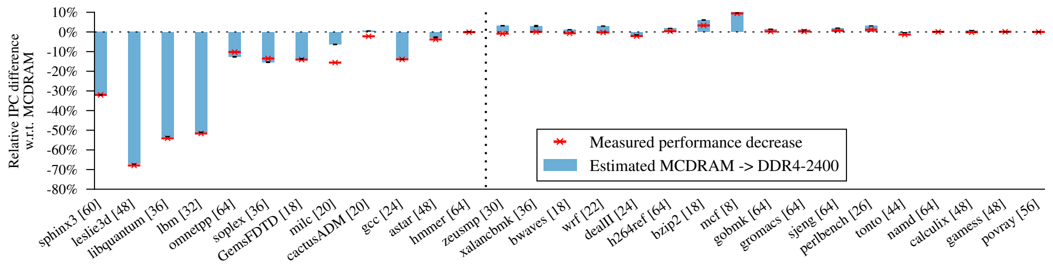


Figure 7.19: Knights Landing, DDR4-2400→MCDRAM: Despite of the wide range of the performance variation, between −68% (*leslie3d*) and 9% (*mcf*), the model shows high accuracy: The average estimation error is below 2%.

improvement, up to a non-negligible 9% for the *mcf* benchmark.

Despite of the wide range of performance variation, the model shows high accuracy, with an average error of below 2% The model also shows high precision, illustrated with narrow estimation error bars, as already discussed in Section 7.8.4. Finally, model predictions clearly distinguish between the benchmarks that are considerably penalized when moving to the DDR4, and the benchmarks that show no performance difference or even experience performance improvement.

7.8.6 Model vs. Hardware Simulator

Novel memory systems are typically explored using hardware simulators. In this section we compare the performance estimates of our model with ZSim+DRAMSim2 hardware simulators.

Experimental

We compared the results of hardware simulators and our model on the Intel Sandy Bridge platform described in Section 7.7.1. Although the Sandy Bridge is a main-stream HPC architecture released eight years ago (January 2011), finding a CPU simulator that accurately models this architecture was not trivial. After extensive search and analysis of the available options, we decided to use the ZSim simulator. Different main memory configurations were simulated with DRAMSim2. Both simulators are described in Section 3.3.2.

The accuracy of the model and the ZSim+DRAMSim2 were compared on an example of increasing memory frequency from DDR3-800 to DDR3-1066/1333/1600. To keep the simulation time at the acceptable level, we had to focus on the CPU2006 benchmarks and exclude the HPC scientific applications. In all the experiments we fully utilize the available 16 cores of the

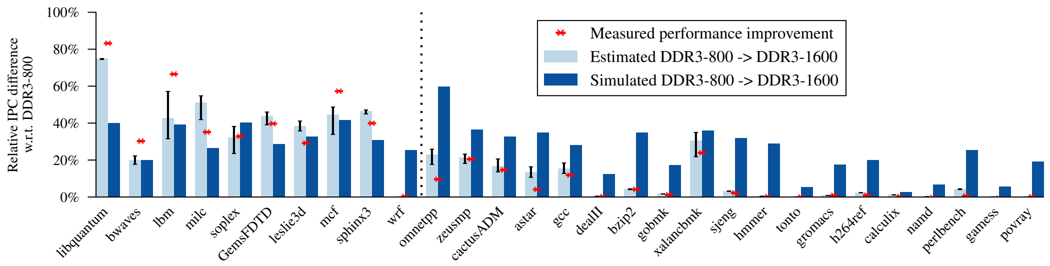


Figure 7.20: Sandy Bridge, DDR3-800→1600: Comparison of the estimated (our model) and simulated (ZSim+DRAMSim2) performance improvement. Estimations of our model correspond to the real-system measurements much better than the simulated performance.

Sandy Bridge platform by executing 16 benchmark copies. The simulations were limited to 150 billion instructions of each benchmark. In order to make a fair comparison with the simulator, the benchmark execution on the actual system and the corresponding model predictions were done for the same 150 billion instructions.

Results

In Figure 7.20, we compare the measured, simulated and estimated performance improvement when increasing the DRAM frequency from DDR3-800 to DDR3-1600. The DDR3-800→DDR3-1066/1333 results show the same trend and lead to the same conclusions.

The actual measured values (red cross markers) show significant performance improvement for the high-bandwidth benchmarks (left hand-side of the chart) and no performance changes for the low-bandwidth benchmarks (right hand-side of the chart). The model estimations are accurate, with an average error of 3.6%, and closely follow the trend of the actual measurements. Also, for the benchmarks for which the estimation error is moderate, e.g., *lbm* and *milc*, the estimated performance have high error bars, clearly indicating a limited estimation precision in these cases. The simulated performance shows discrepancy with the actual measured values. The average simulation error is 15.7%, which is significant considering that the average DDR3-800→DDR3-1600 performance improvement is 17.5%. The range of the simulator error is also very high, between −23.7% (*libquantum*) and 45.7% (*omnetpp*). Finally, trend of the simulated results is completely different from the actual one. As already mentioned, high-bandwidth benchmarks experience high performance improvement, and insignificant performance changes for the low-bandwidth benchmarks. The simulator underestimates performance gains of the high-bandwidth benchmarks (left hand-side of the chart) while it overestimates gains for the low-bandwidth benchmarks (right hand-side of the chart). Therefore, the simulated performance gains are roughly uniform over the benchmark suite, which is a completely different trend from the actual measurements.

7.8.7 Discussion

In addition to the better accuracy, the model has various advantages over hardware simulators. The model is faster than the hardware simulators by three orders of magnitude, so it can be used to analyze production HPC applications, arbitrarily sized systems, and numerous design options. In this study we presented experiments on two hardware platforms, Sandy Bridge and KNL. We analyzed six memory configurations for Sandy Bridge (DDR3-800/1066/1333/1600/1866/2166), and two for the KNL (DDR4-2400 and MCDRAM). In all the experiments, we analyzed all the benchmarks from the SPEC CPU2006 suite. For the Sandy Bridge platform, we also analyzed power and energy consumption in each memory configuration, and four HPC production applications. Finally, in most memory configurations, all but Sandy Bridge DDR3-1866/2166, we analyzed the impact of both, increasing and decreasing DRAM frequency. Performing the study of this size by using hardware simulators would be impossible within a practical length of time.

Additionally, the method is based on profiling of the application's memory behavior, so it does not require detailed modeling of the CPU as it already takes account of the real (and not publicly disclosed) data prefetcher and out-of-order engine. Therefore, it can be used to model various platforms as long as they support the required application profiling. The model was initially developed for the Sandy Bridge platform, and later we evaluated it for the KNL server. Adjustment of the model to the KNL system was trivial, as it required changes to only a few hardware parameters, such as, for example the reorder buffer size.

7.9 Related work

Numerous studies propose CPU and memory analytical models as an alternative to cycle-accurate hardware simulation. Next, we summarize the ones that are directly related to our work.

Karkhanis and Smith [127] present an OOO processor model that estimates the performance impact of instruction window size, branch misprediction, instruction cache misses and data cache misses. The model is validated versus detailed superscalar OOO CPU simulation and the authors conclude that, although the model provides much less data than detailed simulation, it still provides insights on what is going on inside the processor. Published in 2004, this work became the foundation for numerous advanced processor modeling approaches. Eyerman et al. [132] extend the work of Karkhanis and Smith [127] and develop the *mechanistic model and interval analysis*. The analysis breaks the total execution time into intervals based on the miss events, branch mispredictions and TLB/cache misses, and then predicts the execution time of each interval. Genbrugge et al. [141] propose *interval simulation* which combines the high-level mechanistic analytical model [132] and detailed simulation to accelerate multi-core simulation. The mechanic analytical model is used to estimate the core-level performance between two miss events, while the miss events are determined through the simulation of branch predictor and the entire memory hierarchy: private per-core caches and TLBs, shared caches, cache coherence, network on chip, memory controller, and main memory. Finally, Van den Steen et al. [133] present various enhancements of the interval model [132]. First, the study incorporates architecture-independent application profiling, so the application can be profiled once and then simulated on any given platform. The authors also demonstrate that the analytical model can be connected to the McPAT power tool [142] for estimation of the processor power and energy consumption. Finally, the authors make first steps in memory system modeling by considering congestion on the memory bus. Unlike our study, however, they do not take into account contention in the memory controller and memory device itself, which are more challenging.

Gulur and Manikantan [143] focus on the main memory access latency, and propose modeling it as an open queuing network without blocking. The queuing network parameters are workload specific, so the model requires a detailed workload profiling that monitors parameters such as the memory requests arrival rate, row-buffer hit rate, bank-level parallelism and the ratio of accesses that go to the idle banks. The authors perform the application profiling by executing it on simulation infrastructure that comprises M5 [144] and an in-house DRAM simulator.

The greatest challenge of the presented modeling approaches is that, although they do not require simulation to perform performance prediction, they do require detailed application profiling, which can be performed only with hardware simulators. Even with recent advances in hardware performance counters, we are still far away from being able to read values such as, for example, the number of cold, capacity and conflict cache miss. Since they require simulation results, even for application profiling, a significant effort is required to set up and tune for a target architecture, a serious amount of simulation time, and potentially high simulation errors. Our approach avoids these issues by using only those parameters that can be read using performance counters on real hardware. This is the greatest advantage of our work compared with previously-mentioned studies. Limiting the application profiling on the performance counters available on real hardware requires a novel approach to infer various application parameters, such as the MLP or number of executed OOO instructions. Therefore, although we start with the same foundation as the previous studies, as detailed in Section 7.5.6, estimation of the important application parameters and the overall performance calculation are fundamentally different from previously-presented models.

The second important difference of our work compared with the previous studies is the treatment of the main memory access latency. The previous studies use the memory access latency obtained from a detailed simulation of the whole memory hierarchy [141] or even use a constant latency [127, 132, 133]. Our study performs a detailed analysis of memory bandwidth-latency curves and uses these curves as an integral part of the overall performance estimation, as detailed in Section 7.5.5.

Third, unlike previous studies, our model does consider data prefetching. This is an important difference because prefetching may have significant impact on the application performance, behavior and memory bandwidth usage.

Forth, we complement the performance model with power and energy consumption estimates, and evaluate all three models against fully-utilized actual HPC servers with multi-threaded or multi-programmed benchmark execution. The previous models are validated versus the same simulators used for the application profiling. This means that the previous evaluations overlook potentially high errors of application profiling on a simulation versus the actual hardware. Finally, most of the previous studies [127, 132, 133] build and validate the models for single-core processors.

The performance impact of memory bandwidth and latency is frequently estimated by workload characterization studies and memory DVFS proposals. Since there is no publicly-available memory model that can be used to quantify this impact, some studies develop their own models to support some segments of the study (these models were required to run some of the experiments) [125, 135].

Clapp et al. [125] analyze behavior of big data workloads and positions them with respect to enterprise and HPC workloads. Different classes of workloads are compared in terms of CPU utilization, CPI, used memory bandwidth, and estimated performance impact of increasing memory latency or reducing memory bandwidth. To develop the model, they measure CPU utilization, CPI, on a real hardware, using Intel Xeon 2600 server with DDR3 main memory, while the used memory bandwidth is calculated analytically. Afterwards, memory latency and bandwidth sensitivity analysis is based on an analytical model developed in the study. The model also considers the OOO execution and the MLP, and mimic them with a parameter that the authors refer to as the *blocking factor*. The authors recognize the complexity of measuring the blocking factor on the real hardware, and try to estimate it indirectly based on the execution of the same workload on different CPU frequencies, using a linear fit. The authors propose to use the developed model to calculate the impact on future memory systems, taking into account the bandwidth-latency dependency (measured with the Intel Memory Latency Checker (MLC) benchmark)[145]. The memory bandwidth-latency analysis does not account for the differences between read and write memory bandwidth. However, they do not carry out such an analysis and instead perform the sensitivity analysis by reducing the available memory bandwidth in steps of 0.5 GB/s per core, and by increasing the memory access latency in steps of 10 ns.

Deng et al. [135] propose *MemScale*, a scheme for dynamic voltage and frequency scaling (DVFS) on the memory controller, and dynamic frequency scaling (DFS) on the memory channels and DRAM devices. As a part of the *MemScale*, the authors develop a model that predicts relationship between the application performance and the memory frequency. The model targets a simplified case of an in-order processor with no prefetcher. The application performance estimate is based on the hardware counters not available in current architectures, so the authors carefully analyze the possibility and the cost of including these counters into future server-class processors.

These models are developed for very specific tasks in the context of the larger studies and they successfully fulfill their objectives. Still it is questionable whether they can be applied generally because of two main limitations. First, the modeled CPU and memory systems are much simpler than state-of-the-art production platforms. Second, the models are not validated versus any real hardware or hardware simulators, so it is difficult to quantify the errors they may introduce.[14] Even so, we consider these studies as very valuable for any follow-up on this topic because they analyze different approaches for main memory modeling and share their experiences.

7.10 Summary

This study presents an analytical model that quantifies the impact of the main memory on application performance and system power and energy consumption. The model is based on memory system profiling and instrumentation of an application execution on a real platform with a baseline memory system. By running on the real platform, it takes account of the real (not publicly

[14] Deng et al. [135] do evaluate the proposed DVFS scheme and show significant energy savings, but do not perform any evaluation of the presented memory model.

disclosed) prefetcher and out-of-order engine. The model outputs are the predicted performance, power and energy consumption on the target memory.

The model is evaluated on two actual platforms: Sandy Bridge-EP E5-2670 and Knights Landing Xeon Phi platforms with various memory configurations. The evaluation results show that the model predictions are very accurate — the average difference from the performance, power and energy measured on the actual hardware is only 2%, 1.1% and 1.7%.

We also compare the model's performance predictions with simulation results for the Sandy Bridge-EP E5-2670 system with ZSim and DRAMSim2 simulators. The model shows significantly better accuracy while being three orders of magnitude faster than the hardware simulators.

We release the model source code and all input data required for memory system and application profiling. The released model is ready to be used on high-end Intel platforms, and we would encourage the community to use it, adapt it to other platforms, and share their own evaluations.

Anyone can build a fast CPU.
The trick is to build a fast system.
Seymour Cray, considered the Father of the Supercomputer

8 Conclusions

In state-of-the-art HPC clusters, memory system is one of the major contributor to the deployment and operational costs. Nevertheless, memory system is one of the most critical aspects of the system performance. Challenges posed by the exascale milestone question whether DRAM systems will have the strength to meet the required goals. Further research is therefore needed on applications side, to understand memory requirements and bottlenecks of HPC workloads, and on the hardware side, in order to mitigate the bottlenecks and improve system performance. In this thesis we present studies that analyse memory bandwidth and latency aspects of HPC systems and applications, and use them as a memory system profile to model system performance, power and energy.

8.1 Memory bandwidth and latency aspects in HPC systems evaluation

The first study presents memory bandwidth and latency aspects of HPC systems evaluation and comparison. Evaluating and comparing HPC systems is complex, since the variety of HPC systems in the market is increasing. We present a large body of quantitative results, evaluating three mainstream and five alternative HPC platforms, and highlight four important features in HPC systems evaluation that require higher attention by the community.

We show a platform's performance and energy-efficiency depend significantly (n-fold) on the characteristics of the target application. Haswell platform shows 3.6× better efficiency than ThunderX platform for the compute-intensive HPL, but ThunderX platform has 50% better energy efficiency than Haswell when running memory-bound HPCG. We strongly advocate that any comparison among platforms should include HPCG measurements, to add the boundary for memory-intensive HPC applications, alongside the compute-intensive boundary from HPL.

Second, we detect a significant range in the main memory access latency, with a factor of three difference between the fastest and slowest platforms under study. While KNL with MCDRAM has the highest memory bandwidth, it also has the highest memory access latency, due to complex memory controller and its handling of memory requests. Since memory access latency has a direct performance impact any increment above about 100 ns should be analysed and justified.

Our results show that Byte/FLOP can differ by a factor of up to 21× among the platforms under study. While mainstream platforms show a decreasing tendency, alternative platforms trend upwards in this metric. Byte/FLOP ratio is one of the most important design decisions, and we hope that our results will resurface a discussion on its desired range in HPC.

Finally, we strongly suggest not relying on theoretical FLOPS performance and memory bandwidth, even in a first-order system provisioning, since our measurements demonstrate that sustainable performance on the alternative platforms can deviate more than 70% from theoretical ones. These results will hopefully motivate further development of the compilers and scientific libraries for alternative HPC platforms.

8.2 Memory bandwidth requirements of HPC applications

Next study in the thesis characterizes bandwidth requirements of the HPC applications, and discusses the methodology of presenting the results of quantified performance bottlenecks with respect to representative number of execution processes and measurement granularity. To our knowledge, this is the first study that analyses the importance of a deterministic range for the representative scale of the experiments.

Our results show that HPC application behavior is tightly coupled with the number of application processes, for two main reasons. The first reason is that inter-process communication time increases as the application scales-out, lowering the average utilization of FLOPs and memory bandwidth. The second reason is that scaling-out reduces the portion of the input data handled by each process, which changes the effectiveness of cache memory and the overall process behavior. We argue that it is essential that HPC application suites specify narrow ranges on the number of processes, for the results to be representative of a real-world application use.

Moreover, we demonstrate that plotting only the average values of performance metrics and bottlenecks may be misleading. ALYA-16 and CP2K-128 may seem to be bandwidth insensitive, as their average bandwidths are around 50% and 40% of the sustained bandwidth. However, detailed in-time measurements show that they spent significantly different proportions of the time with severe memory bandwidth utilization: CP2K-128 spends only about 4%, while ALYA-16 spends 55% of its computation time, which presents a serious performance penalty. In this case, ALYA-16, but not CP2K-128, is likely to benefit from higher bandwidth memories. Therefore, we suggest that performance measurements should be defined as the percentage of execution time in which applications use certain portions of maximum sustained values.

8.3 First steps on the performance impact of memory bandwidth and latency

The third study presents our preliminary analysis and expectations of how will 3D-stacked DRAM affect the memory wall for a particular set of HPC applications. 3D-stacking technology now enables DRAM devices that support much higher bandwidths than traditional DIMMs. However, memory wall was defined in terms of latency, not bandwidth, therefore higher bandwidth by itself cannot guarantee better performance. Higher bandwidth may lower average latency, provided that our applications offer sufficient MLP and that CPU architectures can exploit it.

We perform a preliminary analysis on a set of four UEABS applications, together with HPL and STREAM benchmarks. To quantify the performance improvement with regard to memory bandwidth, we increase the memory-bus frequency, and therefore available memory bandwidth, by 25%. Memory-intensive applications that are in the linear or exponential regions, using more than 50% of the sustainable memory bandwidth, show the performance improvement less than 10%. For these applications, the bandwidth upgrade of 3D-stacked memories will reduce contention among concurrent memory requests, reduce memory latency, and improve performance. The performance improvement for the rest low-bandwidth applications under study is negligible. Since 3D-stacked DRAM cannot reduce lead-off memory access latency, memory latency, and thus effective bandwidth and overall performance, will not improve for applications for which lack of MLP limits effective bandwidth.

8.4 Memory system evaluation: Modeling system performance and energy without simulating the CPU

The last study represents a method to quantify the impact of the main memory on application performance and system power and energy consumption, using the analytical model. The model is based on memory system profiling and instrumentation of an application execution on a real platform with a baseline memory system. By running on the real platform, application profiling takes account of the real (not publicly disclosed) prefetcher and out-of-order engine. The model outputs are the predicted performance, power and energy consumption on the target memory.

Memory system profile presents memory access latency dependency with respect to used memory bandwidth. We show that the percentage of write requests impacts memory access latency, especially when stress to the memory system increases over 80% of the sustainable maximum. To the best of our knowledge this is the first study of memory access latency that considers how the latency depends on the fractions of reads and writes.

We evaluated the model on two platforms: Sandy Bridge-EP E5-2670 with four DRAM configurations DDR3-800/1066/1333/1600 on a set of SPEC benchmarks and HPC applications, and Knights Landing Xeon Phi 7230 with DDR4 and high-bandwidth 3D-stacked MCDRAM, using SPEC benchmarks. The model estimations are highly-accurate, typically with only 2% difference from the performance, power and energy consumption measurements on the actual hardware. Also, the model can be used in both scenarios, when the target memory system has increased or decreased available memory bandwidth. Finally, the estimations are accurate even when the baseline and target memory systems have fundamentally disparate bandwidth–latency curves, with different lead-off latency and with an n-fold difference in the available bandwidth.

Comparing the model performance estimations with the ZSim+DRAMSim2 simulations showed that our model is more accurate than hardware simulators. It is also faster by three orders of magnitude, so it can be used to analyze production HPC applications, arbitrarily sized systems, and numerous design options, within a practical length of time.

8.5 Future work

The landscape of HPC architectures evolves rapidly and there will likely be more heterogeneous platforms. As a part of the future work, we plan to evaluate memory bandwidth and latency related aspects of more recent mainstream and emerging architectures. They include Intel Xeon Scalable Processors family, IBM POWER9, Cavium ThunderX2, and accelerators like Nvidia GPUs or PEZY-SCx (used in ZettaScaler series of supercomputers). They are more power efficient than previous generations, use innovative memory technologies and more mature software ecosystem. Hence, it would be interesting to compare their evaluation results with the analysed architectures from this thesis.

In the study of bandwidth requirements of HPC applications, we analysed strong scaling of the applications. Next step is to characterize bandwidth requirements and bottlenecks for weak scaling of production HPC applications. Weak scaling analysis is especially important to anticipate future HPC problems with significantly larger input datasets. This analysis, however, requires HPC application benchmark suites which allow a tunable problem size in a similar way to HPL and HPCG, or at least provide a collection of comparable input sets with varying problem size. Furthermore, bandwidth bottlenecks of HPC applications can be characterized on different mainstream and emerging architectures, mentioned in the HPC systems evaluation study.

Estimation model from this thesis is evaluated on two Intel platforms. Further model evaluations should include different architectures, such are ARM or IBM POWER. As for memory systems, interesting evaluation candidates are RLDRAM and 3D XPoint memory. RLDRAM provides lower device latency and thus should provide significant performance improvement for workloads that put moderate pressure on memory bandwidth. On the other side, 3D XPoint memory has somewhat higher latency than that of DDR DIMMs and it would be valuable to estimate its impact on HPC applications. In the current study, we consider that the power consumption of the rest of the system, apart from the memory, remains constant when the memory system changes. The follow-up study could quantify power and energy impact of the rest of the system, with regard to changes in the memory system. Additionally, this analysis should include power and energy modeling of non DDR memories, such as 3D-stacked memory devices.

8.6 List of publications

In this section, we present a list of research articles that are accepted for publication. We also list the titles of the publications on other topics that are not considered to be the contributions of the thesis.

- Milan Radulovic, Kazi Asifuzzaman, Darko Zivanovic, Nikola Rajovic, Guillaume Colin de Verdière, Dirk Pleiter, Manolis Marazakis, Nikolaos Kallimanis, Paul Carpenter, Petar Radojković, and Eduard Ayguadé, *Mainstream vs. Emerging HPC: Metrics, Trade-offs and Lessons Learned*, in 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), Sept. 2018.

- Milan Radulovic, Kazi Asifuzzaman, Paul Carpenter, Petar Radojković, and Eduard Ayguadé, *HPC Benchmarking: Scaling Right and Looking Beyond the Average*, in Euro-Par 2018: Parallel Processing, pp. 135–146, Aug. 2018.

- Milan Radulovic, Darko Zivanovic, Daniel Ruiz, Bronis R. de Supinski, Sally A. McKee, Petar Radojković, and Eduard Ayguadé, *Another Trip to the Wall: How Much Will Stacked DRAM Benefit HPC?*, in Proceedings of the International Symposium on Memory Systems (MEMSYS), pp. 31–36, Oct. 2015.

8.6.1 Under submission

- Milan Radulovic, Rommel Sánchez Verdejo, Paul Carpenter, Petar Radojković, Bruce Jacob, and Eduard Ayguadé, *Memory System Evaluation: Modeling System Performance and Energy Without Simulating the CPU*, Under submission.

8.6.2 Other publications

- Rommel Sánchez Verdejo, Kazi Asifuzzaman, Milan Radulovic, Petar Radojković, Eduard Ayguadé, and Bruce Jacob, *Main Memory Latency Simulation: The Missing Link*, in Proceedings of the International Symposium on Memory Systems (MEMSYS), Oct. 2018.

- Darko Zivanovic, Milan Pavlovic, Milan Radulovic, Hyunsung Shin, Jongpil Son, Sally A. Mckee, Paul M. Carpenter, Petar Radojković, and Eduard Ayguadé, *Main Memory in HPC: Do We Need More or Could We Live with Less?*, ACM Transactions on Architecture and Code Optimization (TACO), vol. 14, pp. 3:1–3:26, Mar. 2017.

- Darko Zivanovic, Milan Radulovic, Germán Llort, David Zaragoza, Janko Strassburg, Paul M. Carpenter, Petar Radojković, and Eduard Ayguadé, *Large-Memory Nodes for Energy Efficient High-Performance Computing*, in Proceedings of the Second International Symposium on Memory Systems (MEMSYS), pp. 3–9, Oct. 2016.

- Kazi Asifuzzaman, Milan Pavlovic, Milan Radulovic, David Zaragoza, Ohseong Kwon, Kyung-Chang Ryoo, and Petar Radojković, *Performance Impact of a Slower Main Memory: A Case Study of STT-MRAM in HPC*, in Proceedings of the Second International Symposium on Memory Systems (MEMSYS), pp. 40–49, Oct. 2016.

- Milan Pavlovic, Milan Radulovic, Alex Ramirez, and Petar Radojković, *Limpio — LIghtweight MPI instrumentatiOn*, in Proceedings of the IEEE International Conference on Program Comprehension (ICPC), pp. 303–306, May 2015.

Bibliography

[1] Advanced Micro Devices, Inc., "AMD Product Roadmaps." http://ir.amd.com/static-files/a63127c4-569f-4fbe-9fcf-54c24dcfa808, Jan. 2018. [Cited on page iv.]

[2] "TOP500 List." https://www.top500.org/, June 2018. [Cited on pages 1, 26, 49, and 98.]

[3] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, and K. Yelick, "ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems," Sept. 2008. [Cited on pages 1, 2, and 69.]

[4] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, Apr. 1965. [Cited on pages 1 and 3.]

[5] A. Rico, F. Cabarcas, C. Villavieja, M. Pavlovic, A. Vega, Y. Etsion, A. Ramirez, and M. Valero, "On the Simulation of Large-scale Architectures Using Multiple Application Abstraction Levels," *ACM Transactions on Architecture and Code Optimization*, vol. 8, pp. 36:1–36:20, Jan. 2012. [Cited on page 2.]

[6] T. Grass, C. Allande, A. Armejach, A. Rico, E. Ayguad, J. Labarta, M. Valero, M. Casas, and M. Moreto, "MUSA: A Multi-level Simulation Approach for Next-Generation HPC Machines," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 526–537, Nov. 2016. [Cited on page 2.]

[7] A. Sodani, "Race to Exascale: Opportunities and Challenges." Keynote Presentation at the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2011. [Cited on pages 2 and 69.]

[8] R. Stevens, A. White, P. Beckman, R. Bair-ANL, J. Hack, J. Nichols, A. GeistORNL, H. Simon, K. Yelick, J. Shalf-LBNL, S. Ashby, M. Khaleel-PNNL, M. McCoy, M. Seager, B. Gorda-LLNL, J. Morrison, C. Wampler-LANL, J. Peery, S. Dosanjh, J. Ang-SNL, J. Davenport, T. Schlagel, BNL, F. Johnson, and P. Messina, "A Decadal DOE Plan for Providing Exascale Applications and Technologies for DOE Mission Needs." Presentation at Advanced Simulation and Computing Principal Investigators Meeting, Mar. 2010. [Cited on pages 2 and 69.]

[9] B. L. Jacob, “Exascale Begins at the Memory System,” Mar. 2016. Keynote at the DATE Workshop on Emerging Memory Solutions. [Cited on page 2.]

[10] B. L. Jacob, “The memory system: You can’t avoid it, you can’t ignore it, you can’t fake it,” *Synthesis Lectures on Computer Architecture*, vol. 4, no. 1, pp. 1–77, 2009. [Cited on pages 2, 16, 17, 53, 70, and 71.]

[11] W. A. Wulf and S. A. McKee, “Hitting the memory wall: Implications of the obvious,” *ACM SIGARCH Computer Architecture News*, vol. 23, pp. 20–24, Mar. 1995. [Cited on pages 2 (3), 6, 17, 36, 40, 59, and 70.]

[12] R. L. Sites, “It’s the memory, stupid!,” *Microprocessor Report*, vol. 10, pp. 2–3, Oct. 1996. [Cited on page 2.]

[13] J. Shalf, S. Dosanjh, and J. Morrison, “Exascale computing technology challenges,” in *Proceedings of the 9th International Conference on High Performance Computing for Computational Science*, pp. 1–25, June 2011. [Cited on pages 2 and 4.]

[14] J. Dongarra, P. Beckman, T. Moore, P. Aerts, G. Aloisio, J.-C. Andre, D. Barkai, J.-Y. Berthou, T. Boku, B. Braunschweig, F. Cappello, B. Chapman, X. Chi, A. Choudhary, S. Dosanjh, T. Dunning, S. Fiore, A. Geist, B. Gropp, R. Harrison, M. Hereld, M. Heroux, A. Hoisie, K. Hotta, Z. Jin, Y. Ishikawa, F. Johnson, S. Kale, R. Kenway, D. Keyes, B. Kramer, J. Labarta, A. Lichnewsky, T. Lippert, B. Lucas, B. Maccabe, S. Matsuoka, P. Messina, P. Michielse, B. Mohr, M. S. Mueller, W. E. Nagel, H. Nakashima, M. E. Papka, D. Reed, M. Sato, E. Seidel, J. Shalf, D. Skinner, M. Snir, T. Sterling, R. Stevens, F. Streitz, B. Sugar, S. Sumimoto, W. Tang, J. Taylor, R. Thakur, A. Trefethen, M. Valero, A. van der Steen, J. Vetter, P. Williams, R. Wisniewski, and K. Yelick, “The international exascale software project roadmap,” *The International Journal of High Performance Computing Applications*, vol. 25, no. 1, pp. 3–60, 2011. [Cited on pages 2 and 3.]

[15] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, “Raidr: Retention-aware intelligent dram refresh,” in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, pp. 1–12, June 2012. [Cited on page 3.]

[16] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, “Flipping bits in memory without accessing them: An experimental study of dram disturbance errors,” in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, pp. 361–372, June 2014. [Cited on pages 3 and 5.]

[17] D. Zivanovic, M. Radulovic, G. Llort, D. Zaragoza, J. Strassburg, P. M. Carpenter, P. Radojković, and E. Ayguadé, "Large-Memory Nodes for Energy Efficient High-Performance Computing," in *Proceedings of the Second International Symposium on Memory Systems*, pp. 3–9, Oct. 2016. [Cited on pages 3 and 45.]

[18] Micron Technology, Inc., "Intel and Micron Produce Breakthrough Memory Technology." https://files.shareholder.com/downloads/ABEA-45YXOQ/6352638360x0x841530/7852AA28-4E57-4D8F-A180-FA135F0BC406/Micron-Intel_Next_Gen_NVM_Press_Release_FINAL_072815.pdf, 2015. [Cited on pages 3 and 4.]

[19] F. T. Hady, A. Foong, B. Veal, and D. Williams, "Platform storage performance with 3d xpoint technology," *Proceedings of the IEEE*, vol. 105, pp. 1822–1833, Sept. 2017. [Cited on pages 3 and 4.]

[20] R. Patti, "Advances in 3d integrated circuits," in *Proceedings of the International Symposium on Physical Design*, pp. 79–80, Mar. 2011. [Cited on page 3.]

[21] Hybrid Memory Cube Consortium, "Hybrid Memory Cube Specification 2.1." https://hybridmemorycube.org/files/SiteDownloads/HMC-30G-VSR_HMCC_Specification_Rev2.1_20151105.pdf, Oct. 2015. [Cited on pages 3, 6, 19, and 60.]

[22] JEDEC Solid State Technology Association, "High Bandwidth Memory (HBM) DRAM." www.jedec.org/standards-documents/docs/jesd235, Oct. 2013. [Cited on pages 3, 6, 20, and 60.]

[23] R. S. Patti, "New Trends in Advanced 3D Vertical Interconnect Technology." https://indico.cern.ch/event/381514/contributions/901448/attachments/760394/1043064/Tezzaron_Presentation_Infieri_042715.pdf, Apr. 2015. Vth INFIERI Workshop at CERN. [Cited on pages 3 and 6 (2).]

[24] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, "A survey of rollback-recovery protocols in message-passing systems," *ACM Computing Surveys*, vol. 34, pp. 375–408, Sept. 2002. [Cited on page 4.]

[25] S. Tyson, G. Wicker, T. Lowrey, S. Hudgens, and K. Hunt, "Non-volatile, high density, high performance phase-change memory," in *IEEE Aerospace Conference. Proceedings (Cat. No.00TH8484)*, vol. 5, pp. 385–390, Mar. 2000. [Cited on page 4.]

[26] M. Gill, T. Lowrey, and J. Park, "Ovonic unified memory - a high-performance nonvolatile memory technology for stand-alone memory

and embedded applications," in *IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, vol. 1, pp. 202–459, Feb. 2002. [Cited on page 4.]

[27] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, pp. 24–33, June 2009. [Cited on page 4.]

[28] H. . P. Wong, H. Lee, S. Yu, Y. Chen, Y. Wu, P. Chen, B. Lee, F. T. Chen, and M. Tsai, "Metaloxide rram," *Proceedings of the IEEE*, vol. 100, pp. 1951–1970, June 2012. [Cited on page 4.]

[29] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, pp. 80–83, May 2008. [Cited on page 4.]

[30] Y. Xie, "Modeling, architecture, and applications for emerging memory technologies," *IEEE Design & Test*, vol. 28, pp. 44–51, Jan. 2011. [Cited on page 4.]

[31] K. Asifuzzaman, M. Pavlovic, M. Radulovic, D. Zaragoza, O. Kwon, K.-C. Ryoo, and P. Radojković, "Performance Impact of a Slower Main Memory: A Case Study of STT-MRAM in HPC," in *Proceedings of the Second International Symposium on Memory Systems*, pp. 40–49, Oct. 2016. [Cited on page 4.]

[32] K. Asifuzzaman, R. Sánchez Verdejo, and P. Radojković, "Enabling a reliable stt-mram main memory simulation," in *Proceedings of the International Symposium on Memory Systems*, pp. 283–292, Oct. 2017. [Cited on page 4.]

[33] S. Ning, T. O. Iwasaki, D. Viviani, H. Huang, M. Manning, T. Rueckes, and K. Takeuchi, "NRAM: High Performance, Highly Reliable Emerging Memory."
https://www.flashmemorysummit.com/English/Collaterals
/Proceedings/2016/20160811_S301A_Ning.pdf, Aug. 2016. Flash Memory Summit presentation. [Cited on page 4.]

[34] H. Servat, A. J. Pea, G. Llort, E. Mercadal, H. Hoppe, and J. Labarta, "Automating the Application Data Placement in Hybrid Memory Systems," in *IEEE International Conference on Cluster Computing*, pp. 126–136, Sept. 2017. [Cited on page 4.]

[35] L. Alvarez, M. Casas, J. Labarta, E. Ayguade, M. Valero, and M. Moreto, "Runtime-Guided Management of Stacked DRAM Memories in Task Parallel Programs," in *Proceedings of the International Conference on*

Supercomputing, pp. 218–228, June 2018. [Cited on page 4.]

[36] D. Locklear, “Chipkill correct memory architecture,” in *DELL Technology Brief*, Aug. 2000. [Cited on page 5.]

[37] V. Sridharan, N. DeBardeleben, S. Blanchard, K. B. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi, “Memory Errors in Modern Systems: The Good, The Bad, and The Ugly,” in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 297–310, Mar. 2015. [Cited on page 5.]

[38] A. Geist, “Supercomputing’s monster in the closet,” *IEEE Spectrum*, vol. 53, pp. 30–35, Mar. 2016. [Cited on page 5.]

[39] Hewlett Packard Enterprise, “How memory RAS technologies can enhance the uptime of HPE ProLiant servers,” Feb. 2016. Technical white paper 4AA4-3490ENW. [Cited on page 5.]

[40] F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, and M. Snir, “Toward exascale resilience: 2014 update,” *Supercomputing Frontiers and Innovations*, vol. 1, pp. 5–28, June 2014. [Cited on page 5.]

[41] J. D. McCalpin, “SC16 Invited Talk: Memory Bandwidth and System Balance in HPC Systems.” https://sites.utexas.edu/jdm4372/2016/11/22/sc16-invited-talk-memory-bandwidth-and-system-balance-in-hpc-systems, Nov. 2016. [Cited on pages 5 and 42.]

[42] N. Chatterjee, M. Shevgoor, R. Balasubramonian, A. Davis, Z. Fang, R. Illikkal, and R. Iyer, “Leveraging Heterogeneity in DRAM Main Memories to Accelerate Critical Word Access,” in *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 13–24, Dec. 2012. [Cited on page 6.]

[43] K. Asanović, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, “The Landscape of Parallel Computing Research: A View from Berkeley,” Tech. Rep. UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec. 2006. [Cited on page 6 (3).]

[44] R. Murphy, “On the effects of memory latency and bandwidth on supercomputer application performance,” in *Proceedings of the IEEE 10th International Symposium on Workload Characterization*, pp. 35–43, Sept. 2007. [Cited on page 6.]

[45] Office of Science at U.S. Department of Energy, “A Science-Based Case for Large-Scale Simulation (SCaLeS),” tech. rep., July 2003. [Cited on

page 6.]

[46] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pp. 1–12, June 2017. [Cited on page 6.]

[47] K. Lepak, "The next generation AMD enterprise server product architecture." https://www.hotchips.org/wp-content/uploads/hc_archives/hc29/HC29.22-Tuesday-Pub/HC29.22.90-Server-Pub/HC29.22.921-EPYC-Lepak-AMD-v2.pdf, Aug. 2017. Hot Chips 2017. [Cited on page 6.]

[48] L. Spelman, "Cascade Lake advanced performance." https://newsroom.intel.com/wp-content/uploads/sites/11/2018/11/cascade-lake-advanced-performance-press-deck.pdf, Nov. 2018. Data Centric Business Update, Intel Corporation. [Cited on page 6.]

[49] P. McLellan, "DDR5 Is on Our Doorstep." https://community.cadence.com/cadence_blogs_8/b/breakfast-bytes/posts/oip-ddr5, Oct. 2018. [Cited on page 6.]

[50] B. Jacob, S. Ng, and D. Wang, *Memory Systems: Cache, DRAM, Disk.* 2007. [Cited on pages 14, 75, and 95.]

[51] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* Wiley, Apr. 1991. [Cited on pages 17 and 62.]

[52] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach.* 6th ed., Nov. 2017. [Cited on pages 18, 79 (2), 82, and 90 (3).]

[53] J. K. Ousterhout, "Why Aren't Operating Systems Getting Faster As

Fast as Hardware?," *USENIX Summer Conference*, pp. 247–256, June 1990. [Cited on page 17.]

[54] J. D. McCalpin, "Memory Bandwidth and Machine Balance in Current High Performance Computers," *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pp. 19–25, Dec. 1995. [Cited on page 17.]

[55] D. Burger, J. R. Goodman, and A. Kägi, "Memory Bandwidth Limitations of Future Microprocessors," in *Proc. ACM/IEEE International Symposium on Computer Architecture*, pp. 78–89, May 1996. [Cited on page 17.]

[56] P. Rosenfeld, E. Cooper-Balis, T. C. Farrell, D. Resnick, and B. Jacob, "Peering Over the Memory Wall: Design Space and Performance Analysis of the Hybrid Memory Cube," Tech. Rep. UMD-SCA-2012-10-01, University of Maryland, Systems and Computer Architecture Group, Oct. 2012. [Cited on page 19.]

[57] Micron Technology, Inc., "Micron Announces Shift in High-Performance Memory Roadmap Strategy." https://www.micron.com/about/blog/2018/august/micron-announces-shift-in-high-performance-memory-roadmap-strategy, Aug. 2018. [Cited on page 19.]

[58] J. Kim and Y. Kim, "HBM: Memory solution for bandwidth-hungry processors," in *IEEE Hot Chips 26 Symposium (HCS)*, Aug. 2014. [Cited on page 20.]

[59] Barcelona Supercomputing Center, "MareNostrum III System Architecture." https://www.bsc.es/marenostrum/marenostrum/mn3, 2013. [Cited on page 24 (2).]

[60] PRACE Research Infrastructure. www.prace-ri.eu. [Cited on pages 24 and 28.]

[61] Intel Corporation, "Intel® Xeon® Processor E5-2600 Product Family Uncore Performance Monitoring Guide," tech. rep., Mar. 2012. [Cited on pages 24, 51, and 99.]

[62] J. D. McCalpin, "STREAM: Sustainable Memory Bandwidth in High Performance Computers," tech. rep., University of Virginia, 1991-2007. [Cited on pages 24, 27, and 75.]

[63] Barcelona Supercomputing Center, "CTE-KNL User's Guide." https://www.bsc.es/user-support/knl.php, 2017. [Cited on page 25 (2).]

[64] Intel Corporation, "Intel® Xeon Phi™ Processor Performance Moni-

toring Reference Manual - Volume 1: Registers," tech. rep., Mar. 2017. [Cited on page 25.]

[65] A. Sodani, R. Gramunt, J. Corbal, H. S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y. C. Liu, "Knights Landing: Second-Generation Intel Xeon Phi Product," *IEEE Micro*, vol. 36, pp. 34–46, Mar. 2016. [Cited on pages 25 and 98 (2).]

[66] A. Sodani, "Knights landing (KNL): 2nd Generation Intel® Xeon Phi processor," in *2015 IEEE Hot Chips 27 Symposium (HCS)*, pp. 1–24, Aug. 2015. [Cited on page 25.]

[67] A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary, "HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers." https://www.netlib.org/benchmark/hpl/, Sept. 2008. [Cited on pages 26 and 48.]

[68] M. Heroux and J. Dongarra, "Toward a New Metric for Ranking High Performance Computing Systems," Tech. Rep. SAND2013-4744, UTK EECS and Sandia National Labs, June 2013. [Cited on page 26.]

[69] J. Dongarra, M. Heroux, and P. Luszczek, "The HPCG Benchmark." https://www.hpcg-benchmark.org, 2016. [Cited on pages 26 and 48.]

[70] V. Marjanović, J. Gracia, and C. W. Glass, *Performance Modeling of the HPCG Benchmark*, pp. 172–192. Springer International Publishing, 2014. [Cited on page 27.]

[71] P. R. Luszczek, D. H. Bailey, J. J. Dongarra, J. Kepner, R. F. Lucas, R. Rabenseifner, and D. Takahashi, "The HPC Challenge (HPCC) Benchmark Suite," in *Proc. of the ACM/IEEE Conference on Supercomputing*, 2006. [Cited on pages 27, 40, and 48.]

[72] L. McVoy and C. Staelin, "Lmbench: Portable Tools for Performance Analysis," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, 1996. [Cited on page 27.]

[73] Standard Performance Evaluation Corporation, "SPEC CPU 2006." `https://www.spec.org/cpu2006/`. [Cited on page 28.]

[74] Partnership for Advanced Computing in Europe (PRACE), "Unified european applications benchmark suite." www.prace-ri.eu/ueabs/, 2013. [Cited on pages 28, 31, and 50.]

[75] Barcelona Supercomputing Center, *Extrae User guide manual for version 3.1.0*, May 2015. [Cited on page 31.]

[76] M. Pavlovic, M. Radulovic, A. Ramirez, and P. Radojković, "Limpio

— LIghtweight MPI instrumentatiOn," in *Proceedings of the IEEE International Conference on Program Comprehension*, pp. 303–306, May 2015. [Cited on page 32.]

[77] V. Pillet, J. Labarta, T. Cortes, and S. Girona, "Paraver: A tool to visualize and analyze parallel code," in *Transputer and Occam Engineering Series*, Apr. 1995. [Cited on page 33.]

[78] J. Treibig, G. Hager, and G. Wellein, "LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments," in *International Conference on Parallel Processing Workshops*, pp. 207–216, Sept. 2010. [Cited on page 33.]

[79] D. Sanchez and C. Kozyrakis, "ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-core Systems," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, pp. 475–486, June 2013. [Cited on pages 33 and 70.]

[80] R. Sánchez Verdejo and P. Radojković, "Microbenchmarks for Detailed Validation and Tuning of Hardware Simulators," in *2017 International Conference on High Performance Computing Simulation (HPCS)*, pp. 881–883, July 2017. [Cited on pages 33 and 70.]

[81] Intel Corporation, "Intel® 64 and IA-32 Architectures Software Developer's Manual," tech. rep., July 2017. [Cited on pages 33, 44, and 51.]

[82] Intel Corporation, "Intel® 64 and IA-32 Architectures Optimization Reference Manual," tech. rep., Jan. 2016. [Cited on pages 34 and 85.]

[83] C. Maurice, N. Scouarnec, C. Neumann, O. Heen, and A. Francillon, "Reverse Engineering Intel Last-Level Cache Complex Addressing Using Performance Counters," in *Proceedings of the 18th International Symposium on Research in Attacks, Intrusions, and Defenses*, pp. 48–65, Nov. 2015. [Cited on page 34.]

[84] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A Cycle Accurate Memory System Simulator," *IEEE Computer Architecture Letters*, vol. 10, pp. 16–19, Jan. 2011. [Cited on pages 34 and 70 (2).]

[85] TOP500.org, "Supercomputing Centers Have Become Showcases for Competing HPC Technologies." https://www.top500.org/news/supercomputing-centers-have-become-showcases-for-competing-hpc-technologies/, June 2017. [Cited on page 35.]

[86] N. Rajovic, P. M. Carpenter, I. Gelado, N. Puzovic, A. Ramirez, and M. Valero, "Supercomputing with Commodity CPUs: Are Mobile SoCs Ready for HPC?," in *Proc. of the International Conference on High*

Performance Computing, Networking, Storage and Analysis, pp. 40:1–40:12, 2013. [Cited on pages 36, 37, and 44.]

[87] N. Rajovic, A. Rico, F. Mantovani, D. Ruiz, J. O. Vilarrubi, C. Gomez, L. Backes, D. Nieto, H. Servat, X. Martorell, J. Labarta, E. Ayguade, C. Adeniyi-Jones, S. Derradji, H. Gloaguen, P. Lanucara, N. Sanna, J.-F. Mehaut, K. Pouget, B. Videau, E. Boyer, M. Allalen, A. Auweter, D. Brayford, D. Tafani, V. Weinberg, D. Brömmel, R. Halver, J. H. Meinke, R. Beivide, M. Benito, E. Vallejo, M. Valero, and A. Ramirez, "The Mont-blanc Prototype: An Alternative Approach for HPC Systems," in *Proc. of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 38:1–38:12, 2016. [Cited on pages 36 and 44.]

[88] M. A. Laurenzano, A. Tiwari, A. Cauble-Chantrenne, A. Jundt, W. A. Ward, R. Campbell, and L. Carrington, "Characterization and bottleneck analysis of a 64-bit ARMv8 platform," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 36–45, Apr. 2016. [Cited on pages 36, 43, and 45.]

[89] Z. Ou, B. Pang, Y. Deng, J. K. Nurminen, A. Yl-Jski, and P. Hui, "Energy- and Cost-Efficiency Analysis of ARM-Based Clusters," in *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 115–123, May 2012. [Cited on page 37.]

[90] A. Selinger, K. Rupp, and S. Selberherr, "Evaluation of Mobile ARM-based SoCs for High Performance Computing," in *Proc. of the High Performance Computing Symposium*, pp. 21:1–21:7, 2016. [Cited on page 37.]

[91] S. W. Williams, A. Waterman, and D. A. Patterson, "Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures," *EECS Technical Report UCB/EECS-2008-134*, Oct. 2008. [Cited on pages 40, 64, and 66.]

[92] Joint Electron Device Engineering Council. https://www.jedec.org. [Cited on page 41.]

[93] ARM Ltd, "ARM Accelerates Mathematical Computation on 64-bit ARM-based HPC Systems." https://www.arm.com/about/newsroom/arm-accelerates-mathematical-computation-on-64-bit-arm-based-hpc-systems.php, Nov. 2015. [Cited on page 43.]

[94] D. Abdurachmanov, B. Bockelman, P. Elmer, G. Eulisse, R. Knight, and S. Muzaffar, "Heterogeneous High Throughput Scientific Computing

with APM X-Gene and Intel Xeon Phi," *CoRR*, 2014. [Cited on pages 43 and 44.]

[95] I. Z. Reguly, A.-K. Keita, and M. B. Giles, "Benchmarking the IBM Power8 Processor," in *Proc. of the International Conference on Computer Science and Software Engineering*, 2015. [Cited on page 43.]

[96] M. Sayeed, H. Bae, Y. Zheng, B. Armstrong, R. Eigenmann, and F. Saied, "Measuring High-Performance Computing with Real Applications," *Computing in Science & Engineering*, vol. 10, pp. 60–70, July 2008. [Cited on pages 48 and 57.]

[97] W. T. Kramer, "Top500 Versus Sustained Performance: The Top Problems with the Top500 List - and What to Do About Them," in *Proc. of the International Conference on Parallel Architectures and Compilation Techniques*, pp. 223–230, Sept. 2012. [Cited on page 49.]

[98] National Science Foundation. https://www.nsf.gov/pubs/2006/nsf0605/nsf0605.pdf. [Cited on pages 50 and 56.]

[99] National Center for Atmospheric Research. https://www2.cisl.ucar.edu/resources/computational-systems/cisl-high-performance-computing-benchmarks. [Cited on pages 50 and 56.]

[100] National Energy Research Scientific Computing Center. https://www.nersc.gov/users/computational-systems/cori/nersc-8-procurement/trinity-nersc-8-rfp/nersc-8-trinity-benchmarks. [Cited on pages 50 and 56.]

[101] Home page of the EuroBen Benchmark. https://www.euroben.nl. [Cited on pages 50 and 56.]

[102] A. Turner, "UK National HPC Benchmarks," tech. rep., UK National Supercomputing Service ARCHER, 2016. [Cited on pages 50 and 56.]

[103] D. Zivanovic, M. Pavlovic, M. Radulovic, H. Shin, J. Son, S. A. Mckee, P. M. Carpenter, P. Radojković, and E. Ayguadé, "Main Memory in HPC: Do We Need More or Could We Live with Less?," *ACM Transactions on Architecture and Code Optimization*, vol. 14, pp. 3:1–3:26, Mar. 2017. [Cited on page 50.]

[104] M. Radulovic, D. Zivanovic, D. Ruiz, B. R. de Supinski, S. A. McKee, P. Radojković, and E. Ayguadé, "Another Trip to the Wall: How Much Will Stacked DRAM Benefit HPC?," in *Proc. of the International Symposium on Memory Systems*, pp. 31–36, Oct. 2015. [Cited on page 53.]

[105] D. H. Bailey, "Misleading performance claims in parallel computations," in *2009 46th ACM/IEEE Design Automation Conference*, pp. 528–533, July 2009. [Cited on page 56.]

[106] D. H. Bailey, "Twelve ways to fool the masses when giving performance results on parallel computers," *Supercomputing Review*, pp. 54–55, Aug. 1991. [Cited on page 56.]

[107] T. Hoefler and R. Belli, "Scientific Benchmarking of Parallel Computing Systems: Twelve Ways to Tell the Masses when Reporting Performance Results," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 73:1–73:12, Nov. 2015. [Cited on page 57.]

[108] V. Marjanović, J. Gracia, and C. W. Glass, "HPC Benchmarking: Problem Size Matters," in *Proceedings of the 7th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computing Systems*, pp. 1–10, Nov. 2016. [Cited on page 57.]

[109] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, "A Case for Intelligent RAM," *IEEE Micro*, vol. 17, pp. 34–44, Mar. 1997. [Cited on page 60.]

[110] R. Crisp, "Direct RAMbus technology: the new main memory standard," *IEEE Micro*, vol. 17, pp. 18–28, Nov. 1997. [Cited on page 60.]

[111] K. Diefendorff, "Sony's Emotionally Charged Chip," *Microprocessor Report*, vol. 13, pp. 1,6–11, Apr. 1999. [Cited on page 60.]

[112] A. Mandapati, "2001: A Graphics Odyssey," *Microprocessor Report*, vol. 16, pp. 7–10, Jan. 2002. [Cited on page 60.]

[113] R. Kalla, B. Sinharoy, W. Starke, and M. Floyd, "Power7: IBM's Next-Generation Server Processor," *IEEE Micro*, vol. 30, pp. 7–15, Mar. 2010. [Cited on page 60.]

[114] P. Hammarlund, "4$^{\text{th}}$ Generation Intel® Core™Processor, codenamed Haswell," *Hot Chips 25*, Aug. 2013. [Cited on page 60.]

[115] Intel Corporation, "Intel® Pentium® 4 Processor and Intel® E7205 Chipset Design Guide," Dec. 2002. [Cited on page 60.]

[116] Intel Corporation, "Intel® 875P Chipset Datasheet," Feb. 2004. [Cited on page 60.]

[117] JEDEC Solid State Technology Association, "Double Data Rate (DDR) SDRAM Standard." www.jedec.org/standards-documents/docs/jesd-79f, Feb. 2008. [Cited on page 60.]

[118] A. Glew, “MLP yes! ILP no!,” *International Conference on Architectural Support for Programming Languages and Operating Systems, Wild and Crazy Ideas Session*, Oct. 1998. [Cited on pages 60 and 84.]

[119] Arira Design, “Hybrid Memory Cube Evaluation & Development Board.” https://www.ariradesign.com/hmc-board, 2013. [Cited on pages 61 and 75.]

[120] Y. Durand, P. Carpenter, S. Adami, A. Bilas, D. Dutoit, A. Farcy, G. Gaydadjiev, J. Goodacre, M. Katevenis, M. Marazakis, E. Matus, I. Mavroidis, and J. Thomson, “EUROSERVER: Energy Efficient Node for European Micro-servers,” in *Proc. Euromicro Conference on Digital Systems Design*, pp. 206–213, Aug. 2014. [Cited on page 61.]

[121] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. S. Duff, “A Set of Level 3 Basic Linear Algebra Subprograms,” *ACM Transactions on Mathematical Software*, vol. 16, pp. 1–17, Mar. 1990. [Cited on page 66.]

[122] Y. Kim, W. Yang, and O. Mutlu, “Ramulator: A Fast and Extensible DRAM Simulator,” *IEEE Computer Architecture Letters*, vol. 15, pp. 45–49, Jan. 2016. [Cited on page 70.]

[123] Micron Technology, Inc., “Calculating Memory System Power for DDR3,” Tech. Rep. TN-41-01, Aug. 2007. [Cited on pages 74 and 91.]

[124] R. Sánchez Verdejo, K. Asifuzzaman, M. Radulovic, P. Radojković, E. Ayguadé, and B. Jacob, “Main Memory Latency Simulation: The Missing Link,” in *Proceedings of the International Symposium on Memory Systems*, Oct. 2018. [Cited on page 75.]

[125] R. Clapp, M. Dimitrov, K. Kumar, V. Viswanathan, and T. Willhalm, “Quantifying the Performance Impact of Memory Latency and Bandwidth for Big Data Workloads,” in *IEEE International Symposium on Workload Characterization*, pp. 213–224, Oct. 2015. [Cited on pages 76, 77, 79, 90 (2), and 113 (2).]

[126] P. G. Emma, “Understanding some simple processor-performance limits,” *IBM Journal of Research and Development*, vol. 41, pp. 215–232, May 1997. [Cited on page 79.]

[127] T. S. Karkhanis and J. E. Smith, “A First-Order Superscalar Processor Model,” in *Proceedings of the Annual International Symposium on Computer Architecture*, pp. 338–349, June 2004. [Cited on pages 79, 82 (2), 83, 84 (2), 85, 90 (3), 111, 112, and 113 (2).]

[128] Y. Chou, B. Fahs, and S. Abraham, “Microarchitecture Optimizations for Exploiting Memory-Level Parallelism,” in *Proceedings of the 31st*

Annual International Symposium on Computer Architecture, pp. 76–87, June 2004. [Cited on pages 79, 84 (2), and 90 (3).]

[129] S. Eyerman, L. Eeckhout, T. Karkhanis, and J. E. Smith, "A Performance Counter Architecture for Computing Accurate CPI Components," in *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 175–184, Dec. 2006. [Cited on page 84.]

[130] D. Kroft, "Lockup-free instruction fetch/prefetch cache organization," in *Proceedings of the Annual Symposium on Computer Architecture*, pp. 81–87, May 1981. [Cited on page 85.]

[131] J. Jeffers, J. Reinders, and A. Sodani, *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition.* 2nd ed., 2016. [Cited on page 85.]

[132] S. Eyerman, L. Eeckhout, T. Karkhanis, and J. E. Smith, "A Mechanistic Performance Model for Superscalar Out-of-order Processors," *ACM Trans. Comput. Syst.*, vol. 27, pp. 3:1–3:37, May 2009. [Cited on pages 90 (2), 112 (3), and 113 (2).]

[133] S. V. den Steen, S. Eyerman, S. D. Pestel, M. Mechri, T. E. Carlson, D. Black-Schaffer, E. Hagersten, and L. Eeckhout, "Analytical processor performance and power modeling using micro-architecture independent characteristics," *IEEE Transactions on Computers*, vol. 65, pp. 3537–3551, Dec 2016. [Cited on pages 90 (2), 112, and 113 (2).]

[134] X. Feng, R. Ge, and K. W. Cameron, "Power and energy profiling of scientific applications on distributed systems," in *IEEE International Parallel and Distributed Processing Symposium*, Apr. 2005. [Cited on pages 91 and 103.]

[135] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, "MemScale: Active Low-power Modes for Main Memory," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 225–238, Mar. 2011. [Cited on pages 91, 93, 113, and 114 (2).]

[136] Micron Technology, Inc., "MT36JSF1G72PZ-1G6M1, 8GB (x72, ECC, DR) 240-Pin DDR3 RDIMM." https://www.micron.com/˜/media/documents/products/data-sheet/modules/parity_rdimm/jsf36c1gx72pz.pdf, Apr. 2013. [Cited on page 91.]

[137] Intel Corporation, "Intel® Xeon® Processor E5-1600/E5-2600/E5-4600 Product Families Datasheet - Volume One," Tech. Rep. 326508, May

2012. [Cited on page 95.]

[138] Intel Corporation, "Intel® Xeon Phi™ Processor Performance Monitoring Reference Manual - Volume 2: Events," tech. rep., Mar. 2017. [Cited on page 99.]

[139] M. Radulovic, K. Asifuzzaman, P. Carpenter, P. Radojković, and E. Ayguadé, "HPC Benchmarking: Scaling Right and Looking Beyond the Average," in *Euro-Par 2018: Parallel Processing*, pp. 135–146, Aug. 2018. [Cited on page 99.]

[140] Y. T. . M. Corporation, "Wt230 digital power meter." https://cdn.tmi.yokogawa.com/IM760401-01E.pdf. [Cited on page 100.]

[141] D. Genbrugge, S. Eyerman, and L. Eeckhout, "Interval simulation: Raising the level of abstraction in architectural simulation," in *The Sixteenth International Symposium on High-Performance Computer Architecture*, pp. 307–318, Jan. 2010. [Cited on pages 112 and 113.]

[142] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 469–480, Dec. 2009. [Cited on page 112.]

[143] N. Gulur, M. Mehendale, R. Manikantan, and R. Govindarajan, "ANATOMY: An Analytical Model of Memory System Performance," in *The 2014 ACM International Conference on Measurement and Modeling of Computer Systems*, pp. 505–517, June 2014. [Cited on page 112.]

[144] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt, "The m5 simulator: Modeling networked systems," *IEEE Micro*, vol. 26, pp. 52–60, July 2006. [Cited on page 112.]

[145] Intel Corporation, "Intel® Memory Latency Checker." https://software.intel.com/en-us/articles/intelr-memory-latency-checker, Nov. 2013. [Cited on page 114.]