Memory Systems for High-Performance Computing: The Capacity and Reliability Implications

Author:
Darko Živanović

Thesis director:
Petar Radojković

Tutor:
Eduard Ayguadé

This dissertation is submitted for the degree of
Doctor of Philosophy
Departament of Computer Architecture
Universitat Politècnica de Catalunya

Barcelona, 2018

Acknowledgements

First of all, I would like to thank my advisor Petar Radojković for his guidance and help. It has been an honor to be his PhD student. Special gratitude for believing in me from the beginning, for his patience and time, and for teaching me to look at the problems from different perspectives. I would also like to thank professor Eduard Ayguadé for his help and constructive comments on each topic covered in this thesis. Special thanks to Paul Carpenter, for his help, especially with mathematics and statistics, and for helping and teaching me to present results in a structural and concise manner. I believe that, over the years, I learned so much from Petar, Edu and Paul, and it was a great pleasure to be part of their team.

I would like to thank all my colleagues from Memory Systems group at Barcelona Supercomputing Center and all guys from my office for sharing both good and bad moments together. I would also like to thank Hyunsung Shin from Samsung in Korea, for his technical help and for all the good times during his stays in Barcelona.

Thanks to pre-lectura committee and their constructive comments; the quality of my PhD thesis manuscript was significantly improved.

Also, I would like to thank my roommates and friends: Zoki, Rajo, LošMi, Coa, Marković, Boki, Brane, Mrdži, Ratko, Tanasije, Ugi, DjoMla, Pile, Milan, Pavle, Ana, Radule and Milica for sharing all the great times in Barcelona. Special thanks to Klimek family, for making Barcelona feel like my second home.

Thanks to my mother and father, my brother and his wife for their unconditional support.

Most of all, I would like thank my Jovana, for her patience, understanding and endless support. Also, I thank my super dog Tofi, for making me smile even in the hardest of times. And thank you Jan, my son, for inspiring me while writing the last paragraphs of this thesis.

Author

This work was supported by the Collaboration Agreement between Samsung Electronics Co., Ltd. and Barcelona Supercomputing Center, Spanish Government through Severo Ochoa programme (SEV-2015-0493), by the Spanish Ministry of Science and Technology through TIN2015-65316-P project and by the Generalitat de Catalunya (contracts 2014-SGR-1051 and 2014-SGR-1272). This work has also received funding from the European Union's Horizon 2020 research and innovation programme under ExaNoDe project (grant agreement № 671578). Author was supported by the Severo Ochoa grant (SVP-2014-068501) of the Ministry of Economy and Competitiveness of Spain.

Abstract

Memory systems are significant contributors to the overall power requirements, energy consumption, and the operational cost of large high-performance computing systems (HPC). Limitations of main memory systems in terms of latency, bandwidth and capacity, can significantly affect the performance of HPC applications, and can have strong negative impact on system scalability. In addition, errors in the main memory system can have a strong impact on the reliability, accessibility and serviceability of large-scale clusters. This thesis studies capacity and reliability issues in modern memory systems for high-performance computing.

The choice of main memory capacity is an important aspect of high-performance computing memory system design. This choice becomes increasingly important now that 3D-stacked memories are entering the market. Compared with conventional DIMMs, 3D memory chiplets provide better performance and energy efficiency but lower memory capacities. Therefore the adoption of 3D-stacked memories in the HPC domain depends on whether we can find use cases that require much less memory than is available now. We analyze memory capacity requirements of important HPC benchmarks and applications. The study identifies the HPC applications and use cases with memory footprints that could be provided by 3D-stacked memory chiplets, making a first step towards the adoption of this novel technology in the HPC domain. For HPC domains where large memory capacities are required, we demonstrate that scaling-in of HPC applications reduces energy consumption and the running time of a batch of jobs. We also show that upgrading the per-node memory capacity enables greater degree of scaling-in and additional energy savings.

Memory system is one of the main causes of hardware failures. In each generation, the DRAM chip density and the amount of the memory in systems increase, while the DRAM technology process is constantly shrinking. Therefore, we could expect that the DRAM failures could have a serious impact on the future-systems reliability. This thesis studies DRAM errors observed on a production HPC system during a period of two years. We clearly distinguish between two different approaches for the DRAM error analysis: categorical analysis and the analysis of error rates. The first approach compares the errors at the DIMM level and partitions the DIMMs into various categories, e.g. based on whether they did or did not experience an error. The second approach is to analyze the error rates, i.e., to present the total number of errors relative to other statistics, typically the number of MB-hours or the duration of the observation period. We show that although DRAM error analysis may be performed with both approaches, they are not interchangeable and can lead to completely different conclusions. We further demonstrate the importance of providing statistical significance and presenting results that have practical value and real-life use. We show that various widely-accepted approaches for DRAM error analysis may provide data that appear to support an interesting conclusion, but are not statistically significant, meaning that they could merely be the result of chance. We hope the study of methods for DRAM error analysis presented in this thesis will become a standard for any future analysis of DRAM errors in the field.

Contents

Front matter **i**
Acknowledgements . . . iii
Abstract . . . v
Contents . . . x
List of figures . . . xiv
List of tables . . . xv

1 Introduction **1**
1.1 Challenges in current memory systems . . . 2
1.1.1 Memory performance . . . 2
1.1.2 Architectural exploration . . . 4
1.1.3 Memory capacity . . . 4
1.1.4 Memory reliability . . . 5
1.2 Thesis contributions . . . 7
1.2.1 Memory capacity requirements of HPC applications . . . 7
1.2.2 Large-memory nodes for energy-efficient HPC . . . 7
1.2.3 DRAM errors in the field . . . 8
1.3 Thesis structure . . . 9

2 Background **11**
2.1 DRAM memory system overview . . . 11
2.1.1 DRAM organization . . . 12
2.1.2 Error-Correcting Codes . . . 14
2.2 Emerging memory technologies . . . 17
2.2.1 3D-stacked DRAM . . . 17
2.2.2 Non-volatile memories . . . 20
2.2.3 Implications on memory system capacity and reliability . . . 21

3 Experimental methodology **23**
3.1 Hardware platform 23
3.2 Applications 24
3.2.1 HPC benchmarks 25
3.2.2 Production HPC applications 25
3.3 Tools 28
3.3.1 Extrae 28
3.3.2 Paraver 29
3.3.3 Limpio 30
3.3.4 Error logging daemons 30

4 Memory capacity requirements of HPC applications **31**
4.1 Introduction 32
4.2 Experimental setup 34
4.2.1 Hardware platform 35
4.2.2 Applications 35
4.2.3 Methodology 35
4.3 High-Performance Linpack 36
4.3.1 Measured memory requirements 36
4.3.2 Analysis 37
4.3.3 Mathematical analysis 39
4.4 High-Performance Conjugate Gradients 42
4.4.1 Measured memory requirements 42
4.4.2 Analysis 44
4.5 Production HPC applications 45
4.5.1 Memory footprint vs. Number of processes 45
4.5.2 Selecting the number of processes 48
4.5.3 Memory requirements of production
HPC applications 52
4.5.4 Towards weak scaling analysis 55
4.6 Implications 57
4.6.1 3D-stacked DRAM in HPC memory systems:
Opportunities and challenges 57
4.7 Related work 60
4.8 Summary 62

5 Large-memory nodes
for energy efficient HPC **63**
5.1 Introduction 65
5.2 Methodology 66
5.2.1 Hardware platform 66

5.2.2 Applications 66
5.2.3 Power and energy measurements 67
5.3 Scaling-in on standard nodes 68
5.3.1 Execution time *vs.* node-hours *vs.* energy 68
5.3.2 Understanding energy *vs.* node-hours 71
5.3.3 Implications and impact 71
5.4 Large-memory nodes for energy efficiency 72
5.4.1 Large-memory nodes for capacity computing 74
5.4.2 Large-memory node cost-benefit analysis 75
5.5 Related work 76
5.6 Second thoughts on scalability 78
5.7 Summary 78

6 DRAM errors in the field 81
6.1 Introduction 81
6.2 Background 84
6.2.1 Taxonomy: Are correctable DRAM errors failures? . . 84
6.2.2 Statistical significance 85
6.3 Environment description 86
6.3.1 MareNostrum 3 86
6.3.2 Data collection 87
6.4 Categorical analysis 88
6.4.1 Uncorrected errors 88
6.4.2 Corrected errors 90
6.4.3 Corrected vs. uncorrected errors 91
6.4.4 Errors vs. Faults 92
6.4.5 Summary 92
6.5 Error rate analysis 93
6.5.1 Uncorrected error characterization 93
6.5.2 Corrected error characterization 97
6.5.3 Statistical significance 100
6.5.4 Corrected vs. uncorrected errors 101
6.5.5 Error rates vs. Categorical analysis 102
6.5.6 Summary 104
6.6 Related Work 105
6.6.1 Uncorrectable DRAM errors and whole system resiliency 105
6.6.2 Correctable DRAM errors 106
6.7 Summary 108

7 Conclusions **111**
7.1 Thesis contributions . . . 111
7.1.1 Memory capacity requirements of HPC applications . . . 111
7.1.2 Large-memory nodes for high-performance computing . . . 112
7.1.3 DRAM errors in the field . . . 112
7.2 Future work . . . 113
7.2.1 Memory capacity study . . . 113
7.2.2 Memory reliability study . . . 114
7.3 Publications . . . 115
7.3.1 Conferences . . . 115
7.3.2 Journals . . . 115
7.3.3 Posters . . . 116
7.3.4 Under submission . . . 116
7.3.5 Other publications . . . 116

Back matter **117**
Bibliography . . . 117
Glossary . . . 128

List of Figures

1.1 Memory system research directions. This thesis focuses on memory capacity and memory resiliency. 3

2.1 Main memory system comprises of memory controller, memory bus and a number of DRAM chips organized into DIMMs. 12
2.2 DRAM cell consists of one access-transistor and one capacitor. The cell is accessed through the corresponding wordline and bitline. 12
2.3 Large number of DRAM cells is grouped together to form DIMM array structure. 13
2.4 One rank of DRAM comprises of several banks, each spreading over several DRAM chips that are accessed in parallel. 14
2.5 Parity bit generation. Parity bit is generated from a recursive application of the exclusive-or function to the bits in the data bit vector. 15
2.6 Humming distance illustration. 16
2.7 Chipkill illustration. 17
2.8 HMC internal structure. 19
2.9 HBM internal structure. 19

3.1 MareNostrum 3 compute node comprises of two eight-core Sandy Bridge processors. Each processor connects to main memory through four memory channels. 24

4.1 Per-core memory capacity of HPC systems leading the TOP500 list (June 2015). Systems with exactly 2, 3, and 4 GB of memory per core are included in bars [2,3) GB, [3,4) GB, and [4,5) GB, respectively. Today's HPC systems are dominated by x86 architectures coupled with 2–3 GB of main memory per core. The next most prevalent systems are Blue Gene platforms based on PowerPC cores with 1 GB of memory per core, included in the *[1,2) GB* bar. 32

4.2 HPL performance also depends on the available memory capacity. When increasing per-core memory, HPL performance first increases and then reaches the saturation point — the sustained floating-point rate (GFLOP/s) of the system. As we increase the number of processes, the saturation point moves towards larger per-core memory capacities. 37

4.3 Per-core memory needed to get 90%, 95% and 99% of the ideal HPL performance, for different sizes of HPC systems. Three systems, with 250 MB/core (year 2003), 2 GB/core (year 2016) and 16.1 GB/core (future potential system) are compared with our estimation curves. Moving from 95% to 99% of ideal HPL performance requires a huge step in the amount of per-core memory (axes are plotted on logarithmic scales). 38

4.4 HPCG performance also depends on the available memory capacity. Performance increases until it reaches the saturation point, where it is constrained by the sustained memory bandwidth. The saturation point remains constant across a wide range of HPCG processes, at around 512 MB of main memory per process. 43

4.5 Per-process memory footprints shrink as the number of processes increases. 47

4.6 Trade-offs between normalized execution time and experiment cost (CPU-hours) for applications with good and limited scalability. 50

4.7 The representative number of application processes is determined by application scalability and the targeted HPC category. 51

4.8 Memory footprints of production HPC applications depend on application scalability and the targeted HPC category. Only the applications with limited scalability that target capacity computing require gigabytes of main memory per process. . . 53

4.9 As the number of processes increase, memory footprints of worker processes decrease as expected, but memory footprint of the master process increases: ALYA, Test Case A. 54

4.10 Increasing the input dataset changes memory footprint and scalability of NAMD application. We detect the same trend for all UEABS applications. 56

5.1 Evolution of per-core memory capacity of HPC systems leading the TOP500 list during the period from 2014 until 2016. Each year, number of HPC systems that comprise more than 4GB of main memory per core is increasing. 64

5.2 ALYA, 1–64 nodes: Increasing the number of nodes increases both energy and node-hours, with strong correlation. 69
5.4 ALYA, 1–64 nodes: Scaling-out decreases power per node, since nodes spend more time in communication. 70
5.5 Scaling-in increases memory requirements and energy efficiency of HPC applications. Node-hours and energy are shown relative to the experiment on the minimum number of standard nodes. 73
5.6 Summary of energy savings enabled by using large-memory nodes. $[a \rightarrow b]$ refers to a shift from a standard to b large-memory nodes. 74
5.7 System scaling can be horizontal (*scale-in* or *-out*) and vertical (*scale-up* or *-down*). Traditionally, HPC community is focused mainly on *scale-out*, referring to it simply as *scaling*. Our study analyzes *scale-in* on standard nodes, and a combined *scale-up* and *scale-in* approach on large-memory nodes. 77

6.1 Percentage of DIMMs with uncorrected errors: Manufacturer and technology comparison. 89
6.2 Percentage of DIMMs with corrected errors: Manufacturer and technology comparison. 91
6.3 Uncorrected errors per day. The bars show the sum of the UEs for all the DIMMs of a given manufacturer. 95
6.4 Average uncorrected errors per MB-hour: each point is the running average number of uncorrected errors per MB-hour observed up to that point. The running average is an *impulse and down-ramp* function. Depending on the moment of observation, we reach different conclusions about the ranking of the different DIMM manufacturers. During the observation period, *Manufacturer A* and *C* switched order twice. 96
6.5 Corrected errors per day. On most days we detect zero or close to zero correctable errors; but on a few days there are very large numbers of correctable errors, up to about 110,000 errors. 98
6.6 Average corrected errors per MB-hour: each point is the running average observed up to that point. The running average is an *impulse and down-ramp* function. Depending on the moment of observation, we reach different conclusions about the ranking of the DIMM manufacturers. During the observation period, *Manufacturer A* and *B* switched order eight times. . . 99

6.7 Corrected errors, faults and uncorrected errors per billion MB hours. The corrected error and fault rates have the same trend, but the uncorrected error rates exhibit a **different trend**. . . 102
6.8 Mean time between corrected errors, faults and uncorrected errors. The corrected error and fault rates have the same trend, but the uncorrected error rates exhibit a **different trend**.103
6.9 Technology comparison ambiguity, uncorrected errors: Errors per MB-hour vs. Percentage of DIMMs with errors. 104
6.10 Technology comparison ambiguity, corrected errors: Errors per MB-hour vs. Percentage of DIMMs with errors. 104

List of Tables

3.1 Production HPC applications analyzed in the thesis 29

4.1 Scientific HPC applications used in the study 46

5.1 UEABS applications used in the study. 67
5.2 ALYA, 1 vs. 64 jobs: The scale-in approach is 27× slower for a single job, but 2.37× *faster* for 64 jobs. In both experiments scaling-in reduces node-hours by 2.37× and energy consumption by 2.07×. 70
5.3 CP2K, 1 vs. 8 jobs: Execution on large-memory nodes is 6.5× slower for one job, but 1.23× *faster* for eight jobs. For both job sizes, node-hours and energy reduce when using large-memory nodes. 75
5.4 Payback from large-memory nodes over five-year system lifetime [%]. 76

6.1 Contingency table: Dependency between the number of DIMMs that experienced uncorrectable error (UE) and the DIMM manufacturer. The statistical test indicates no dependency, p-value = 0.24, so we cannot claim any statistically significant difference in the probability that DIMMs from *Manufacturers A*, *B* and *C* will experience uncorrectable errors. 89
6.2 Contingency table: Dependency between correctable (CE) and uncorrectable (UE) errors (All manufacturers). The statistical test indicates strong dependency, p-value $< 2.2 \times 10^{-16}$; i.e. we can claim that DIMMs that experienced CEs have higher probability of also experiencing UEs. 92

CHAPTER 1

Introduction

High-performance computing (HPC) is indispensable for a diverse range of scientific, engineering, and business domains. Over the years, it has become a crucial pillar of science and a forefront of scientific discovery and commercial innovation. HPC refers to the use of *supercomputers* to solve complex computational problems through computer modeling, simulation, and data analysis. A supercomputer or an HPC system is essentially a network of compute nodes, each containing multiple processing cores as well as its own local memory to execute a wide range of software algorithms. Current generation supercomputing systems are comprised of tens or hundreds of thousands of compute nodes based on modern multi-core architectures and petabytes of memory to satisfy the demands of the HPC applications. However, to reach the next-generation HPC computing system able to perform 10^{18} floating point operations per second, i.e. the *exascale system*, HPC is undergoing a major change in terms of energy and power, memory and storage, concurrency, and resilience [49].

Memory systems are significant contributors to the overall power requirements, energy consumption, and the operational cost of large HPC clusters [49, 102, 109]. The limitations of main memory system in terms of latency, bandwidth and capacity, can significantly affect the performance of HPC applications, and could have strong negative impact on system scalability [31, 71]. Dynamic Random Access Memory (DRAM) has been the dominant main memory technology of most computing systems for decades. Its relatively low cost per bit of stored data and high access speed have satisfied the most typical requirements of computing systems. However, the rate of improvement in microprocessor speed exceeded the rate of improvement in DRAM memory speed, so for most of the applications in current computer systems, memory system limits the performance [67, 101]. In the future, we can expect even higher pressure on HPC memory systems. Future HPC processors will be based on many-core architectures which will result in significant reduction in per-core memory capacity and bandwidth relative to current architectures [26].

Consequently, it is questionable whether conventionally-used memory architectures based on mature DRAM technology and DIMM organization will meet the needs of next-generation HPC systems. Thus, significant effort is invested in research and development of novel memory systems. On the one hand, memory manufacturers propose hybrid memory cube (HMC) [37] and high-bandwidth memory (HBM) [44]. These solutions use 2.5D and 3D stacking to bring DRAM main memory closer to the processing units, which significantly improves bandwidth over conventional DIMM-based architectures. On the other hand, expected end of scaling of DRAM technology motivates development of non-volatile memories (NVMs) and research on heterogeneous memory systems that combine DRAM and NVMs [84, 110, 120]. Recent trends in the research, development, and manufacturing of NVM technologies provide optimism that these technologies can be used to provide adequate amounts of main memory capacities for future systems.

1.1 Challenges in current memory systems

Research on memory systems for HPC can take several directions. Some of the most important research directions are summarized in Figure 1.1. However, in practice, these research directions are not completely orthogonal and independent, as Figure 1.1 might suggest. In this section we briefly comment on each of the four directions: memory performance, architectural exploration, memory capacity and memory reliability. This thesis focuses on the capacity and reliability issues in memory systems for HPC. In the first part of the thesis we analyze the memory capacity requirements of HPC applications, and in the second part, we present a field study of DRAM errors on the MareNostrum 3 supercomputer.

1.1.1 Memory performance

Since the first vector supercomputers, large scale applications have traditionally been floating point oriented codes, and HPC architectures have evolved to meet the needs of those applications. However, in today's supercomputers, the application performance is dominated by the memory system. Analyzing memory system performance implies studying the bottlenecks in the execution of HPC applications such as memory bandwidth and memory latency.

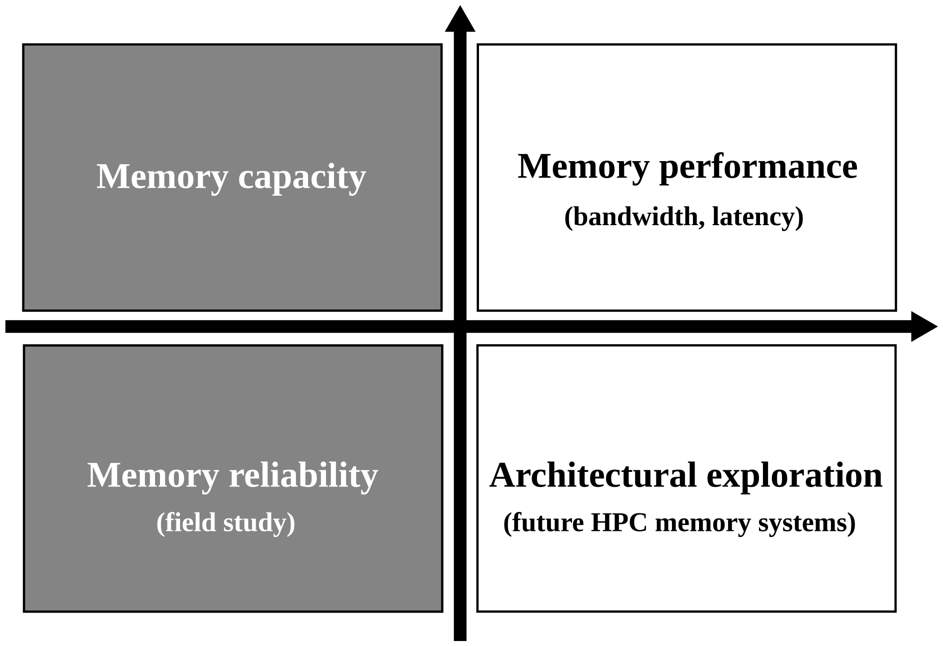


Figure 1.1: Memory system research directions. This thesis focuses on memory capacity and memory resiliency.

Dynamic Random Access Memory (DRAM) technology has been the dominant main memory in most computing systems since 1970s until today, primarily due to its low cost-per-bit [43]. As DRAM process technology scaled, memory manufacturers were able to integrate more DRAM cells into the die area. As a result, each successive DRAM generation had more memory capacity at relatively low cost. However, expected end of DRAM technology scaling prevents further improvements in memory capacity and cost-per-bit.

Contrary to capacity, bandwidth and cost improvements, DRAM latency remained almost constant. This, together with constant improvements in processor speed, created a performance gap between processor and main memory, known as the memory wall [119]. Memory wall refers to a phenomenon of processors being able to execute code faster than memory can feed them with instructions and data. From the perspective of the processor, an access to DRAM takes hundreds of cycles. This is the time during which a modern processor is likely to be stalled, waiting for data from DRAM memory. DRAM chips that provide lower latency exist, such as Reduced Latency DRAM (RLDRAM), but their higher cost of production means that they are mostly used in specialized applications.

The introduction of chip multiprocessors (CMP) managed to hide DRAM latency up to a point. But, increasing the number of processing cores that compete for memory requests over limited number of memory channels created a new problem — memory bandwidth bottleneck [65]. Lately, novel memory solutions emerged that bring main memory closer to processor and increase memory bandwidth significantly [16, 30, 114]. However, solutions such as processing-in-memory (PIM), high-bandwidth memory, hybrid memory cube, will first need to be tested on many platforms and domains before being adopted as mainstream memory solutions in HPC.

1.1.2 Architectural exploration

Due to the limitations of mature DRAM technology significant effort is invested in research and development of novel memory systems. Architectural exploration refers to the analysis of the suitability of various memory solutions for high-performance computing domain, such as 3D stacking DRAM and non-volatile memory solutions. It also includes the DRAM architectural proposals that can improve the performance of HPC workloads.

High-Bandwidth Memory (HBM) and Hybrid Memory Cube (HMC) use 2.5D and 3D stacking to bring DRAM main memory closer to the processing units, which significantly improves access latency and bandwidth over conventional DIMM-based architectures. On the down side, 3D-stacked DRAMs will likely reduce the main memory capacity, at least in early generations, due to significantly higher cost per bit than conventional DIMMs. Many studies analyze their suitability to replace DIMM based DRAM memory, and propose different architectural solutions that can potentially improve the performance [54, 59, 69].

The expected end of scaling of DRAM technology motivates development of non-volatile memories (NVMs) and research on heterogeneous systems that combine DRAM and NVMs. NVM solutions that attracted a lot of interest in the academy and industry include Phase Change Memory (PCM), Resistive RAM (RRAM), and Spin-Transfer Torque Magnetoresistive (STT-MRAM) [6, 51, 53, 117].

1.1.3 Memory capacity

In modern HPC systems, DRAM main memory accounts for a significant portion of node's overall power consumption [31]. This consumption is determined by several factors including memory capacity and configuration. Several DARPA and DOE reports state that the future Exascale machine will have very limited power budget, and that memory energy-efficiency is of utmost importance [49, 109]. Therefore, any attempt to limit the power consumption of a future system will necessarily constrain DRAM capacity. On the other hand, projections expect that future computing workloads will often process unstructured datasets with sizes that are orders of magnitude larger than the current ones [26, 93]. As a consequence, the projected limitations of main memory capacity of future HPC systems could have several important design implications. Limited main memory capacity would leave the system architecture out of balance, and would impact the system scalability and parallel efficiency. Namely, a smaller memory per node reduces the amount of computation a node can execute without the communication with other nodes, increases the frequency of communication, and reduces the sizes communicated messages. Therefore, smaller memory capacities would be efficient for only the most computationally intense workloads.

In state-of-the-art HPC clusters, provisioning of memory system is one of the most important design issues. From the perspective of a scientist executing applications on HPC cluster, the ratio of memory capacity to processor computing capability is critical in determining the size of the problem that can be solved. The processor dictates how much computing can be done, and the memory dictates the size of the problem that can be handled. Therefore, sizing of memory systems is one of the most important decisions in the HPC system design — over-provisioning of main memory capacity leads to a significant increase in the total cost, while under-provisioning limits the problem size that can be simulated, and hinders the overall usability of the system.

Proper provisioning of memory systems for HPC clusters and research on next-generation memory systems require profound understanding of the memory behavior of HPC applications. Although HPC system memory provisioning is a very important task, insufficient effort has been dedicated to understand HPC application memory capacity requirements. This thesis bridges this gap and focuses on the analysis of memory capacity requirements of important HPC benchmarks and applications.

1.1.4 Memory reliability

Modern supercomputers, because of their scale and complexity, run for only a few days before rebooting one of the nodes. Exascale systems will be even more complex and comprise more processors and memory modules. Growing number of components, smaller transistor sizes and the need for more energy-efficient machines will likely increase the fault rate. So, the major challenge in the resilience of Exascale machines is that faults will be regular rather than an exceptional event. In current HPC systems, every failure kills the application running on the affected nodes, and the application has to be restarted from the beginning or from their last checkpoint [97]. However, the checkpoint approach will not work indefinitely, because as computers get larger, the checkpointing overhead increases. Eventually, this interval will become longer than the typical period before the next failure. In order to cope with the increased fault rates, we first have to understand the source and nature of faults and failures.

In current large-scale compute clusters, main memory is one of the main causes of the hardware failures [36]. From a programmer's perspective, main memory (DRAM) is a computer component where programs write a byte of data, overwrite this data if needed, and correctly read it repeatedly until the machine is turned off. However, DRAM does not always comply to this rule. External events such as alpha particles, cosmic rays, or hardware corruptions caused by DRAM manufacturing process can change the data stored in DRAM cells and damage DRAM cells permanently or temporarily. Memory failures can also be the result of corruption along the data path between the memory and the processor. In each generation, the DRAM chip density and the amount of the main memory in the servers increases, while the technology process is constantly shrinking. Therefore, we expect that memory failures will have a serious impact on the future-systems reliability [19, 22]. The reliability is especially important in the high-performance computing (HPC) domain where tightly-coupled jobs comprising thousands of processes may execute for days [35]. In these systems, the failure of a single process leads to failure of the whole job, so reliability of each system component becomes an important limitation of the overall scalability.

The field studies of the DRAM errors are essential for the understanding of the nature of memory failures; failure patterns, rates and distributions. Additionally, they are the main requirement for quantifying the effectiveness of any error mitigation strategy. Therefore, researchers strive to get access to field data to analyze the errors that could and could not be corrected by Error-Correcting Code (ECC) mechanism, and investigate the impact and occurrence of memory errors even on systems without ECC [13]. This thesis describes a study of the DRAM errors observed on the MareNostrum 3 supercomputer [12] over period of more than two years.

1.2 Thesis contributions

1.2.1 Memory capacity requirements of HPC applications

The first problem that we address in the thesis is characterizing the memory capacity requirements of important HPC benchmarks and applications [121]. This analysis becomes increasingly important as 3D-stacked memories are entering the market. Compared with conventional DIMMs, 3D memory chiplets provide better performance and energy efficiency but lower memory capacities. Therefore the adoption of 3D-stacked memories in the HPC domain depends on whether we can find use cases that require much less memory than is available now. With good out-of-the-box performance, these use cases would be the first success stories for these memory systems, and could be an important driving force for their further adoption.

We theoretically analyze and confirm with experimental measurements the memory capacity requirements of High-Performance Linpack (HPL) and High-Performance Conjugate Gradients (HPCG), the former being the benchmark used to rank the supercomputers on the TOP500 list. We detect that HPCG could be an important success story for 3D-stacked memories in HPC. With low memory footprints and performance directly proportional to the available memory bandwidth, this benchmark is a perfect fit for memory systems based on 3D chiplets. HPL, however, could be one of the main show-stoppers because reaching a good performance requires memory capacities that are unlikely to be provided by 3D chiplets.

We also study the memory footprints of the Unified European Application Benchmark Suite (UEABS), large-scale scientific workloads carefully selected to provide good coverage of production HPC applications running on Tier-0 and Tier-1 HPC systems in Europe [89]. Most of the UEABS applications have per-core memory footprints in the range of hundreds of megabytes — an order of magnitude less than the main memory available in state-of-the-art HPC systems; but we also detect applications and use cases that still require gigabytes of main memory. We further demonstrate that even within the same application, different processes can have memory footprints that vary by an order of magnitude.

1.2.2 Large-memory nodes for energy-efficient HPC

The second problem that is investigated is the potential for saving energy through *scaling-in* on large-memory nodes [122]. Energy consumption is by far the most important contributor to HPC cluster operational costs, and it accounts for a significant share of the total cost of ownership. Advanced energy-saving techniques in HPC components have received significant research and development effort, but a simple measure that can dramatically reduce energy consumption is often overlooked. We show that, in capacity computing, where many small to medium-sized jobs have to be solved at the lowest cost, a practical energy-saving approach is to scale-in the application on large-memory nodes. Scaling-in refers to executing a fixed problem on a fixed machine, but using a reduced number of application processes and compute nodes. Scale-in increases single job execution time, but, as we quantify in our study, it substantially decreases energy consumption and reduces the running time of a batch of jobs. We investigate the sources of this energy savings, and show that its main source is a reduction in node-hours.

Scaling-in is limited by the per-node memory capacity, since, for a fixed size problem, reducing the number of nodes increases the memory required at each node. We therefore investigate the benefits of upgrading the per-node memory capacity in terms of energy savings and reducing the node-hours, and follow this with a financial cost-benefit analysis. We show that the additional energy savings mean that an investment in upgrading the memory would be typically recovered in less than five years.

1.2.3 DRAM errors in the field

Finally, we study the reliability of modern memory systems for HPC. We present a study of DRAM errors on the MareNostrum 3 supercomputer during a period of more than two years. Our study covers data logged on 3056 compute nodes, with more than 25,000 memory DIMMs and more than 2000 billion MB-hours of DRAM in the field. The field studies of DRAM errors are essential for the understanding of the nature, rates and distributions of errors in memory systems and are the main requirement for quantifying the effectiveness of any error mitigation strategy.

Our study clearly distinguishes between two different approaches for the DRAM error analysis, categorical and via error rates. The first approach compares the errors at the DIMM level and partitions the DIMMs into various categories, e.g. based on whether they did or did not experience an error. The second approach is to analyze the error rates, i.e., to present the total number of errors relative to other statistics, typically the number of MB-hours or the duration of the observation period. Although DRAM error analysis is typically performed with both approaches, we show that the approaches are not interchangeable, and can lead to completely different conclusions.

We find that the percentage of DIMMs that experience uncorrectable errors is very small. In contrary to the common belief that scaling down the technology reduces the DRAM reliability, our measurements show that percent of DIMMs that experience uncorrected errors is reduced in each DRAM generation. We further show that the findings based on the average errors rates, errors per MB-hour and mean time between failures, can be completely different depending on the moment in which the measurements are taken. We also show that using the correctable errors and faults rates as a DRAM reliability indicator is misleading because the uncorrected error trends can be completely different.

1.3 Thesis structure

The rest of the thesis is organized as follows:

- In Chapter 2, we describe basic architecture and mechanisms in modern DRAM based memory systems. We also briefly introduce emerging memory technologies: 3D-stacked DRAM and non-volatile memories.

- Chapter 3 presents the experimental environment used in our experiments. In this chapter, we describe the hardware platform, the applications and benchmarks, and software tools used in the study.

- Chapter 4 summarizes our analysis of memory capacity requirements of production HPC applications and two important HPC benchmarks: High-Performance Linpack (HPL) and High-Performance Conjugate Gradients (HPCG).

- In Chapter 5, we demonstrate that scaling-in of HPC applications can substantially decrease energy consumption and reduce the running time of a batch of jobs. We also show that upgrading the per-node memory capacity enables greater degree of scaling-in and additional energy savings.

- Chapter 6 presents our study of DRAM errors in the field that covers data logged over a period of more than two years on our supercomputer. In this chapter we propose a methodology for more formal analysis of memory system reliability.

- Chapter 7 summarizes the conclusions of the thesis, briefly describes future work, and lists the relevant publications.

CHAPTER 2

Background

In this chapter, we describe basic architecture of modern DRAM based memory systems and DRAM devices. The goal in this chapter is to give a broad overview of the main memory system, its common blocks and mechanisms, sufficient to provide a good understanding of modern DRAM memory systems. We also explain the organization of Dual In-line Memory Modules (DIMMs) and Error-Correcting Code (ECC) mechanisms needed to better understand our study of DRAM errors in the field (see Chapter 6). Finally, we present promising memory technologies that might replace or complement DRAM memory in the future and their implications on the capacity and reliability of memory systems.

2.1 DRAM memory system overview

The main memory system is comprised of a memory controller, memory bus and a number of DRAM chips organized into memory modules (DIMMs), as illustrated in Figure 2.1. The memory controller can be located on-chip or off-chip, and in current computer systems memory controller is usually found integrated on the chip with the CPU. Contrary, DRAM memory chips are always separate from the CPU to facilitate memory upgrades. Individual DRAM chips have small capacity and narrow data-bus, and therefore, are commonly grouped together and form memory modules, which provide larger capacity and wide data-bus. In current systems, the data-bus is usually 64-bit wide, or 72-bit wide when eight additional bits are added for error-correcting codes (ECC). Memory bus consists of several memory channels that connect memory controller with several memory modules.

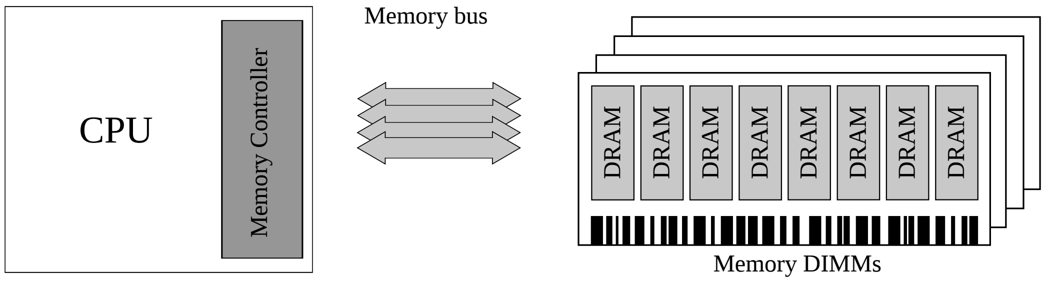


Figure 2.1: Main memory system comprises of memory controller, memory bus and a number of DRAM chips organized into DIMMs.

2.1.1 DRAM organization

DRAM cell is the main building block of DRAM memory. It comprises of one access-transistor and one capacitor where the data information is stored, see Figure 2.2. Depending on whether the capacitor is fully charged or discharged, the cell is in the binary state "1" or "0", respectively. When the storage capacitor is charged, it can hold the stored charge after the access-transistor is switched off. However, due to the capacitor leakage, stored charge can be retained only for a limited time. For this reason, all DRAM cells must periodically be refreshed in a process known as DRAM refresh.

DRAM cells are grouped together to form DRAM array structures. Every DRAM cell lies at the intersection of two perpendicular wires: a horizontal wordline and a vertical bitline. DRAM array is basically a two-dimensional matrix, where each *row* of cells shares a common wordline, and cells on the same *column* are connected to the same bitline, as illustrated in Figure 2.3. When a row wordline is raised to high-voltage, access-transistors are enabled,

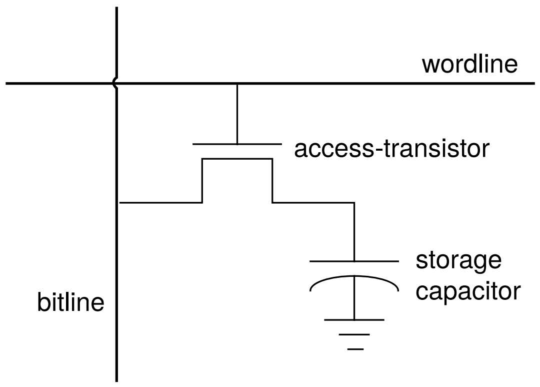


Figure 2.2: DRAM cell consists of one access-transistor and one capacitor. The cell is accessed through the corresponding wordline and bitline.

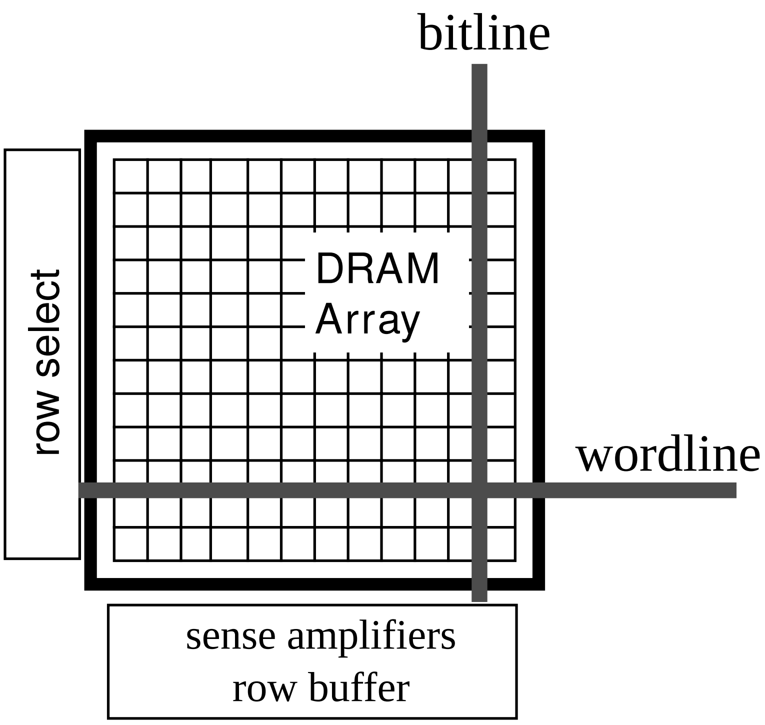


Figure 2.3: Large number of DRAM cells is grouped together to form DIMM array structure.

which connects all capacitors to the respective bitlines. Then the complete row of data is transfered to the row-buffer. In practice, each memory array is divided in memory sub-arrays, each providing one bit of data on a memory access. Each DRAM chip usually provides four or eight data bits (**DQs**), and these chips are called x4 and x8 DRAMs, respectively. All sub-arrays (four or eight) in a memory array are accessed in parallel, and each DQ provides a bit of data from a corresponding memory sub-array.

A group of rows is called a *bank*, and each bank has its own dedicated row-buffer. Therefore, multiple banks increase the parallelism, as multiple banks can be served concurrently. One bank spreads over several chips that are accessed in parallel, as shown in Figure 2.4. Finally, multiple banks come together and form a *rank*. So, a rank of memory comprises of several banks, and each bank spreads over several DRAM chips that are accessed in parallel. All chips in a rank respond to the same DRAM command. Each memory DIMM can contain one or more memory ranks. Figure 2.4 shows a 2GB rank, where 256K rows are vertically partitioned into eight banks of 32K rows, where each row is 64Kb = 8 KB (64K columns). Each DRAM chip has a capacity of 2Gb or 256MB.

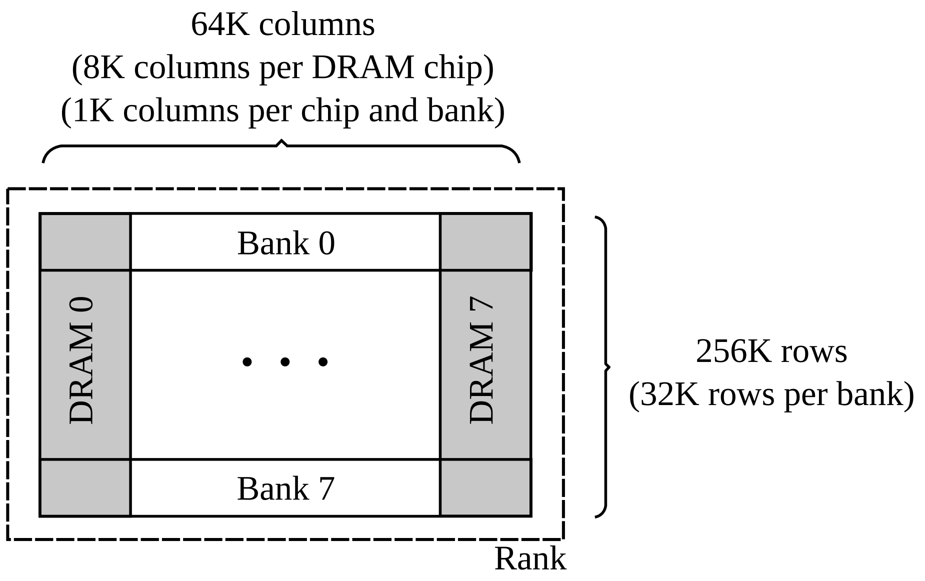


Figure 2.4: One rank of DRAM comprises of several banks, each spreading over several DRAM chips that are accessed in parallel.

To summarize, for each memory access, the memory address specifies the memory channel, memory module, rank, bank, row and column. This specifies a unique memory location inside every DRAM chip in the addressed rank. Then, in parallel, all DRAM chips provide data on their DQ pins, and data is read from the memory. For example, in case of 72-bit wide data bus and x8 DRAM chips, one rank of memory comprises of nine DRAM chips (eight chips for data and one for ECC data). Because these are x8 chips, every chip provides eight bits of data — each chip provides eight DQs (9 chips $\times$ 8 DQs = 72 bits). So, all DRAM chips provide eight data bits from eight memory sub-arrays, from the intersection of addressed row and column inside the addresses bank. This way, 72 bits of data are provided. One memory access implies providing data from several consecutive locations in memory (consecutive columns). This is know as a memory burst. The size of memory burst, i.e., the number of consecutive locations that provide data, is specific to DDR technology, and for DDR3 DRAM it is a burst of eight.

2.1.2 Error-Correcting Codes

In this section, we examine Error-Correcting Codes (ECC) techniques to detect and correct single-bit and multi-bit memory errors. We start from a simple one, and move towards more complex techniques present in modern DRAM memory systems.

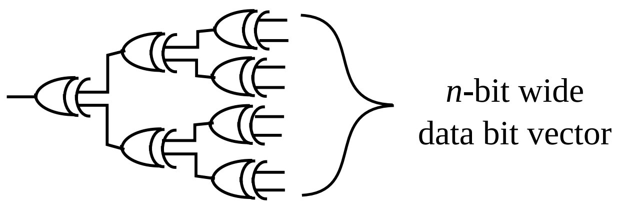


Figure 2.5: Parity bit generation. Parity bit is generated from a recursive application of the exclusive-or function to the bits in the data bit vector.

Parity is one way to provide minimal data protection — it provides single-bit error detection. It indicates whether even or odd number of "1" bits is in the data bit vector. Figure 2.5 illustrates the generation of a parity bit. It is generated from a recursive application of the exclusive-or function to the bits in the input data. The parity bit is stored in the DRAM memory system together with the original data. When the data vector along with the previously computed parity bit is read from the memory, the parity bit is recomputed from the vector and compared to the retrieved parity bit. If the recomputed parity bit differs from the retrieved parity bit, an error is generated and reported by the memory controller. Therefore, parity mechanism does not have data correction capability, it can simply detect if an odd number of bits has changed its state.

Single-bit Error Correction (SEC ECC) is the basic ECC algorithm that consists of adding redundant bits to a data word, mapping the original data vector into a longer word, and using Hamming distance between valid code words to correct the corrupted bit. The number of storage bits needed for error detection is proportional to $log_2 n$, where n is the number of bits in the data vector. Hamming distance is the number of bits that differ between two binary vectors. For example, Hamming distance between 10101010 and 10101001 is two, because only last two bits are different. In SEC ECC, all valid code words differ from each other by at least three bits, and between each two valid code words are two invalid code words, see Figure 2.6. This ensures that any valid SEC ECC code word can sustain a one-bit error. If any bit in a valid code word is corrupted, the code word is moved a Hamming distance of *one* away from its original place. It becomes an invalid code word, but, more importantly, it becomes an invalid code word at a Hamming distance of *one* from one and only one valid code word, which is the original valid code word that was corrupted. Thus, any single-bit error can be corrected by transforming the invalid code word into the nearest valid code word. However, note that a double-bit error creates an invalid code word that lies at a distance of two from the original valid code word, but a dis-

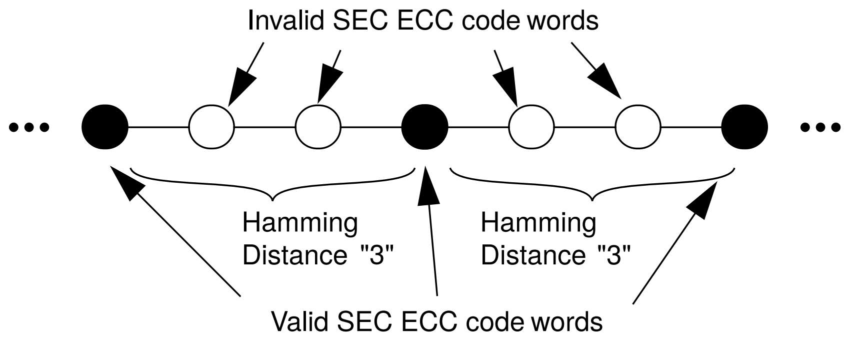


Figure 2.6: Humming distance illustration.

tance of one away from a different, totally unrelated valid code word. Thus, a two-bit error will be corrected to the wrong value.

Single-Error Correction Double-Error Detection (SECDED) ECC algorithm upgrades SEC ECC by being able to distinguish single-bit errors from double-bit errors. Compared to SEC ECC, an additional parity bit is added to create the SECDED ECC. A parity checking bit (on the whole data vector) is added and it provides a quick sanity check to ensure that, in case the ECC syndrome is a non-zero vector and indicates an error, the error is not a double-bit error. SECDED ECC is able to correct single-bit errors and detect double-bit errors with minimal overhead in storage capacity. For a data bus width of 64 bits ($n = 64$), the SECDED ECC algorithm requires 8 check bits. This is the usual data with in modern memory systems, where 72 bits on the memory bus comprise of 64 data bits and 8 ECC bits.

Advanced ECC mechanisms can correct a multi-bit error within one DRAM chip; these ECC techniques can correct a complete DRAM chip failure, i.e., the **Chipkill** capability [21, 36, 58]. One way to provide such a capability is illustrated in Figure 2.7. This solution ensures that a single chip, independently of its data width, does not affect more than one bit in any given ECC word. For example, in case of x4 DRAMs, each of four DQ pins affects a different ECC word. Therefore, in case of a whole DRAM chip failure, none of the ECC words would experience more than one bit of corrupted data. This can be easily corrected with standard SEC ECC. This way, Chipkill can provide correction even in case of failure of an entire DRAM chip. Other methods provide more ECC bits (e.g. 144-bit ECC word that consists of 128 data bits and 16 ECC bits) so that each ECC word can correct more than a single-bit failure. In industry, some combination or variation of these two methods is usually used in order to provide Chipkill capability.

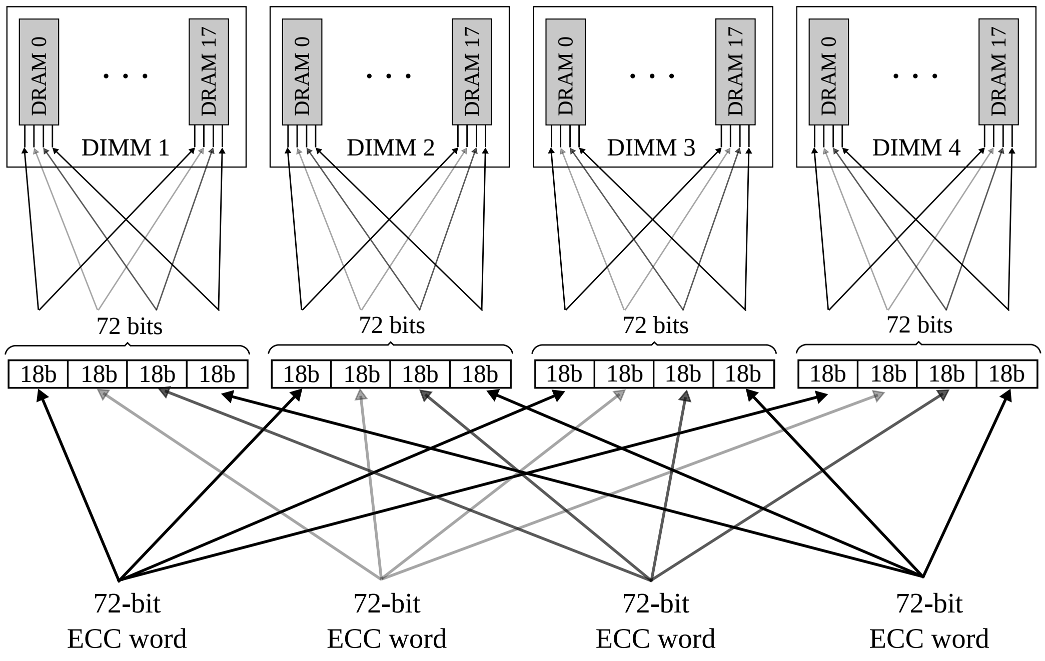


Figure 2.7: Chipkill illustration.

2.2 Emerging memory technologies

Existing DRAM based memory systems face inherent limitations that impede scaling of both memory bandwidth and capacity. Moreover, analyses of the modern HPC systems show that the main memory is one of the major contributors to the total energy consumption and the operational cost. With these issues in mind, we have enough reasons to be concerned whether conventionally-used memory architectures based on mature DRAM technology and DIMM organization will meet the needs of future HPC systems and applications. In this section, we briefly describe some of the promising memory technologies that have recently attracted attention in the community.

2.2.1 3D-stacked DRAM

A potentially promising solution for memory bandwidth and energy-efficiency problems of conventional DIMM based DRAM memory is the use of 3D-stacked DRAMs. 3D stacking increases package density, with memory chiplets placed on a silicon interposer instead of a printed circuit board. Stacked DRAM dies are connected using through-silicon vias (TSVs), which shorten the interconnection paths and reduce connectivity impedance and channel latency. Hence, data can be moved at a higher rate with a lower energy per bit.

In this section, we briefly overview the currently-available products based on this technology: Hybrid Memory Cube (HMC) and High-Bandwidth Memory (HBM). We also describe Intel Knights Landing, the first processor that brings in 3D-stacked DRAM in addition to the traditional DDR DIMMs.

Hybrid Memory Cube

The Hybrid Memory Cube (HMC) [37] is connected to the CPU with a high-speed serial interface that provides up to 480 GB/s per device. 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. Memory capacity can be increased by integrating multiple HMC devices into the package, but doing so is non-trivial. Each HMC device can be directly connected to up to four other devices (CPUs or HMCs) via four independent serial links. Figure 2.8 illustrates the HMC internal structure. HMC is composed of stacked DRAM dies (dies 0 to n) 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. 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.

High-Bandwidth Memory

High-Bandwidth Memory (HBM) [44] is connected to the host CPU or GPU with a wide 1024-bit parallel interface that delivers up to 256 GB/s. Similarly to HMC, the standard specifies up to 8 GB devices and integrating multiple devices in the package is challenging. Only a single HBM device can be connected to each interface (channel), so using multiple HBM devices requires a large silicon interposer with multiple 1024-bit wide interfaces, increasing cost. Figure 2.9 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.

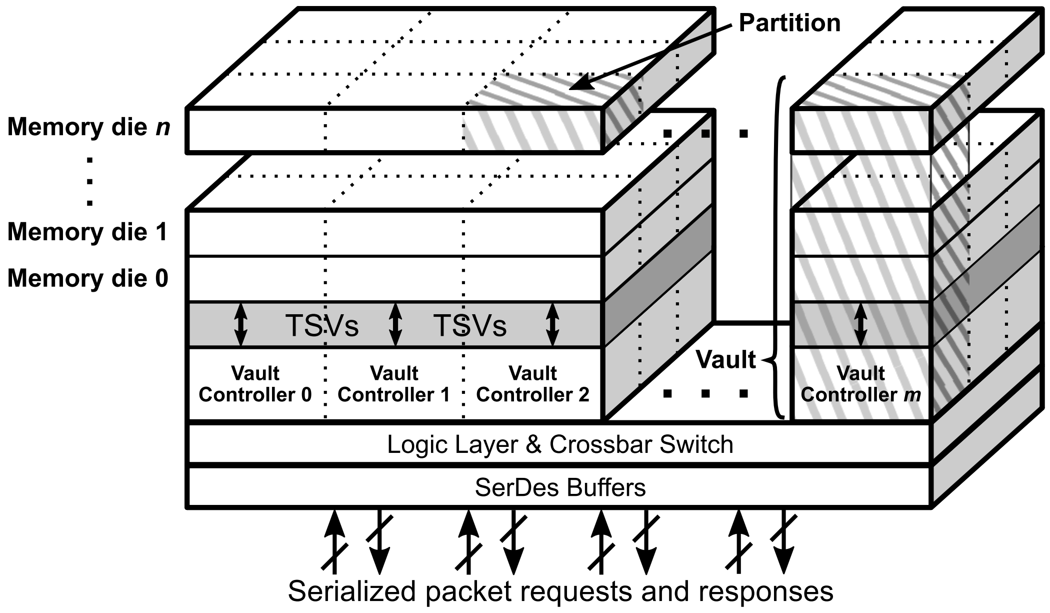


Figure 2.8: HMC internal structure.

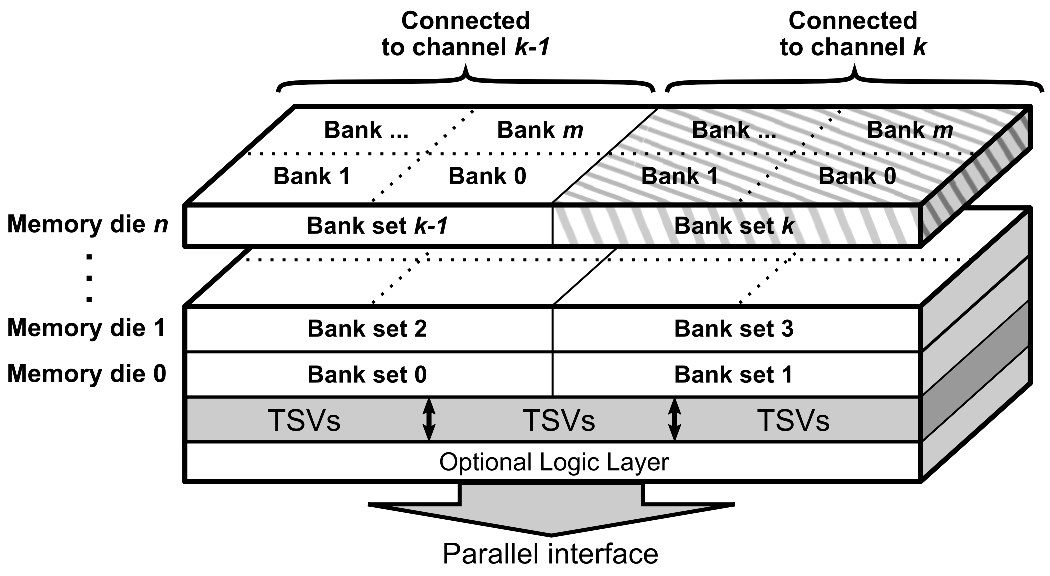


Figure 2.9: HBM internal structure.

Heterogeneous memory system: Intel Knights Landing

Intel Knights Landing (KNL) processors [103] are the first CPUs that bring in 3D-stacked DRAM in addition to the traditional DDR DIMMs. KNLs comprise up to 72 cores supported by two levels of main memory. At the first level, **3D multi-channel DRAM (MCDRAM)** is connected to the CPU through an on-package interposer and it offers a capacity of up to 16 GB (0.2 GB per core) with 400 GB/s of peak theoretical bandwidth. In addition to MCDRAM, KNL can be connected to up to 384 GB (5.4 GB per core) of standard DDR4 memory. MCDRAM and DDR can be organized in three modes: *cache*, *flat* and *hybrid*. In the cache mode, MCDRAM behaves as an additional (L3) level of the cache hierarchy. In the flat mode, MCDRAM and DDR are two distinct memory nodes with different capacity, latency and bandwidth that can be addressed by different APIs. In case that the input dataset does not fit into the MCDRAM, it is responsibility of the programmer to perform data partitioning, allocation and migration that would use efficiently the advanced memory organization. Finally, the hybrid mode combines the cache and the flat mode. In this mode, the MCDRAM is partitioned into two segments — one is used as the L3 cache for the DDR, while the other is addressable and used as MCDRAM memory node in the flat mode.

2.2.2 Non-volatile memories

In recent years, research and development of non-volatile memory (NVM) technologies has escalated. Some of the technologies have been known for decades, but, with raised concerns about the DRAM technology scaling, NVM development, improvement and its suitability to serve as a main memory has increased. These include Phase Change Memory (PCM) [117], Spin-Transfer Torque Magnetoresistive (STT-MRAM) [6], Resistive RAM (RRAM), etc. These emerging technologies usually involve a trade-off and seem unlikely to completely replace DRAM. For example, PCM is advantageous over DRAM because of higher density, non-volatility (no refreshing), and low idle power consumption. On the other hand, PCM has shortcomings compared to DRAM, which include higher latency, higher read and write energy, and limited endurance.

However, it seems that any of the non-volatile solutions may be added as a complement to DRAM memory DIMMs, as no solution is strictly superior to DRAM. Even recently advertised non-volatile solution from Intel and Micron, 3D XPoint, seems likely to be considered as an another level of hierarchy between DRAM and storage, instead of completely replacing DRAM. In that cotext, recently available Intel Knights Landing processors bring 3D-stacked DRAM in addition to the traditional DDR DIMMs [103]. Then, users can configure whether these memories will be accessed separately, or 3D-stacked DRAM will serve as an another level of cache to the DIMM based DRAM. All these technologies, however, will have to be tested in many domains and platforms before being adopted as mainstream memory solutions in HPC.

2.2.3 Implications on memory system capacity and reliability

3D stacked memory solutions offer significantly higher memory bandwidth and energy efficiency, but higher production cost compared to DIMM based DRAM means that the capacities will be limited at least in their first generations. Because of limited capacities, 3D stacking memory technologies seem likely to be implemented alongside DIMM based DRAM in a heterogeneous memory system. Such a memory solution is already present on the market in Intel Knights Landing processor [46]. On the other hand, non-volatile memories offer higher capacities and more density, but none of NVM solutions is strictly superior to DRAM. Because of higher latency, these solution are mostly considered as an another level of memory hierarchy, between DRAM main memory and storage.

Several layers of stacked DRAM lead to increased power dissipation and to the lack of sufficient thermal dissipation. Therefore, higher temperatures in 3D stacked DRAM memories raise reliability concerns. For this reason, HMC has implemented in-memory ECC for command, address and data signals. NVM solutions have reliability concerns depending on the technology; e.g. MRAM devices are immune to the effects of incident alpha particles or cosmic ray neutrons because their cells operate on the principle of magnetics and not on the storage of electrical charges. In conclusion, every different memory solution has its own sources of reliability concerns, and none of the solutions will be completely immune to errors.

CHAPTER 3

Experimental methodology

This chapter presents the high-level experimental environment used in our research. Later, each chapter covers the specifics of the experimental methodology used for the separate part of the study. The experimental methodology used for the analysis of memory capacity requirements of HPC applications is presented in Chapter 4. Chapter 5 includes the description of the environment used for the experiments related to the study of scaling-in of applications to reduce the energy consumption. Chapter 6 covers the experimental environment used for the field study of DRAM errors.

3.1 Hardware platform

We execute all our experiments on the MareNostrum 3 supercomputer [10], which is one of the six Tier-0 (largest) HPC systems in the Partnership for Advanced Computing in Europe (PRACE) [90]. Researchers in Spain and across Europe use it for simulations and experiments in various scientific areas, such as climate analysis and weather forecast, geophysics, bioinformatics, etc.

MareNostrum 3 is a Petaflop machine and has a peak performance of 1.1 Petaflops. MareNostrum 3 contains 3,056 compute nodes connected via InfiniBand FDR 10. Figure 3.1 illustrates one compute node. Each compute node comprises of two eight-core Sandy Bridge-EP E5-2670 processors at 2.6 GHz. Each core in a processor has local L1 and L2 caches, and eight cores share one 20 MB L3 cache.

The processors connect to main memory (DDR3-1600 DIMMs) through four channels. Regular MareNostrum compute nodes include 32 GB of DRAM memory (8 DIMMs $\times$ 4 GB), i.e., 2 GB per core. Large-memory nodes contain 128 GB of DRAM (8 DIMMs × 16 GB), i.e., 8 GB per core. Later, in each chapter, we specify whether experiments were executed on regular or large-memory nodes on MareNostrum. One MareNostrum node provides memory bandwidth of 77.86 GB/s. We measured the sustained memory bandwidth using STREAM triad benchmark [66]. Main memory latency on our system is 102.93 ns; a result obtained using LMbench benchmark suite [68].[1]

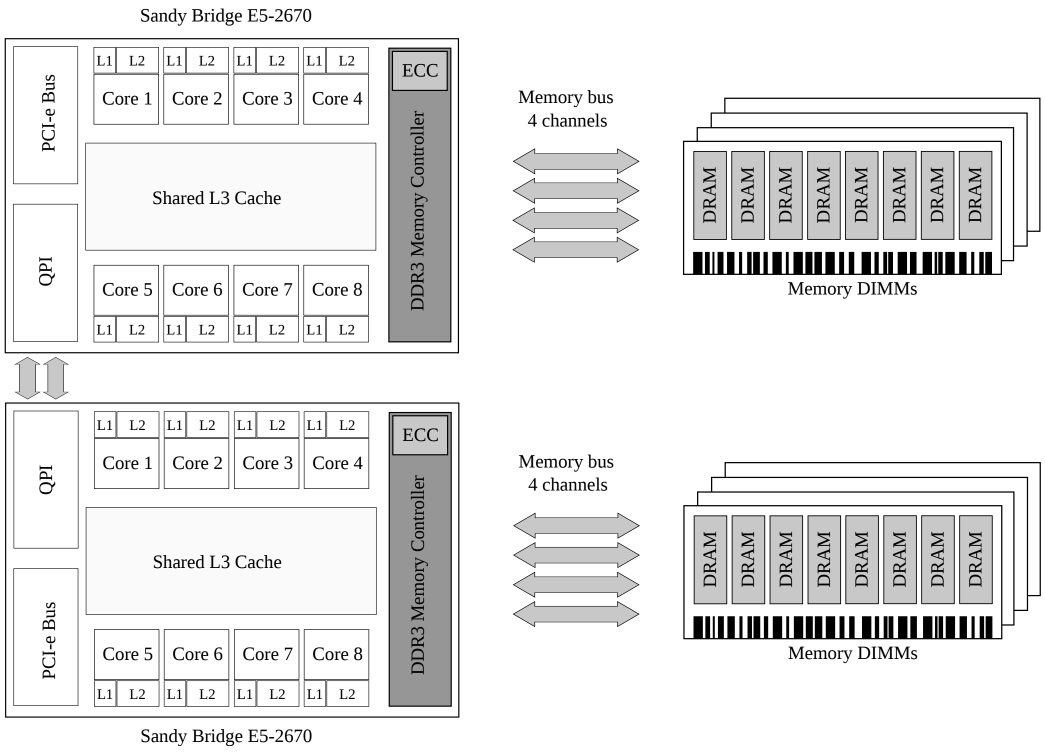


Figure 3.1: MareNostrum 3 compute node comprises of two eight-core Sandy Bridge processors. Each processor connects to main memory through four memory channels.

3.2 Applications

In our experiments, we execute important HPC benchmarks, as well as production HPC applications. All workloads are parallelized using Message

[1]LMbench suite contains several benchmarks which measure performance of different hardware and software components in a system. We used memory read 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 input load size, we measure access latency to all memory hierarchy levels.

Passing Interface (MPI), and we always execute one MPI process per core.

3.2.1 HPC benchmarks

We analyze two widely used HPC benchmarks, High-Performance Linpack (HPL) and High-Performance Conjugate Gradients (HPCG). Both benchmarks are used to rank supercomputers worldwide.

High-Performance Linpack

For more than 20 years, the High-Performance Linpack (HPL) benchmark is the most widely recognized and discussed metric for ranking of HPC systems on TOP500 list [113]. TOP500 list shows the 500 most powerful commercially available supercomputing systems in the world. HPC systems are ranked based on the HPL performance, and the list is regularly updated every six months.

HPL measures the sustained floating-point rate (GFLOP/s) for solving a dense system of linear equations using double-precision floating-point arithmetic on distributed-memory computers. 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 size to achieve the best performance on a given system. In our experiments, we execute the 2.1 version of the HPL benchmark.

High-Performance Conjugate Gradients

The High-Performance Conjugate Gradients (HPCG) benchmark [111] has been introduced as a complement to HPL and the TOP500 rankings, since the community questions whether HPL is a good proxy for production applications. HPCG is based on an iterative sparse-matrix conjugate gradient kernel with double-precision floating-point values, and is representative of HPC applications governed by differential equations. Such applications tend to have much greater demands on the memory system, in terms of bandwidth and latency, and they access data using irregular patterns [27]. In our experiments, we execute the 2.4 version of the HPCG benchmark.

3.2.2 Production HPC applications

We also study the Unified European Application Benchmark Suite (UEABS) [89], the set of production applications and datasets designed for benchmarking the European Partnership for Advanced Computing in Europe (PRACE) HPC systems for procurement and comparison purposes [90]. This application suite comprises codes from several scientific domains, and provides a set of scalable, currently relevant and publicly available codes and datasets of a size which can realistically be run on large HPC systems.

We study 10 of the 12 UEABS applications:[2]

- 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.

- 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

[2]We could not finalize the Code_Saturne and GPAW installations. These errors have been reported to the application developers.

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.

- SPECFEM3D simulates three-dimensional global and regional seismic wave propagation based upon the spectral-element method. Spectral-element method was originally developed in computational fluid dynamics and has been successfully adapted to address problems in seismic wave propagation. It is a continuous Galerkin technique, which can easily be made discontinuous; it is then close to a particular case of the discontinuous Galerkin technique, with optimized efficiency because of its tensorized basis functions.

In this thesis, we run all UEABS applications parallelized using the Message Passing Interface (MPI). UEABS applications are regularly executed on hundreds to thousands of cores, and most of them come with two input datasets. Smaller datasets (Test Case A) are deemed suitable for Tier-1 systems up to about 1000 cores, and larger datasets (Test Case B) target Tier-0 systems up to about 10,000 cores. For BQCD, GADGET and NEMO, a single dataset (Test Case A) is provided that is suitable for both system sizes. Table 3.1 summarizes the applications and input datasets. For each application, we show the area of science that it targets and briefly describe the input dataset.

3.3 Tools

In this section, we present software tools used to conduct our experiments; tools used for the instrumentation of MPI applications and software daemons used for logging the memory errors on our supercomputer.

3.3.1 Extrae

Extrae [11] is a dynamic instrumentation package used to trace programs compiled with the shared memory model (OpenMP and pthreads), the Message Passing Interface (MPI) programming model, or both programming models (different MPI processes using OpenMP or pthreads within each MPI process). Extrae generates trace files for a post-mortem analysis of parallel applications. It uses different interposition mechanisms to inject probes into

Table 3.1: Production HPC applications analyzed in the thesis

Application	Area of science	Test Case A: Smaller input dataset	Test Case B: Larger input dataset
ALYA	Computational mechanics	27 million element mesh	552.9 million element mesh
BQCD[a]	Particle physics	$32^2 \times 64^2$ lattice	N/A
CP2K	Computational chemistry	Energy calculation of 1024 waters	216 LiH system with Hartree-Fock exchange[b]
GADGET	Astronomy and cosmology	135 million particles	N/A
GENE	Plasma physics	Ion-scale turbulence in Asdex-Upgrade	Ion-scale turbulence in Jet
GROMACS	Computational chemistry	150,000 atoms	3.3 million atoms
NAMD	Computational chemistry	2×2×2 replication of the STM Virus	4×4×4 replication of the STM Virus
NEMO	Ocean modeling	12° global configuration; 4322×3059 grid	N/A
QE	Computational chemistry	112 atoms; 21 iterations	1532 atoms; two iterations
SPECFEM3D	Computational geophysics	6×12×768 mesh of the earth	6×24×1760 mesh of the earth

[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] CP2K cannot run Test Case B on our platform. The errors have been reported to the application developers.

the target application so as to gather information regarding the application performance. Extrae captures time-stamped events, e.g. entry/leave of a MPI function call, and provides support for gathering additional statistics such as performance counters values at each sampling point.

We get a complete picture about applications performance running on our target platform by combining time-stamps and performance counters. We also use modified version of Extrae, which is able to capture the memory usage of applications, both on each entry/leave of MPI function calls and by periodic sampling of application (more details can be found in Section 4.2).

3.3.2 Paraver

For the visualization and analysis of Extrae traces we use Paraver [9]. Paraver is powerful and flexible graphical user interface (GUI) data browser.

The tool has been demonstrated to be very useful for performance analysis studies, giving much more details about the applications behavior than most performance tools. It supports trace visualization in terms of timelines and histograms, and allows for detecting OS and hardware issues and different imbalances found in parallel applications.

3.3.3 Limpio

Limpio [85] is a framework for profiling of MPI applications. Limpio 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 to fit the requirements of the analysis. Limpio can invoke external application profiling tools, and can switch between various tools in a single execution. It can also generate application traces of timestamped events that can be visualized by general-purpose visualization tools or libraries. Limpio is regularly used in Barcelona Supercomputing Center for the instrumentation of large-scale HPC applications.

In this thesis, we use Limpio for application profiling and analysis, i.e. calculating the percentage of execution time spent in communication and computation and measuring the application memory usage (see Section 5.3.2).

3.3.4 Error logging daemons

In our study of DRAM errors in the field, we analyze errors that are corrected by Error-Correcting Code (ECC) mechanism inside the memory controller, as well as errors that could not be corrected by ECC. These two types of errors, corrected and uncorrected errors, are collected from two sources. Corrected error events are logged by daemon running on each node, a modified `mcelog` Linux kernel module [47]. Uncorrected errors are recorded in a separate log by IBM firmware after the node reboots as a result of the uncorrected error [39]. More detailed explanation about the memory error logging can be found in Chapter 6.

CHAPTER 4

Memory capacity requirements of HPC applications

This chapter analyzes the memory capacity requirements of important HPC benchmarks and applications. This analysis becomes increasingly important as 3D-stacked memories are entering the market. Compared with conventional DIMMs, 3D memory chiplets provide better performance and energy efficiency but lower memory capacities. Therefore the adoption of 3D-stacked memories in the HPC domain depends on whether we can find use cases that require much less memory than is available now. With good out-of-the-box performance, these use cases would be the first success stories for these memory systems, and could be an important driving force for their further adoption.

We find that the High Performance Conjugate Gradients benchmark could be an important success story for 3D-stacked memories in HPC, but High-performance Linpack is likely to be constrained by 3D memory capacity. The study also emphasizes that the analysis of memory footprints of production HPC applications is complex and that it requires an understanding of application scalability and target category, i.e., whether the users target capability or capacity computing. The results show that most of the HPC applications under study have per-core memory footprints in the range of hundreds of megabytes, but we also detect applications and use cases that require gigabytes per core. Overall, the study identifies the HPC applications and use cases with memory footprints that could be provided by 3D-stacked memory chiplets, making a first step towards adoption of this novel technology in the HPC domain.

4.1 Introduction

Memory systems are important contributors to the deployment and operational costs of large-scale HPC clusters [49], [109], [102], making memory provisioning one of the most important aspects of HPC system design.[1] In spite of this, most available analysis guiding memory provisioning is surprisingly ad hoc. Usually large HPC systems follow a rule of thumb that couples 2–3GB of main memory per x86 core or 1 GB per Blue Gene PowerPC core (see Figure 4.1). It seems that this rule of thumb is based on experience with previous HPC clusters and on undocumented knowledge of the principal system integrators, and it is uncertain whether it matches the memory requirements of production HPC applications.

Even though there are various reports and projections that roughly estimate the memory requirements of existing HPC applications [7], [74], [75], [76], [77], [78], [79], there are no or very few studies that thoroughly analyze and quantify the memory footprints of HPC workloads across multiple domains. In this study we try to bridge this gap and to examine whether current memory design strategies meet the memory requirements of important

[1]In our system, the MareNostrum supercomputer [10], main memory accounts to more than 10% of server cost and 10–15% of server energy consumption.

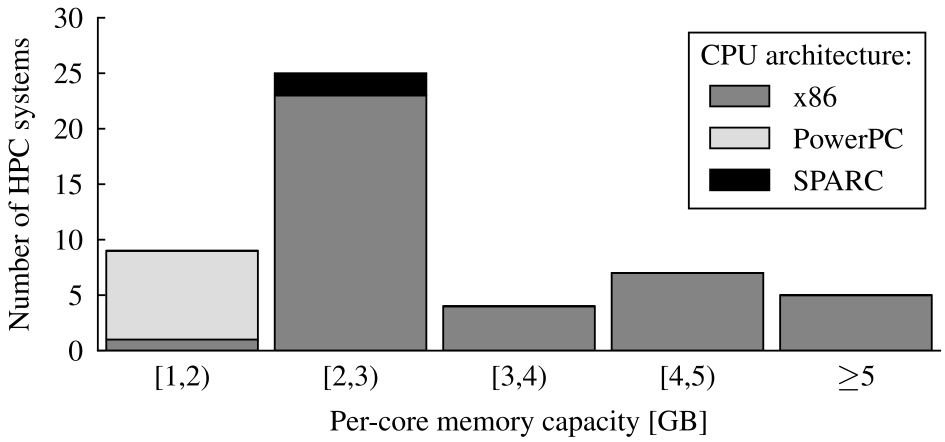


Figure 4.1: Per-core memory capacity of HPC systems leading the TOP500 list (June 2015). Systems with exactly 2, 3, and 4 GB of memory per core are included in bars [2,3) GB, [3,4) GB, and [4,5) GB, respectively. Today's HPC systems are dominated by x86 architectures coupled with 2–3 GB of main memory per core. The next most prevalent systems are Blue Gene platforms based on PowerPC cores with 1 GB of memory per core, included in the *[1,2) GB* bar.

HPC benchmarks and applications.

In this study, we theoretically analyze and confirm with experimental measurements the memory capacity requirements of High-Performance Linpack (HPL) and High Performance Conjugate Gradients (HPCG), the former being the benchmark used to rank the supercomputers on the TOP500 list. Our measurements show that in current systems achieving good HPL scores requires at least 2 GB of main memory per core, which matches the main memory sizing trends of the large HPC clusters that dominate the TOP500 list. The analysis also shows that, as the total number of cores is increased, more memory per core will be needed to achieve good performance, between 7.6 GB and 16.1 GB in a million-core cluster. In contrast, HPCG memory requirements are fundamentally different. To converge to the optimal performance, the benchmark requires roughly 0.5 GB of memory per core, and this will not change as the cluster size increases.

We also study the memory footprints of the Unified European Application Benchmark Suite (UEABS), large-scale scientific workloads carefully selected to provide good coverage of production HPC applications running on Tier-0 and Tier-1 HPC systems in Europe [89]. We observe a bimodal distribution in memory requirements, finding that memory requirements depend on application scalability and the targeted HPC category, i.e., whether the workloads represent capability or capacity computing. In HPC, capability computing refers to using large-scale HPC installations to solve a single, highly complex problem *in the shortest possible time*, while capacity computing refers to optimizing system efficiency to solve as many mid-size or smaller problems as possible at the same time *at the lowest possible cost.* Based on our findings, we recommend guidelines for selecting an appropriate level of parallelism when designing experiments to quantify memory capacity requirements of production HPC applications. Most of the UEABS applications have per-core memory footprints in the range of hundreds of megabytes — an order of magnitude less than the main memory available in state-of-the-art HPC systems; but we also detect applications and use cases that still require gigabytes of main memory. We also demonstrate that even within the same application, different processes can have memory footprints that vary by an order of magnitude.

To the best of our knowledge, this is the first study that detected and analyzed the dependency between available memory capacity and HPL and HPCG performance. Also, for the first time, we explored the complexity of memory footprint analysis for production HPC applications, and showed how memory footprint depends on application scalability and target HPC category. We hope that this study will motivate the community to question the current trends for memory system sizing in HPC clusters, and will lead to further analysis of memory capacity requirements of HPC systems.

This analysis becomes increasingly important as 3D-stacked memories are hitting the market. Replacing conventional DIMMs with new 3D memory chiplets located on the silicon interposer could be the next breakthrough in memory system design. It would provide significantly higher memory bandwidth and lower latency, leading to higher performance and energy-efficiency. On the down side, however, it is unlikely that (expensive) 3D memory chiplets alone would provide the same memory capacities as DIMM-based memory systems [103]. Therefore the adoption of 3D-stacked memories in the HPC domain depends on whether we can find use cases that require much less memory than is available now.

Academia and industry are also exploring hybrid memory systems that combine 3D-stacked DRAM with standard DIMMs [25], [20], [100], [69], [103]. The general idea behind these hybrid systems is to bring the best of two worlds — the bandwidth, latency and energy-efficiency of 3D-stacked DRAM together with the capacity of DIMMs. In these systems, however, good performance requires efficient data allocation and migration between different memory segments. Data management requires profound application profiling, and up to now, no automatic algorithms — whether in the hardware, compiler or runtime environment — can provide out-of-the-box performance for legacy codes. Instead, efficient use of the advanced memory organization is still the responsibility of the programmer, which has significant impact on code development cost [80], [18], [45].

Therefore, in the context of hybrid memory systems, it is still important to find use cases with (small) memory footprints that fit into the 3D-stacked memory. With good out-of-the-box performance, these use cases would be the first success stories for 3D memory systems. Our study indeed identified the HPC applications and use cases with memory footprints that could be provided by 3D-stacked memory chiplets, making a first step towards adoption of this novel technology in the HPC domain.

4.2 Experimental setup

We analyze the memory footprints of HPC applications running on a large-scale cluster. We first describe our hardware platform and applications, and then we explain how we gather our data.

4.2.1 Hardware platform

We execute all experiments on the MareNostrum supercomputer [10] (see Chapter 3). The interconnect is InfiniBand FDR-10 (40 Gb/s), with a non-blocking two-level fat-tree topology offering full bisection bandwidth, built from 36-port leaf switches and 648-port core switches.

Regular MareNostrum compute nodes include 32 GB of DRAM memory (8 DIMMs × 4 GB), i.e., 2 GB per core. To study application memory footprints in systems with higher capacity, we execute some experiments on large-memory nodes containing 128 GB of DRAM (8 DIMMs × 16 GB), i.e., 8 GB per core.

4.2.2 Applications

We study memory capacity requirements for two widely used HPC benchmarks, High-Performance Linpack (HPL) and High-Performance Conjugate Gradients (HPCG) (see Chapter 3). We also study the Unified European Application Benchmark Suite (UEABS) [89], the set of production applications and datasets designed for benchmarking the European PRACE HPC systems for procurement and comparison purposes [90]. For more details, please see Chapter 3. All UEABS applications are parallelized using MPI, and are regularly run on hundreds or thousands of cores. For all applications, we run MPI-only versions, and always execute one MPI process per core. So, in the rest of the chapter *per-process* and *per-core* memory footprint have equivalent meanings.

4.2.3 Methodology

In this study, we measure the memory footprints of HPC applications running with various numbers of processes. We obtain footprint information from the */proc/*[*pid*]*/status* system files, which together log the memory usage of all running processes. The memory footprint of a process corresponds to the amount of physical memory it uses, i.e., the resident set size, or VmRSS. We use the Extrae and Limpio instrumentation tools [11], [85] to read and log VmRSS values at equidistant time intervals chosen to provide at least 1000 samples per process. We track, in each experiment, the maximums, means, and standard deviations of all memory footprint measurements. For production HPC applications, unless specifically stated that we analyze the master process, we report footprints of worker processes. For HPL and HPCG, we report memory footprints for all the processes.

4.3 High-Performance Linpack

For more than 20 years, the High-Performance Linpack (HPL) benchmark is the most widely recognized and discussed metric for ranking of HPC systems [113]. HPL measures the sustained floating-point rate (GFLOP/s) for solving a dense system of linear equations using double-precision floating-point arithmetic. The linear system is randomly generated, with a user-specified size, so that the user can scale the problem size to achieve the best performance on a given system. The documentation recommends setting a problem size that uses approximately 80% of the available memory [87]. We determine how reducing the memory capacity would affect performance by appropriately decreasing the problem size. Specifically, we set the problem size in order to use 80% of the *target* main memory capacity and verify at runtime that the memory footprint fits inside the target memory capacity.

4.3.1 Measured memory requirements

Figure 4.2 shows relative HPL performance in FLOP/s (Y-axis) for various amounts of available memory per core (X-axis) and different numbers of processes (different lines on the chart). We plot performance relative to the results with 32 MB of main memory, the smallest amount of memory used in any of these experiments. The experiments are performed for a range of 16–16,384 processes, and in each experiment the number of processes equals the number of cores.

As we increase the available main memory, HPL performance first increases, then reaches the saturation point and remains approximately constant. We observe this trend for any number of HPL processes. Also, we detect that stable HPL performance (after the saturation point) is directly proportional to the number of processes used in the experiment. For instance, increasing the number of processes from 16 to 64, 256, and 1024 increases the steady-state performance by roughly 4×, 16×, and 64× respectively.[2] We also detect that, as we increase the number of processes, the saturation point moves towards larger per-core memory capacities; for instance, 16, 256, and 4096 processes reach their saturation points at 256 MB, 768 MB, and 1280 MB, respectively. This means that for larger HPC clusters, more memory per core is required to achieve maximum HPL performance. When HPL comprises 16,384 processes (executed on one third of the MareNostrum supercomputer), the saturation point reaches 1.5 GB of memory per core.

[2]This trend is not visible in Figure 4.2 because for each number of processes the relative performance is normalized to the 32 MB results.

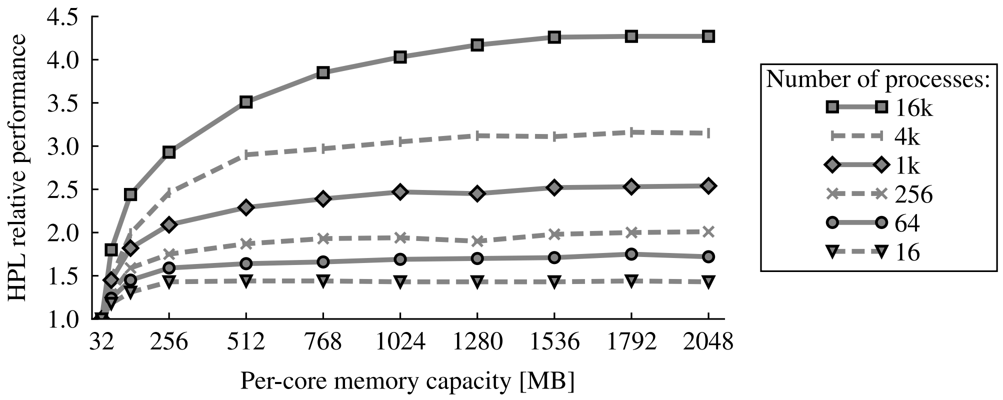


Figure 4.2: HPL performance also depends on the available memory capacity. When increasing per-core memory, HPL performance first increases and then reaches the saturation point — the sustained floating-point rate (GFLOP/s) of the system. As we increase the number of processes, the saturation point moves towards larger per-core memory capacities.

These results indicate that on large HPC systems with tens or hundreds of thousands of high-end x86 cores, reaching the HPL performance saturation point, or at least the point of diminishing returns, requires approximately 2 GB of memory per core. This matches the main memory sizing trends of the large HPC systems that dominate the TOP500 list.

4.3.2 Analysis

In order to confirm and better understand our measurements, we analyze how HPL performance depends on single-core floating-point rate, available memory capacity, interconnect bandwidth and latency, and how this dependency changes as we vary the number of processes used in the HPL runs. In this section, we summarize the analysis focusing on the conclusions that have an impact on the memory system sizing; detailed step-by-step explanation and mathematical formulas are presented in Section 4.3.3.

Our analysis confirms the HPL performance trends presented in Figure 4.2. For small input datasets, HPL has relatively poor overall performances because of high interprocess communication overheads. As the input dataset increases, the computation factor becomes dominant, the communication overhead mitigates and the HPL performance converges to the saturation point. For large per-process memory capacities (large input datasets), the HPL execution time is dominated by the dense matrix-vector multi-

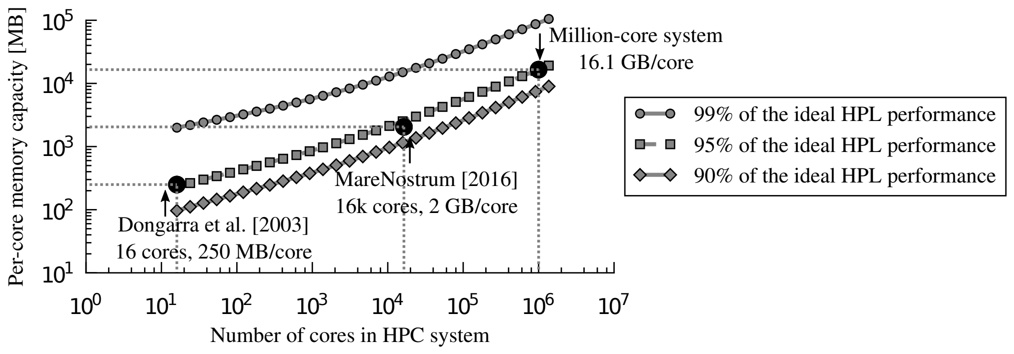


Figure 4.3: Per-core memory needed to get 90%, 95% and 99% of the ideal HPL performance, for different sizes of HPC systems. Three systems, with 250 MB/core (year 2003), 2 GB/core (year 2016) and 16.1 GB/core (future potential system) are compared with our estimation curves. Moving from 95% to 99% of ideal HPL performance requires a huge step in the amount of per-core memory (axes are plotted on logarithmic scales).

plication computational routines that require significant processing power. In current HPC systems with gigabytes of main memory, HPL is clearly a CPU bound application typically able to achieve close to the theoretical peak floating-point rate. Our theoretical analysis indeed shows that as the memory capacity is increased, the HPL performance analytically converges to a steady value proportional to the number of processes used in the HPL run, but the performance optimum is theoretically reached for *infinite main memory*.

We also analytically quantify the HPL performance loss due to a finite main memory capacity and determine the capacity required to reach close-to-optimal HPL performance, e.g., within 10%, 5% and 1% of the optimal. We fit the HPC system parameter constants in the formulas to our experimental data (see Fig. 4.2) and estimate main memory capacity that will lead to good HPL scores in larger systems. Since we fit the constants to *our hardware platform*, this analysis shows the *trend* of the main memory needed to achieve good HPL performance for different system sizes, and not the firm values. The outcome of this analysis is presented in Figure 4.3.

Figure 4.3 plots the amount of memory per core (Y-axis) needed to achieve 90%, 95% and 99% of the ideal (infinite memory) HPL performance. We present the results for a wide range of system sizes, from 16 up to over 1,000,000 cores (X-axis). Both axes plot the values in a logarithmic scale. In scaling the interconnect, we assume that latency remains constant and bisection bandwidth scales with the number of cores.[3] This is a conservative assumption because interconnect latency may increase with the number of cores, which would *further increase* the per-core memory requirement for large systems. From Figure 4.3, we see that increasing the system size causes more memory per core to be needed to achieve good HPL performance (recall that both axes of Figure 4.3 have a logarithmic scale). For systems with 100,000 cores, 2.6 GB and 5.5 GB of per-core memory would be needed to reach 90% and 95% of the ideal performance, respectively. This approximately matches the memory sizing of the HPC systems dominating the current TOP500 list. For an HPC system with 1,000,000 cores, however, 7.6 GB and 16.1 GB of per-core memory would be needed to reach 90% and 95% of ideal performance, respectively. To increase efficiency to 99% of the ideal performance, a system with 100,000 cores would require 31.4 GB and a system with 1,000,000 cores would require 88.6 GB per core, a huge amount.

To put our estimation into perspective on Figure 4.3 we plot three systems: the 16-core system used in the first study that analyzed HPL memory capacity requirements [29], our largest experiment with 16,384 cores on MareNostrum supercomputer (year 2016), and a future potential 1,000,000-core system.[4] These points validate the model over more than a decade of system scaling and illustrate that larger systems will need more memory per core to achieve good HPL performance.

4.3.3 Mathematical analysis

The section complements the HPL analysis in Section 4.3.2; it presents step-by-step mathematical formulas that analyze HPL performance as a function of per-core memory capacity and the number of processes. The analysis indeed shows that the HPL performance analytically converges to a steady value proportional to the floating-point rate (GFLOP/s) of the system, but the performance optimum is theoretically reached for *infinite main memory.* We also calculate the per-core memory capacity needed to achieve steady values of HPL performance, and how this amount of memory changes when increasing the size of HPC systems (number of cores).

Number of FLOPs. HPL solves a dense linear system of N unknowns using LU factorization [87]. For a given problem size N the benchmark performs the following number of double-precision floating-point operations

[3]In the model developed in Section 4.3.3, α and β are constants independent of the number of cores.

[4]In this study, we consider only high-end x86 processing cores. Few HPC systems on the newest TOP500 list comprise more than a million cores [113], but instead of high-end x86 cores, these systems have either embedded or accelerating cores.

($\#FLOPs$) [29]:

$$\#FLOPs = \frac{2}{3}N^3 + 2N^2 + \mathcal{O}(N) \tag{4.1}$$

Since $N \gg 1$, and for number of processes n:

$$\#FLOPs_{\text{per\_process}} \simeq \frac{2N^3}{3n} \tag{4.2}$$

Execution time. HPL execution time on a specific system depends on various system parameters:

- γ_3: The time that a single processing unit (e.g. CPU core) requires to perform one floating-point operation when performing matrix-matrix operations.
- α: The time to prepare a message for transmission between processes.
- β: The time $L \times \beta$ indicates the time taken by the message of length L to traverse the network to the destination.

The execution time also depends on the way the data (matrices) are partitioned and distributed among the processes. The coefficient matrix is first logically partitioned into blocks, each of dimension $N_\text{B} \times N_\text{B}$, and these blocks are cyclically distributed onto the process grid. In all our experiments, factor N_B is kept constant. Finally, the data is distributed onto a two-dimensional grid of processes, $P \times Q$, where the total number of processes is $n = P \times Q$. When possible it is suggested to keep the same values for P and Q, i.e., $P = Q = \sqrt{n}$ [87].

An approximation of the HPL execution time T that illustrates the cost of the dominant factors is [87]:

$$T = \frac{2\gamma_3 N^3}{3PQ} + \frac{\beta N^2(3P+Q)}{2PQ} + \frac{\alpha N((N_\text{B}+1)\log P + P)}{N_\text{B}} \tag{4.3}$$

HPL performance. HPL performance is the number of FLOPs divided by the execution time, and is expressed in FLOP/s. In order to simplify our mathematical formulas, we observe *execution time per FLOP*, which is the reciprocal for HPL performance.

If we assume $P = Q = \sqrt{n}$ and analyze HPL execution time per FLOP by dividing (4.3) by (4.2), we have:

$$T_{\text{per\_FLOP}} = \gamma_3 + \frac{3\beta\sqrt{n}}{N} + \frac{3\alpha n\left(\frac{1}{2}(N_\text{B}+1)\log n + \sqrt{n}\right)}{2N_\text{B}N^2} \tag{4.4}$$

Equation 4.4 describes the HPL execution time per FLOP as a function of the problem size N. Per-core memory capacity m depends on the problem size N and the number of processes n: $m = \frac{10N^2}{n}$.[5] Therefore, the problem size that generates per-process memory capacity m when the HPL is executed on n processes can be computed as: $N = \sqrt{\frac{mn}{10}}$. If we put this into Equation 4.4, we determine the dependency between the time per FLOP ($T_{\text{per\_FLOP}}$) and the per-core memory capacity (m):

$$T_{\text{per\_FLOP}} = \gamma_3 + \frac{3\beta\sqrt{10}}{\sqrt{m}} + \frac{15\alpha\left(\frac{1}{2}(N_B+1)\log n + \sqrt{n}\right)}{N_B m} \tag{4.5}$$

We observe time per FLOP as a function of $\frac{1}{\sqrt{m}}$. For smaller values of per-process memory, the second and third terms in Equation 4.5 increase the execution time per FLOP, which means that communication overheads lower the HPL performance. For infinite memory, $T_{\text{per\_FLOP}} = \gamma_3$. This means that when increasing per-core memory, HPL execution time per FLOP, and therefore the HPL performance, indeed analytically converge to a steady value. This steady value is proportional to the number of processes because the number of FLOPs is also proportional to the number of cores (processes) used in the HPL run.

Reaching the steady execution time per FLOP. Next, we calculate per-core memory needed to achieve steady values of execution time per FLOP. Also, we analyze how this amount of memory changes when increasing the size of HPC systems (number of cores). Dividing (4.5) by γ_3 to normalize relative to the lowest execution time with infinite memory gives the relative execution time per FLOP:

$$T_{\text{rel\_per\_FLOP}} = 1 + k_1\frac{1}{\sqrt{m}} + \left(k_2\sqrt{n} + k_3\log n\right)\frac{1}{m} \tag{4.6}$$

Constants k_1, k_2, k_3 and γ_3 depend on the hardware platform, and for our platform we fit the constants according to the results from Figure 4.2. First we fit the constants k_2 and k_3, and then k_1 and γ_3 using linear regression. We get maximum error against experiments of 13%. Then, we analyze how much memory per core is needed to get close to ideal result with infinite memory, i.e., to reach a relative execution time per FLOP of ε. This we get by solving a quadratic equation:

$$\left(k_2\sqrt{n} + k_3\log n\right)\left(\frac{1}{\sqrt{m}}\right)^2 + k_1\left(\frac{1}{\sqrt{m}}\right) + (1-\varepsilon) = 0 \tag{4.7}$$

The results for ε values that contribute to 90%, 95% and 99% of ideal (infinite memory) HPL performance are explained in detail in Section 4.3.2. By taking the derivative of the solution to Equation 4.7, we find that, for

[5]Problem size should be set to 80% of available memory [87]. The $N \times N$ coefficient matrix requires $8N^2$ bytes, so the per-core memory capacity is $\frac{100}{80} \times \frac{8N^2}{n} = \frac{10N^2}{n}$.

any fixed target overhead $\varepsilon > 0$, increasing the number of cores n, always increases the memory per core, m. This shows that, as the total number of cores is increased, more memory per core is needed to achieve good execution time per FLOP, and therefore good HPL performance. This trend is confirmed by the experimental results in Section 4.3.1 and historical systems in Figure 4.3. Writing k_1, k_2 and k_3 in terms of α and β then taking the derivatives with respect to α and β shows that increasing interconnect latency or reducing interconnect bisection bandwidth also increase the memory per core for any fixed overhead $\epsilon > 0$.

4.4 High-Performance Conjugate Gradients

The High-Performance Conjugate Gradients (HPCG) benchmark [111] has been introduced as a complement to HPL and the TOP500 rankings, since the community questions whether HPL is a good proxy for production applications. HPCG is based on an iterative sparse-matrix conjugate gradient kernel with double-precision floating-point values, and is representative of HPC applications governed by differential equations. Such applications tend to have much greater demands on the memory system, in terms of bandwidth and latency, and they access data using irregular patterns [27]. Similarly to HPL, the user can scale the problem size to achieve the best performance on a given system. Therefore, to determine the capacity requirements, we analyze HPCG performance for a range of 16–8192 processes as a function of the problem size, i.e., main memory used in the experiment. Then, we analyze whether the trends detected in the real-system measurements match the expected tendencies based on algorithm complexity and data pattern accesses.

4.4.1 Measured memory requirements

In Figure 4.4, we show the relation between relative HPCG performance (Y-axis) and memory footprint (X-axis). We executed HPCG for various memory footprints, by changing the problem sizes from 24-24-24 up to 120-120-120 with an additive step of 8, always keeping the three dimensions identical, and reported average per-process memory footprints. For each number of processes, the performance was plotted relative to the results with the smallest problem size used in the experiments. We analyzed this trend for different numbers of processes that are plotted with different lines of the chart. Recall that, for each experiment, the number of processes equals the number of cores.

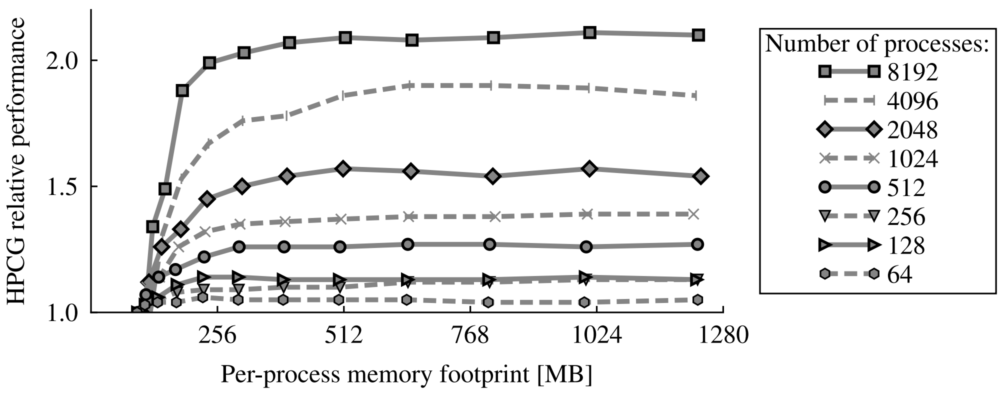


Figure 4.4: HPCG performance also depends on the available memory capacity. Performance increases until it reaches the saturation point, where it is constrained by the sustained memory bandwidth. The saturation point remains constant across a wide range of HPCG processes, at around 512 MB of main memory per process.

For a small number of processes and small input datasets, the HPCG workload may (partially) fit into on-CPU caches which leads to performance that significantly exceeds stable ones (on larger input datasets). In our experiments, we detected this trend for 16 and 32 processes. This trend is detected and analyzed by previous studies [63], and it is considered to be a non-representative use of HPCG [28], [111]. Therefore, we neither plot nor analyze these results.

As we increase the HPCG problem size, i.e., per-process memory footprint, HPCG performance rapidly increases and then reaches a stable value directly proportional to the number of processes used in the experiment.[6] Unlike HPL, we detect that the saturation point remains constant, roughly 512 MB of main memory per-process, for a large range of HPCG processes. In the following section, we analyze this in detail. Finally, we also detect that for memory footprints of around 1.5 GB, HPCG performance decreases. The performance drop is caused by swapping, and we did not detect it when the experiments were repeated on large-memory nodes comprising 8 GB of main memory per core. The HPCG performance drop because of memory swapping was not reported in past, and we would suggest that HPCG developers take this into account when providing suggestions about dataset sizing.

[6]As for HPL, this trend is not visible in Figure 4.4 because for each number of processes, the performance is normalized to the result with the smallest input dataset.

4.4.2 Analysis

Although the HPCG benchmark was released only a couple of years ago, several studies analyze its behavior and performance bottlenecks, and even estimate its performance on future exascale HPC systems [63], [82]. When running HPCG, the user sets the *per-process* problem size N. For a given problem size N, the number of floating-point operations and memory accesses are both proportional to N.

$$\#FLOPs_{\text{per\_process}} \approx \mathcal{O}(N) \tag{4.8}$$

The execution time of HPCG depends on the problem size N and number of processes n:

$$T \approx \mathcal{O}(N) + \mathcal{O}(N^{\frac{2}{3}}) + \mathcal{O}(\log n) \tag{4.9}$$

The first factor $\mathcal{O}(N)$ refers to the computational complexity, while factors $\mathcal{O}(N^{\frac{2}{3}})$ and $\mathcal{O}(\log n)$ refer to point-to-point and collective communication, respectively.

For small memory capacities, below 256 MB per process in Figure 4.4, the HPCG performance is affected by the interprocess communication, factors $\mathcal{O}(N^{\frac{2}{3}})$ and $\mathcal{O}(\log n)$ in Equation 4.9. However, as the input dataset increases, the factor $\mathcal{O}(N)$ becomes dominant and the communication overhead mitigates; the HPCG performance rapidly converges to the saturation point determined by the memory bandwidth.

For large per-process memory capacities (large values of N), the HPCG execution time is dominated by the computational routines, which mainly perform sparse matrix-vector multiplication [27] and require modest CPU power but significant memory bandwidth. For each floating-point operation (FLOP), HPCG requires a transfer of at least 4 bytes from main memory, i.e., HPCG byte-per-FLOP ratio is higher than 4. In state-of-the-art HPC systems, the byte-per-FLOP ratio is below 1,[7] meaning that in current systems memory bandwidth is the main performance bottleneck. As we increase the system size, the total available memory bandwidth and therefore HPCG performance increase proportionally. Because of this, HPCG performance is proportional to the number of processes, as detected in Section 4.4.1.

Finally, we analyze whether the HPCG performance saturation point moves as we increase the number of processes. As the number of processes increases, the factor $\mathcal{O}(\log n)$ might cause the HPCG saturation point to move towards the larger per-process memory capacity, i.e., towards the right

[7]Our node comprises two 8-core Sandy Bridge sockets and each core can execute up to 8 double-precision FLOPs per cycle. Each socket has four 64-bit wide, 1.6 GHz memory channels. Therefore byte/FLOP = $\frac{8\times(8 \text{ bytes})\times(1.6 \text{ GHz})}{16\times(8 \text{ FLOP})\times(3 \text{ GHz})} = 0.27$.

in Figure 4.4. Marjanović et al. [63] analyze in detail the impact of collective communication on HPCG performance, and conclude that for any plausible input dataset sizes and numbers of processes this impact is negligible. The authors also analyze the communication overheads in future systems, and estimate that even in a million-core HPC system, the communication overhead stays below 1.2%.

4.5 Production HPC applications

In this section, we analyze how the number of application processes affects the memory footprints of production HPC applications, taking into account application scalability, the targeted HPC category, and the size of the input dataset.

We study 10 of the 12 applications from the Unified European Application Benchmark Suite (UEABS) [89], which has been designed to represent production applications running on large-scale Tier-1 and Tier-0 HPC systems in Europe. Each application is detailed explained in Section 3.2.2. Table 4.1 summarizes the applications and input datasets. For each application, we briefly describe the input dataset, and indicate the number of processes used in the experiments. As for HPL and HPCG, 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. Some of the applications have memory capacity requirements that exceed the available memory on a single node, which limits the lowest number of processes we use in the experiments, e.g., BQCD in Test Case A cannot be executed with less than 64 processes (four nodes). The largest number of processes we use is 8192, except for Quantum Espresso (QE), which reports errors when executing on 4096 or 8192 cores. Note that SPECFEM3D always runs with the specified numbers of cores: 864 in Test Case A and 11,616 in Test Case B.

4.5.1 Memory footprint vs. Number of processes

The memory footprints of an HPC application executed with a given input dataset can vary significantly for different numbers of application processes. In general, the more processes used for the computation, the smaller the portion of the input data handled by each process. On the other hand, distributing the computation over a larger number of processes also means more replication of data on the boundaries of adjacent data segments, or in per-process private data segments, external libraries, and communication buffers [50]. Although it may seem obvious that memory footprints

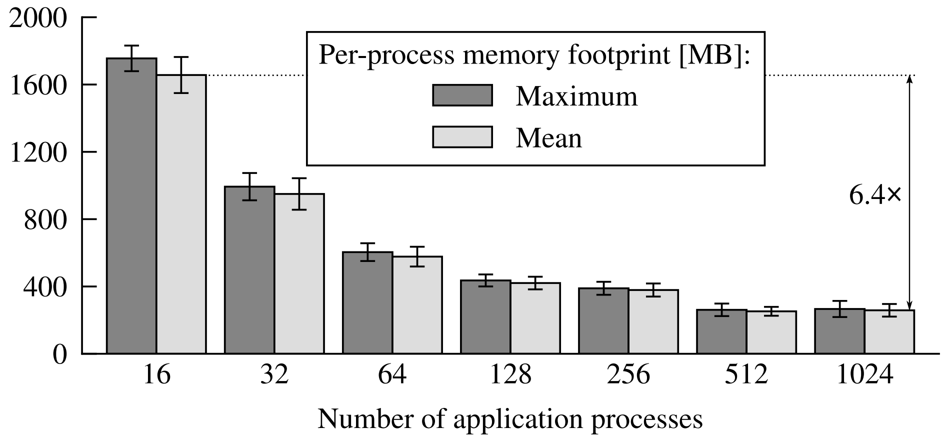
Table 4.1: Scientific HPC applications used in the study

Application	Test Case A: Smaller input dataset		Test Case B: Larger input dataset	
	Problem size	Process range	Problem size	Process range
ALYA	27 million element mesh	16–1k	552.9 million element mesh	256–8k
BQCD[a]	$32^2\times 64^2$ lattice	64–8k	N/A	N/A
CP2K	Energy calculation of 1024 waters	128–1k	216 LiH system with Hartree-Fock exchange	∅[b]
GADGET	135 million particles	512–8k	N/A	N/A
GENE	Ion-scale turbulence in Asdex-Upgrade	64–1k	Ion-scale turbulence in Jet	2k–8k
GROMACS	150,000 atoms	16–1k	3.3 million atoms	16–8k
NAMD	2×2×2 replication of the STM Virus	16–1k	4×4×4 replication of the STM Virus	64–8k
NEMO	12° global configuration; 4322×3059 grid	512–8k	N/A	N/A
QE	112 atoms; 21 iterations	16–1k	1532 atoms; two iterations	1k–2k
SPECFEM3D	6×12×768 mesh of the earth	864	6×24×1760 mesh of the earth	11,616

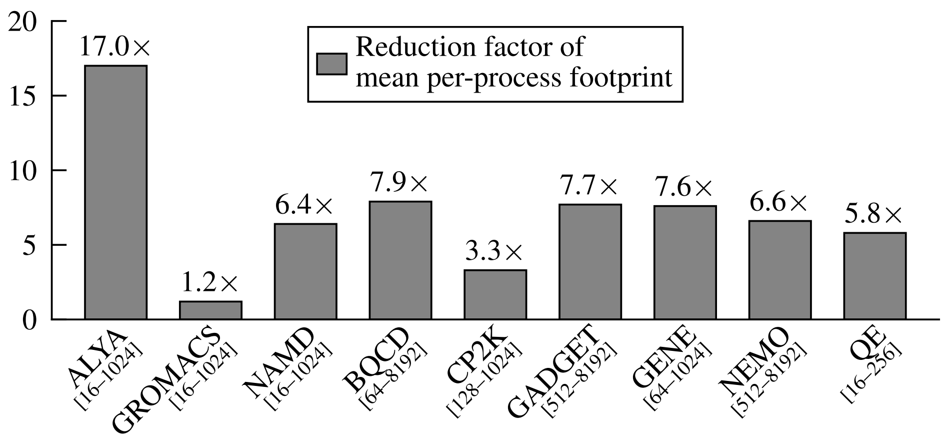
[a] Quantum Chromo-Dynamics (QCD) is a set of five kernels. We study Kernel A, also called Berlin Quantum Chromo-Dynamics (BQCD), which is commonly used in QCD simulations.
[b] CP2K cannot run Test Case B on our platform. The errors have been reported to the application developers.

of HPC applications are tightly-coupled with the number of application processes, previous studies of memory footprints *ignore this relationship* (see Section 4.7).

Figure 4.5a illustrates the relationship between the per-process (per-core) memory footprint and the number of application processes for NAMD running Test Case A. This is a strong scaling case, i.e, we keep the same input dataset (Test Case A) and change the number of processes. This corresponds to a real-life use of production applications in which users have to choose the number of processes that will be used to solve an already defined problem with a fixed input size. For each process we track the maximum and mean memory footprints, and we plot the average values and the standard deviations among all processes. When NAMD runs as 16 processes, the mean per-process memory footprint is 1656 MB. As we increase the number



(a) NAMD, Test Case A. Mean and maximum memory footprints exhibit the same trend.



(b) UEABS applications, Test Case A. The range of processes is indicated below each benchmark.

Figure 4.5: Per-process memory footprints shrink as the number of processes increases.

of processes, the footprint drops significantly. When the application runs using 1024 processes, the per-process memory footprint is only 258 MB, a difference of 6.4×. Maximum memory footprints exhibit the same trends (see Figure 4.5a), and thus in the rest of the chapter we discuss only mean footprint values.

Figure 4.5b summarizes memory footprint results for all UEABS workloads. Except GROMACS, which has a small overall memory footprint,[8] the general trend is the same for the remaining applications. We detect memory footprint changes from 3.3× for CP2K up to 17× for ALYA.

Discussion

Our analysis emphasizes that the memory footprints of HPC applications are tightly-coupled with the number of application processes. State-of-the-art parallel benchmark suites, however, do not strictly define the number of processes to use in experiments. UEABS recommends experiments with up to 10,000 processes, but the minimum number of processes is not specified. Similarly, other parallel benchmark suites either provide loose recommendations about the number of processes (SPEC OMP2012 [2], SPEC MPI2007 [1], SPLASH-2 [118]) or do not discuss this issue at all (NAS [116], PARSEC [14], HPC Challenge [60], Berkeley dwarfs [4]). Therefore, when analyzing memory capacity requirements, it is essential that the users themselves determine a number of processes that is representative of real production use. This, in turn, requires knowledge of the HPC category that the user is targeting together an understanding of the scalability of the applications under study, as we discuss in the following sections.

4.5.2 Selecting the number of processes

HPC categories

High-performance computing is broadly divided into two categories [32]. *Capability computing* refers to using a large-scale HPC installation to solve a single problem in the shortest possible time, for example simulating a human brain on a Tier-0 HPC system. *Capacity computing* refers to optimizing system efficiency to solve as many mid-size or smaller problems as possible at the same time at the lowest possible cost, for example when small or medium enterprises use rented (on-demand) HPC resources to simulate numerous design choices for their products. Analyzing pricing policies for renting HPC resources is beyond the scope of this study; in the rest of this chapter, we therefore approximate the *cost* of a given experiment as proportional to the CPU-hours, i.e., the number of cores used in the experiment (*#cores*) multiplied by the execution time: $cost \propto CPU\_hours = \#cores \times exe\_time$.

[8]The GROMACS developers explain that the application requires only around 100 MB in total for Test Case A (divided among all processes). The dominant part of the per-process GROMACS memory footprints comes from the MPI library and other external libraries, and it remains constant as the number of application processes increases.

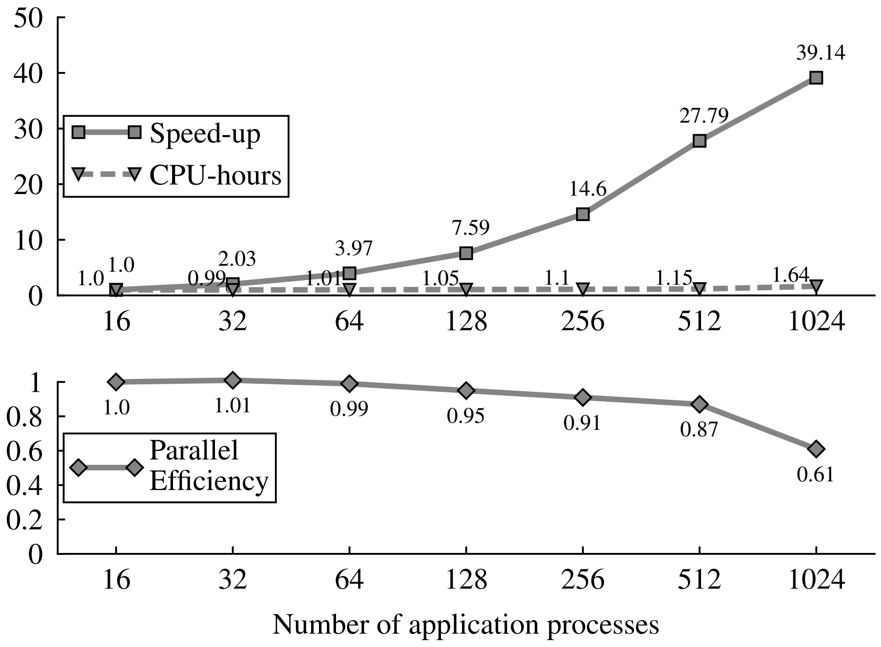
Although capability computing targets application runs with the lowest execution time, excessive application scaling may deliver diminishing returns in performance improvement while linearly increasing CPU-hours. This is an unacceptable scenario that leads to inefficient resource utilization. Similarly, although capacity computing targets low-cost HPC computation, excessive slowdown of application runs may have unacceptable impact, e.g., on the productivity of engineers waiting for simulation results.

Application scalability

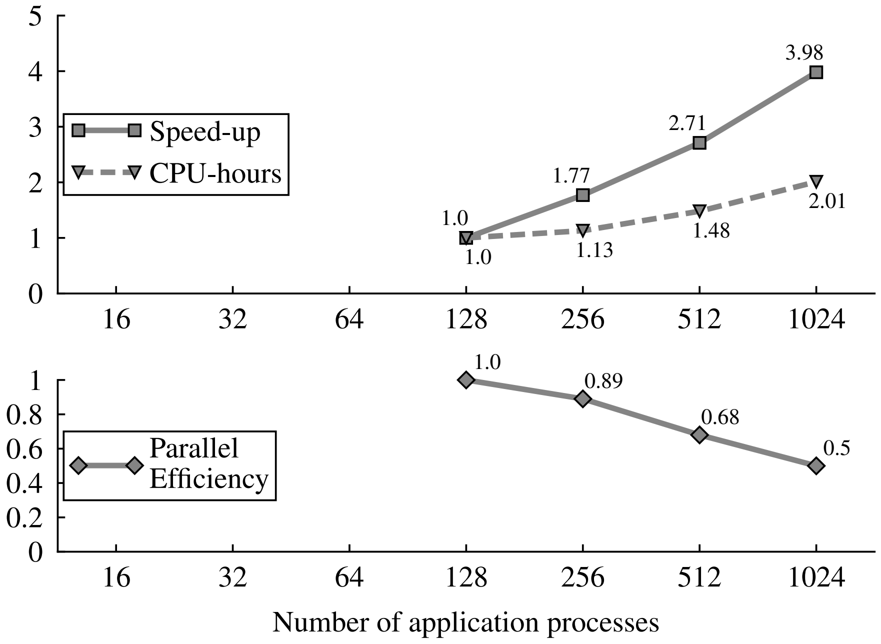
It is important to understand that CPU-hours and execution time are dependent metrics, and that in the production runs, users must analyze the trade-offs between them. In Figure 4.6, we analyze this relationship for NAMD and CP2K, respectively. The upper graphs of Figures 4.6a and 4.6b show normalized speed-up and CPU-hours for each experiment, and the lower graphs show application parallel efficiency. All statistics are computed relative to the experiments with the fewest processes, 16 for NAMD and 128 for CP2K.[9] Parallel efficiency (a number between 0 and 1) quantifies how effectively the resources are utilized, and it is the main metric for analyzing application scalability. A parallel efficiency of 1 means that the application speed-up is directly proportional to the number of processes. Low parallel efficiency means that significantly increasing processing resources only delivers low or moderate speed-ups.

NAMD is an example of an application with good scalability (Figure 4.6a). Increasing the number of processes causes significant speed-ups with negligible increments in CPU-hours. When we change the number of processes from 16 to 256, 512, and 1024, we measure speed-ups of 14.60×, 27.79×, and 39.14× at cost increments of 10%, 15%, and 64%, respectively. When used in capability computing, NAMD should be executed with a large number of processes (we use 1024 processes in the experiments presented in Figure 4.6a). Although CPU-hours is the primary metric in capacity computing, it is reasonable to expect that a user would accept small increases in CPU-hours if they lead to high improvements in execution time. Therefore, in capacity computing as well, experiments with a large number of processes are most representative of real-life production use of NAMD. We observe similar trends for ALYA, BQCD, GENE, and Quantum Espresso. All of them show good scalability and significant speed-ups with negligible CPU-hours increments. In both capability and capacity computing, these applications should be executed with many processes.

[9]Recall that CP2K cannot be executed with 16, 32 and 64 processes because its memory requirements exceed the available memory.



(a) NAMD (Test Case A), good scalability



(b) CP2K (Test Case A), limited scalability

Figure 4.6: Trade-offs between normalized execution time and experiment cost (CPU-hours) for applications with good and limited scalability.

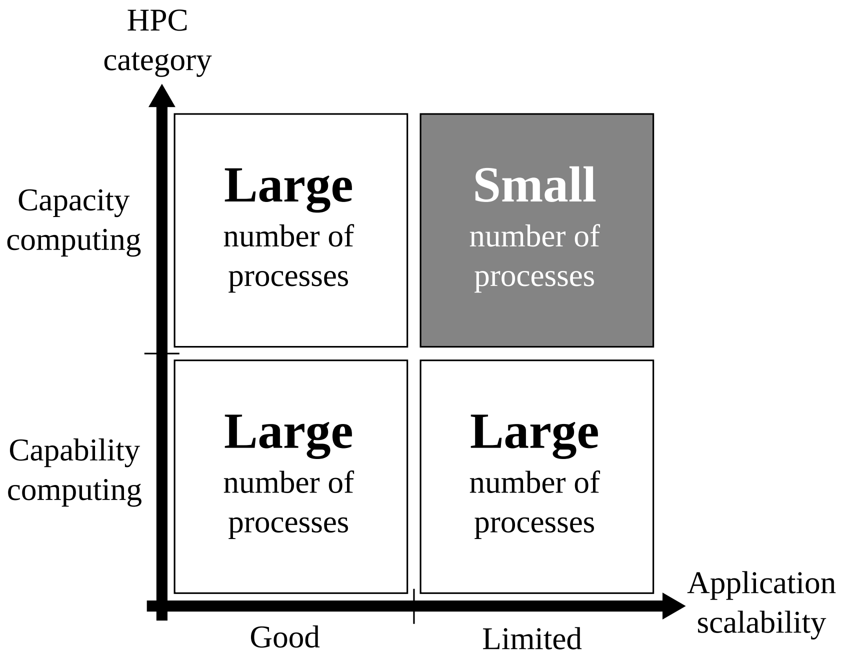

HPC category
Capacity computing
Capability computing
Large number of processes
Small number of processes
Large number of processes
Large number of processes
Application scalability
Good
Limited

Figure 4.7: The representative number of application processes is determined by application scalability and the targeted HPC category.

CP2K is an example of an application with limited scalability (Figure 4.6b). When we change the number of processes from 128 to 256, 512, and 1024, we observe speed-ups of 1.77×, 2.71×, and 3.98×, while the CPU-hours increase 1.13×, 1.48×, and 2.01×, respectively. Results for CP2K show clear trade-offs between cost and speed-up. When used in capability computing, CP2K should be executed with a large number of processes, 512 and 1024 in the experiments presented in Figure 4.6b. When targeting capacity computing, users should try to reduce CPU-hours. Thus, CP2K should be partitioned into smaller numbers of processes. We observe similar behavior for GADGET, GROMACS, and NEMO.

Summary

In this section, we showed how memory footprints of HPC applications depend on the number of application processes, and we provided guidelines for selecting the number of processes to be representative of production application use. Figure 4.7 summarizes this analysis. The figure shows how the representative number of application processes depends on application scalability and how this may change for different HPC categories. Applications with good scalability should be executed with large numbers of processes, regardless of the targeted HPC category. This leads to significant speed-ups with only a small increase in experimentation cost. For applications with limited scalability, increasing the number of processes reveals a clear trade-off between execution time and CPU-hours. For experiments that target capability computing, these applications should be executed with a large number of processes providing low execution time at the expense of the CPU-hours. On the other hand, when targeting capacity computing, applications should be partitioned into a small number of processes, sacrificing execution time to improve experimentation cost and overall system throughput [122].

4.5.3 Memory requirements of production HPC applications

In this section, we analyze per-process memory capacity requirements of the production applications. In all experiments, the applications were run with Test Case A inputs and up to 8192 processes. Test Case A can be run for all UEABS applications, and it supports a wider range of processes compared to Test Case B (which could only run six out of ten applications, see Section 4.2.2). We analyze Test Case B in more detail in Section 4.5.4. Recall that our applications roughly scale up to 1000–10,000 processes when running these input datasets.

The results are summarized in Figure 4.8. The left side of Figure 4.8 shows results for the applications with good scalability — ALYA, BQCD, GENE, NAMD, and Quantum Espresso (QE). These applications should be executed with a large number of processes regardless of the targeted HPC category. The average per-process memory footprints for these applications ranges from 57 MB for ALYA to 258 MB for NAMD. The footprints for BQCD, GENE, and QE are 116 MB, 137 MB, and 197 MB, respectively.[10]

The right side of Figure 4.8 shows the per-process memory footprints of the applications with limited scalability. In the capability computing experiments, we execute these applications on a large number of cores: 1024 for CP2K, and 8192 for GADGET and NEMO. The per-process memory footprints are again fairly small — 336 MB, 154 MB, and 203 MB, for CP2K, GADGET, and NEMO, respectively. Since the processor under study has eight cores, and we allocate one process per core, the per-socket memory footprint in these experiments ranges between 0.4 GB (ALYA, 8×57 MB)

[10]Although Quantum Espresso processing Test Case A should scale up to 1024 processes [89], we observed very good scalability up to 256 processes but poor scalability (slowdowns) for 512 and 1024 processes. We therefore report results for 256 processes for this application.

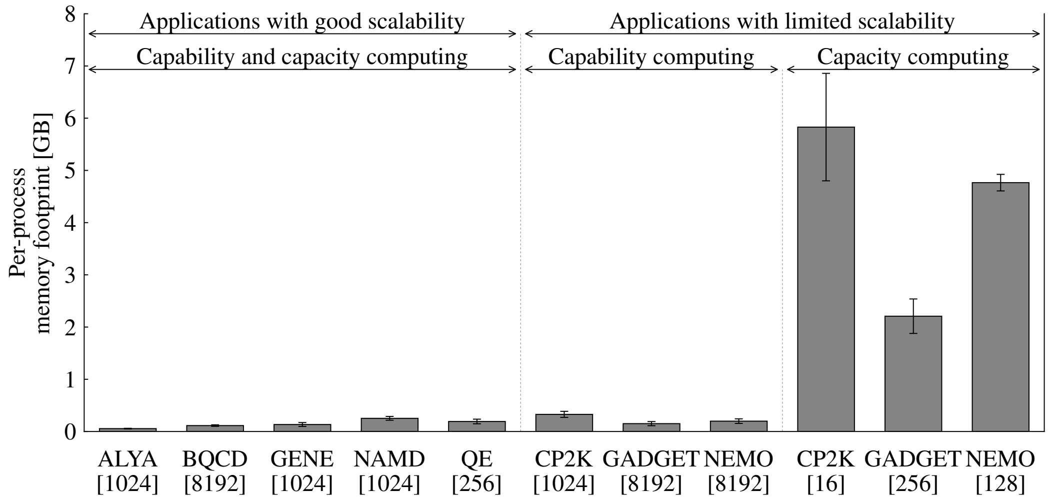


Figure 4.8: Memory footprints of production HPC applications depend on application scalability and the targeted HPC category. Only the applications with limited scalability that target capacity computing require gigabytes of main memory per process.

and 2.6 GB (CP2K, 8×336 MB). The first-generation 3D memory devices already provide such memory capacities, see Chapter 2.

In capacity computing, we execute the applications with limited scalability on a small number of cores, and this number is dictated by the memory capacity of the compute nodes — scaling the parallelism down further would cause the per-process memory footprints to exceed the available memory. To understand how the memory footprints increase in systems with higher memory capacities, we run experiments on large-memory nodes containing 128 GB of main memory, i.e., 8 GB per core. When we partition CP2K, GADGET, and NEMO to 16, 256, and 128 processes, their per-process footprints are 5.8 GB, 2.2 GB, and 4.8 GB, respectively (rightmost part of Figure 4.8). On standard nodes with 2 GB of memory per core, we could partition these applications to 128 processes for CP2K, and 512 processes for GADGET and NEMO. In this scenario, we measured the per-process memory footprints of 1.1 GB, 1.2 GB, and 1.3 GB, for CP2K, GADGET, and NEMO, respectively.

Figure 4.8 omits results for GROMACS and SPECFEM3D. The per-process footprint of GROMACS is very low, between 60 MB and 70 MB, and it decreases only slightly from 16 to 1024 processes (see Section 4.5.1). SPECFEM3D requires exactly 864 processes, and thus we cannot analyze how its memory footprint changes with the number of processes. When executed with 864 processes, the average memory footprint is 2.53 GB.

These results show that different production HPC applications — or even

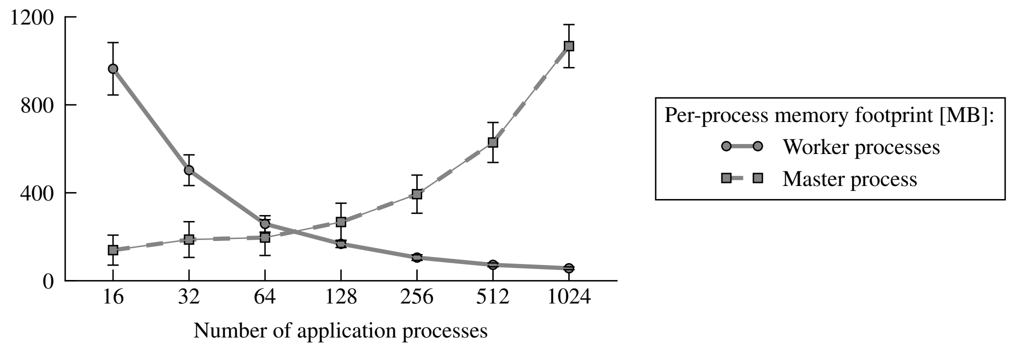


Figure 4.9: As the number of processes increase, memory footprints of worker processes decrease as expected, but memory footprint of the master process increases: ALYA, Test Case A.

a single application used in different HPC categories — can have significantly different memory capacity requirements. Applications that scale well and those that target capability computing have low per-process memory footprints. These applications require from 57 MB to 258 MB of memory, which means that they heavily under-utilize the memory capacity of our HPC platform: their average memory usage is below 10% of the full capacity. Only the applications with limited scalability that target capacity computing require gigabytes of main memory per process.

Master process memory requirements

Many HPC applications are written using a master–worker process model. In these applications, the problem is decomposed into data segments that can be processed independently by different *worker processes*. The *master process* (usually the first process) assigns work to each worker process, and collects the intermediate and final results of the computation.

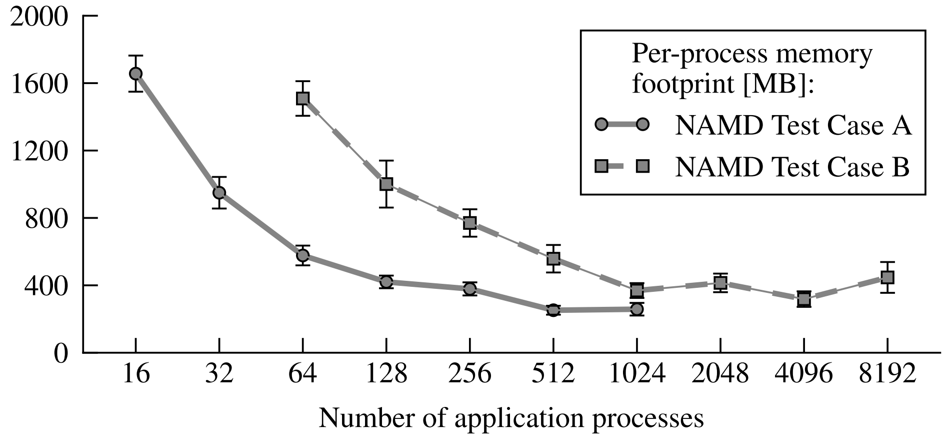
Figure 4.9 plots the memory footprints of the ALYA master and worker processes as the number of processes increases from 16 to 1024. Memory footprints of worker processes drop as we increase the number of processes, following the trend described in Section 4.5.1. The master process, however, exhibits the opposite trend, and its memory footprint increases with the number of processes. This is a general trend in master–worker applications. For the UEABS applications, we detect that master process may have significantly higher memory footprints than workers, up to 36.6× for NEMO (master 7.2 GB, worker 0.2 GB) and 62.5× for BQCD (master 7.1 GB, worker 0.1 GB). Both application developers and computer architects should pay attention to this phenomenon, still not well highlighted and quantified by the community.

4.5.4 Towards weak scaling analysis

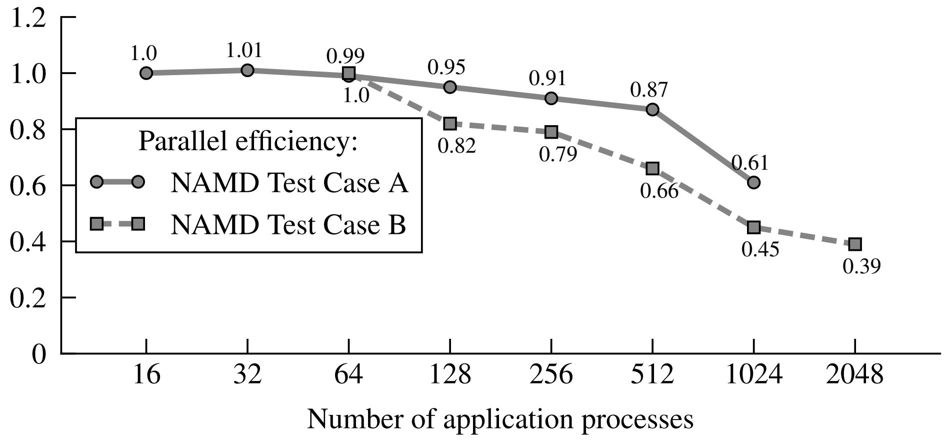
The presented analysis keeps constant the input dataset size and varies the number of application processes. This refers to the strong scaling case of production HPC applications. In addition to this, it is also important to perform a weak scaling analysis, i.e., to analyze memory capacity requirements when both the number of processes and input dataset size are increased — similar to the study performed for HPL and HPCG benchmarks. Since the problem inputs are specified by the benchmark suite, such analysis requires either that the benchmark suite support a user-defined problem size (as for HPL and HPCG) or that it provides a set of inputs specifically intended for weak scaling analysis. We are not aware of such a real application benchmark suite. UEABS for instance has just two problem sizes, Test Case A and Test Case B, and in many cases the problems being solved are fundamentally different, making them unsuitable for weak scaling analysis. For example, in case of ALYA, Test Case A is a model of the respiratory system whereas Test Case B is a mesh of generic elements [17]. As an intermediate step, we analyze the two input datasets for the NAMD benchmark distributed with the UEABS suite, which is one of few benchmarks where the two datasets are comparable [17], and observe the changes in the application memory footprint and scalability when increasing the input dataset size.

In order to analyze how the per-process memory footprint changes with dataset size, in Figure 4.10 we plot the NAMD memory footprint results for both the smaller and larger input datasets. The Test Case A curve starts at 16 processes, a single MareNostrum node. Test Case B exceeds the memory capacity of one or two MareNostrum nodes (16 and 32 processes), so the curve starts from 64 processes. We show the Test Case A results on up to 1024 processes, and for Test Case B on up to 8192 processes, as recommended by UEABS documentation [89]. For both input datasets, the NAMD footprint is around 1600 MB on a small number of processes, and it drops rapidly as we increase the process count. Both memory footprint curves follow the same tendency, with the Test Case B results being shifted towards larger numbers of processes. We detect the same trend for all UEABS applications.

Next, we analyze the impact of dataset size on application scalability. Figure 4.10b plots parallel efficiency of NAMD with both input datasets. Parallel efficiency of NAMD with Test Case A reduces to 0.61 when the process count increases from 16 to 1024 (64×). With Test Case B, when increasing from 64 to 2048 processes (32×), the parallel efficiency drops to



(a) When input dataset increases memory footprint curve shifts towards larger number of processes.



(b) Increasing the input dataset lowers the scalability of NAMD application.

Figure 4.10: Increasing the input dataset changes memory footprint and scalability of NAMD application. We detect the same trend for all UEABS applications.

0.39.[11] For all the applications under study, we find that it is harder to achieve good scalability for larger numbers of processes, even if the input dataset size increases. This is not surprising, because increasing the number of processes causes more communication and synchronization overheads, and increases the penalty of sequential code segments. The simple increase in

[11]NAMD running Test Case B should scale up to 10,000 processes, but we detect very low or no speed-up after 2048 processes. Therefore we plot parallel efficiency results up to 2048 processes for Test Case B.

dataset size does not address all of these problems.

To summarize, our results show that when input datasets increase, the memory footprint as a function of the number of processes keeps the same trend, but the curve is shifted toward larger number of processes (Figure 4.10). Application scalability, however, reduces, in some cases significantly. Therefore, increasing the input dataset requires repeating the analysis of the trade-offs between execution time and CPU-hours (Section 4.5.2) in order to determine the representative number of processes for a production run of the application.

4.6 Implications

The current trend in HPC system design is to increase the number of memory channels per CPU and the number of I/Os in each DDR generation. This approach is limited by the package size, plus it is expensive, and it increases memory system power consumption. A potentially promising solution for these problems is the use of 3D-stacked DRAMs. Detailed overview of the currently-available products based on this technology: Hybrid Memory Cube (HMC), High-Bandwidth Memory (HBM), and multi-channel DRAM (MCDRAM) incorporated into the Knights Landing processors is given in Chapter 2. In this section, we summarize the pros and the cons of 3D-stacked DRAM. We also discuss the opportunities and challenges of these solutions in the context of high-performance computing, and outline our expectations on how these devices may change the design of next-generation memory systems.

4.6.1 3D-stacked DRAM in HPC memory systems: Opportunities and challenges

Understanding application memory capacity requirements is essential for the design of HPC memory systems based on 3D-stacked DRAM. The main question to be answered is whether 3D memory chiplets can on their own provide the capacity required by HPC applications.

HPCG vs. HPL

An important driving force for 3D-stacked DRAM could be the HPCG benchmark. With performance directly proportional to main memory bandwidth, and memory footprints below 1 GB per process even when targeting million-core systems, HPCG could be the first success story for 3D-stacked DRAM in HPC. Regarding KNL, hundreds of MBs per core of MCDRAM may be sufficient for an outstanding HPCG performance [48], especially on small clusters (see Figure 4.4). Therefore, HPCG could be a good example of an important benchmark that works out-of-the-box and performs well on KNL.

On the other hand, one of the main show-stoppers for 3D-stacked memory could be HPL. In contrast to HPCG, high memory bandwidth provides no benefits for HPL, while the limited capacity of 3D-stacked memory can lead to significant performance loss, especially in large-scale systems. As shown in Section 4.3.2, a million-core system would require 16.1 GB per core to achieve 95% of the potential performance (Figure 4.3). Although the HPC community is questioning whether HPL is representative of modern production HPC applications [72], [73], and is actively looking for alternative benchmarks (HPCG being one of them), high HPL scores are still important objectives in the design of large HPC clusters. Looking forward, it will be interesting to see whether KNL-based TOP500 systems will use the 3D-stacked MCDRAM in the HPL runs, i.e. whether the developers of optimized HPL and corresponding linear algebra libraries will find a way to benefit from hybrid MCDRAM + DDR memory systems.

Production HPC applications

In Section 4.5.3 and Figure 4.8 we saw that the memory capacity requirements of production HPC applications had a bimodal distribution. Most of our HPC applications and use cases require only hundreds of megabytes of main memory. This capacity can be provided by 3D memory chiplets located on the silicon interposer (e.g. KNL MCDRAM), with no need for conventional DIMMs on the printed circuit board (PCB). Such a memory will provide significantly higher memory bandwidth and lower latency, which, in turn, will lead to higher system performance and energy-efficiency. Since 3D-stacked memory chiplets could directly replace DIMMs, the main memory would still comprise a single level with uniform latency.

For HPC applications that require gigabytes of main memory, sufficient memory capacity can be provided using hybrid 3D memories plus on-PCB DIMMs, similar to KNL systems. The main memory would therefore consist of two levels of hierarchy with different latencies, bandwidths, and capacities. For computer architects, this opens design options to optimize the capacities, organizations, and interconnections of the 3D memory chiplets and the DIMMs.

Although hybrid memory systems support functional portability, i.e. execution of legacy codes, there is a clear tradeoff between the achieved performance and the effort invested in code profiling and development. For example, in the KNL cache mode, large-footprint applications can be executed with no changes in the source code, but this approach could lead to significant performance loss. Although having large caches may intuitively suggest higher performance, in KNL it may not be the case. Since MC-DRAM and DDR4 have separate data paths (as they use separate memory controllers), MCDRAM misses require two consecutive accesses — first to the MCDRAM and second to the DDR — leading to a high cache miss penalty and potentially low overall performance.

Good performance of hybrid memory systems is conditioned by the need for advanced data allocation, migration, and prefetching policies [20], [69]. Optimal data management in these systems, such as the KNL flat memory mode, requires profound application profiling and a significant increase in the code development cost. In order to reduce this effort and increase adoption of the KNL architecture, Intel released various profiling tools [41] and data management libraries and APIs [42] that simplify efficient programing of the systems with hybrid main memory. It will be interesting to see whether the KNL — as the first system that combines the 3D-stacked and the DDR main memory — will be adopted by the users, and whether the increased cost in software development and maintenance will be justified by the performance gains.

Message from application developers

New HPC systems should be designed taking into consideration the requirements from future users and application developers. With regard to memory capacity, certain applications, in domains such as theoretical physics and inorganic chemistry, have a constant need for more memory. Based on Figure 4.8, however, we see that there are many applications that have low memory requirements. Users of such applications that could in fact live with less DRAM usually remain quiet, since for them the additional DRAM under discussion will not degrade performance. Moreover, these users generally do not have to pay the costs associated with extra memory per core, in terms of capital cost and power consumption. This leads to general-purpose HPC clusters with 2 GB per high-end x86 core, with some large-memory nodes having 4 GB to 8 GB per core, i.e., more than 100 GB per node.

With the introduction of 3D-stacked memory, this dynamic changes. Users may wish to "trade" DIMM capacity, which they do not need, for 3D-stacked DRAM, which provides higher bandwidth and lower latency. For applications with relatively low memory capacity requirements, 3D-stacked DRAM is likely to lead to significantly better overall performance [91]. It is therefore essential that application developers understand the performance–

capacity–cost tradeoffs between DIMM-based, 3D-stacked and hybrid DRAM solutions, in order to clearly express their preferences to the HPC hosting centres. Whereas user demand has already led to large-memory nodes, messages from the users and developers of small-memory application may lead to specialization in the other direction; i.e., small-memory nodes with high-bandwidth low-latency 3D-stacked memory.

4.7 Related work

Dongarra et al. [29] present the Linpack benchmarks suite, the TOP500 list, and the HPL code. The authors execute HPL on a small 4×4 cluster of Pentium III 500 MHz CPUs and analyze the benchmark performance for various interconnects and input dataset sizes of up to around 250 MB per core. The results show that increasing the HPL input dataset size can lead to significant performance improvements. Our work extends this study in various directions. We detect the point of diminishing returns when increasing the input dataset size, and analyze how this changes with the number of processes used in the HPL run, i.e., with the size of the HPC system. We also estimate the amounts of physical memory required for the close-to-optimal HPL performance on future large-scale HPC clusters.

The HPL benchmark has been extensively used in the past. In general, HPL studies analyze how to tune arithmetic libraries, OS kernel and network parameters to improve HPL performance on a given system. The studies use the maximum input dataset that fits into the physical memory while preventing swapping, as suggested by the HPL developers, and do not analyze the impact of changes in the physical memory capacity on the HPL performance.

Marjanović et al. [63] analyze the HPCG benchmark and predict the HPCG performance on a given architecture based on the memory bandwidth and the highest network latency between compute units. They conclude that for modern systems with a decent network, highly accurate prediction can be done based only on the memory bandwidth. On the node level, they show that small problem sizes that fit in the CPU caches can have HPCG performance that exceeds the stable values and are therefore non representative. However, they do not analyze how HPCG performance depends on the problem size for larger numbers of processes, as we did in this study.

Although memory provisioning for large-scale HPC clusters is an important task, to the best of our knowledge only three prior studies analyze memory footprints of HPC applications [15], [86], [83]. However, these studies do not analyze the relationship to the number of processes, which is very important as we show in this study. Biswas et al. [15] and Perks et al. [86]

investigate different techniques to reduce memory footprints in order to improve the performance of HPC workloads. Biswas et al. [15] leverage the data similarity often exhibited in MPI applications. They identify identical memory blocks across MPI tasks on a single node and use a novel memory allocation library to merge them. The authors evaluate their proposal on a range of MPI applications (SPEC MPI2007, NAS, ASC Sequoia benchmarks, and two production applications), and show memory footprint reduction of 32% on average. Perks et al. [86] investigate the impact of compiler choice on the memory usage of distributed MPI codes. The authors compare memory usage of four versions of simple MPI benchmarks compiled with GNU, Intel, PGI, and Sun compilers. Their results show that compiler choice can make a difference of up to 32% in memory usage. Pavlovic et al. [83] characterize memory behavior for four scientific applications to estimate the memory system requirements of future HPC systems with hundreds or thousands of cores per node. The authors estimate memory footprints of HPC applications comprising thousands of processes by using linear regression based on results of a few experiments with a small number of processes. Even though the authors target systems running applications with thousands of processes, the study does not analyze application scalability, nor does it evaluate whether the input sets used in the study are large enough to take advantage of such parallelism.

As the first 3D-stacked DRAM devices are hitting the market, various studies analyze how to incorporate these devices into the memory hierarchy. It is generally accepted that 3D-stacked DRAM is unlikely to fulfill the memory capacity requirements of server and HPC applications, so the community is exploring hybrid systems in which 3D-stacked DRAM is complemented by standard DIMMs [25], [20], [100], [69]. The essence of these studies is the development of techniques for advanced data migration between 3D-stacked DRAM and DIMMs. An important requirement of this work is to avoid excessive code development costs and improve performance of legacy codes. So, all the studies keep the unified view of the main memory at the application level; the data management policies are performed in hardware by complex data path enhancements [20], [100] or by an interaction between hardware and the operating system [25], [69]. Overall, all studies agree that managing hybrid memory systems with 3D and DIMMs is a difficult task and that simple approaches, such as using 3D-memory as an additional level of cache, may lead to significant performance loss.

4.8 Summary

This study analyzed memory capacity requirements of important HPC benchmarks and applications. This analysis becomes increasingly important as 3D-stacked memories are hitting the market. These novel memories provide significantly higher memory bandwidth and lower latency, leading to higher performance and better energy-efficiency. However, the adoption of 3D memories in the HPC domain requires use cases needing much less memory capacity than currently provisioned. With good out-of-the-box performance, these use cases would be the first success stories for these memory systems, and could be an important driving force for their further adoption.

We detected that HPCG could be an important success story for 3D-stacked memories in HPC. With low memory footprints and performance directly proportional to the available memory bandwidth this benchmark is a perfect fit for memory systems based on 3D chiplets. HPL, however, could be one of the main show-stoppers because reaching a good performance requires memory capacities that are unlikely to be provided by 3D chiplets.

The study also emphasizes that the analysis of memory footprints of production HPC applications requires an understanding of their scalability and target category, i.e., whether the workloads represent capability or capacity computing. The results show that most of the HPC applications under study have per-core memory footprints in the range of hundreds of megabytes — an order of magnitude less than the main memory available in the state-of-the-art HPC systems; but we also detect applications and use cases that still require gigabytes of main memory.

Overall, the study indeed identified the HPC applications and use cases with memory footprints that could be provided by 3D-stacked memory chiplets, making the first step towards adoption of this novel technology in the HPC domain. Also, it showed that the simple question *"How much memory do we need in HPC?"* may not have a simple answer. We hope that this will motivate the community to question the trends for memory system sizing in current HPC clusters, and will lead to further analysis targeting future ones.

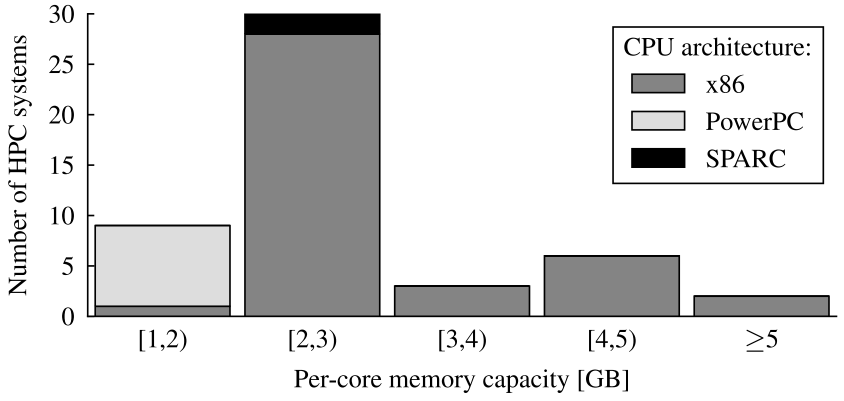
CHAPTER 5

Large-memory nodes for energy efficient HPC

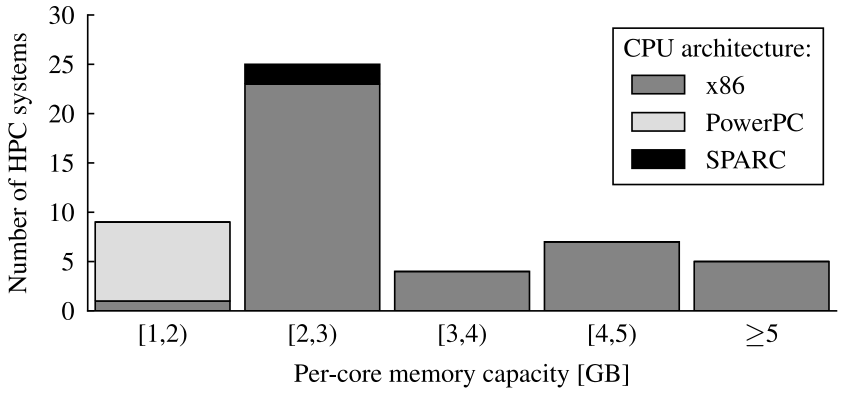
In recent years, we witness the trend of increasing number of HPC systems that comprise more than 4 GB of memory per core. Figure 5.1 shows per-core memory capacities of the top 50 HPC systems from the TOP500 list from years 2014 (Figure 5.1a), 2015 (Figure 5.1b) and 2016 (Figure 5.1c). We can see that each year, the number of HPC systems comprising more that 4 GB of memory per-core is increasing. Moreover, even the systems that comprise less than 4 GB of memory per core still include some percentage of compute nodes with increased memory capacity to satisfy the needs of some users.

In Chapter 4, we saw that applications that require gigabytes of main memory, have limited scalability and target capacity computing in HPC, i.e. they target solving as many mid-size or smaller problems as possible at the lowest possible cost. In this chapter we go a step further, and for this type of applications we investigate whether we can benefit in terms of energy efficiency from having large memory capacity on the node.

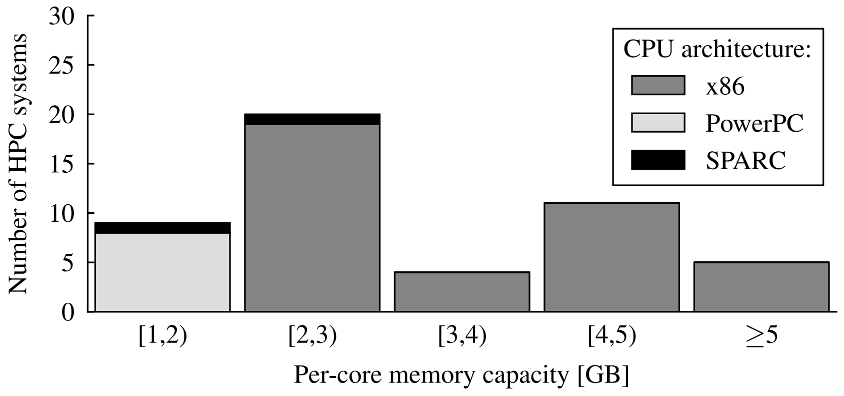
In this chapter, we investigate the potential for saving energy in high-performance computing (HPC) through scaling-in on large-memory nodes. Energy consumption is by far the most important contributor to HPC cluster operational costs, and it accounts for a significant share of the total cost of ownership. Advanced energy-saving techniques in HPC components have received significant research and development effort, but a simple measure that can dramatically reduce energy consumption is often overlooked. We show that, in capacity computing, where many small to medium-sized jobs have to be solved at the lowest cost, a practical energy-saving approach is to *scale-in* the application on large-memory nodes. We evaluate scaling-



(a) Top 500 list - November 2014.



(b) Top 500 list - June 2015.



(c) Top 500 list - June 2016.

Figure 5.1: Evolution of per-core memory capacity of HPC systems leading the TOP500 list during the period from 2014 until 2016. Each year, number of HPC systems that comprise more than 4GB of main memory per core is increasing.

in; i.e. decreasing the number of application processes and compute nodes (servers) to solve a fixed-sized problem, using a set of HPC applications running in a production system. Using standard-memory nodes, we obtain average energy savings of 36%, already a huge figure. We show that the main source of these energy savings is a decrease in the node-hours ($node\_hours = \#nodes \times exe\_time$), which is a consequence of the more efficient use of hardware resources.

Scaling-in is limited by the per-node memory capacity. We therefore consider using large-memory nodes to enable a greater degree of scaling-in. We show that the additional energy savings, of up to 52%, mean that in many cases the investment in upgrading the hardware would be recovered in a typical system lifetime of less than five years.

5.1 Introduction

Energy consumption is by far the most important contributor to HPC cluster operational costs, and it accounts for a large share of the total cost of ownership [5, 108]. For this reason, advanced energy-saving techniques in CPUs, cooling systems, next-generation memories and interconnects have been the subjects of significant industrial and academic research and development effort. Despite this investment, as shown in this chapter, researchers and users continue to overlook a simple measure that can dramatically reduce energy consumption, that of simply optimizing the number of compute nodes (servers) used to execute each job.

High-performance computing is broadly divided into capability and capacity computing. Capability computing refers to the use of a large-scale HPC installation to solve a single problem in the shortest possible time; e.g. simulating the human brain on a Tier-0 HPC system. In contrast, *capacity* computing refers to optimizing system efficiency to solve many mid-size or smaller problems at the lowest possible cost [32]. Typical examples of capacity computing would be small and medium enterprises using on-demand HPC resources to explore future product designs. In the context of capacity computing, the user is concerned not with the running time of a single job, but with the total running time of a batch of jobs and total system throughput.

This study investigates the potential for saving energy through *scaling-in* on large-memory nodes. Scaling-in refers to executing a fixed problem on a fixed machine, but using a reduced number of application processes and compute nodes. Scale-in increases single job execution time, but, as we quantify in this chapter, it substantially decreases energy consumption and reduces the running time of a batch of jobs. It is therefore of particular interest in the context of capacity computing. We study the trade-off between job/batch execution time, energy consumption and node-hours ($node\_hours = \#nodes \times exe\_time$) using a set of large-scale HPC applications running on a production HPC system. In summary, we find that scaling-in on standard memory nodes improves energy consumption by 36% on average, a huge figure. We investigate the sources of this energy savings, and show that its main source is a reduction in node-hours.

Scaling-in is limited by the per-node memory capacity, since, for a fixed size problem, reducing the number of nodes increases the memory required at each node. We therefore investigate the benefits of upgrading the per-node memory capacity in terms of energy savings and reducing the node-hours, and follow this with a financial cost-benefit analysis. We show that the additional energy savings, of up to 52%, mean that an investment in upgrading the memory would be typically recovered in less than five years.

5.2 Methodology

5.2.1 Hardware platform

We execute experiments on the MareNostrum 3 supercomputer [10] (see Chapter 3). Regular MareNostrum compute nodes comprise 32 GB of DDR3-1600 main memory (i.e. 2 GB per core). To evaluate the impact of main memory capacity upgrade on energy-efficiency, we execute some experiments on large-memory nodes. Large-memory nodes are identical to standard nodes except that their memory capacity has been upgraded to 128 GB (i.e. 8 GB per core).

5.2.2 Applications

We study HPC scaling behaviour using the Unified European Application Benchmark Suite (UEABS) [89], the set of production applications and datasets designed for benchmarking the European PRACE HPC systems for procurement and comparison purposes [90]. For more details about each application, please see Chapter 3. All applications are parallelized using MPI, and we executed them with the Test Case A dataset, which is scalable up to 1,024 processes. We ran the benchmarks with one MPI process per core, i.e. sixteen processes per node. Table 5.1 shows the six benchmarks that we analyzed. The table also shows the range of nodes on which we ran the benchmarks. In all cases, the maximum was 64 nodes (1,024 cores). The minimum was either a single node or the least number of nodes necessary to meet the memory re-

Table 5.1: UEABS applications used in the study.

Application	Science area	Memory [GB] [a]	Number of nodes
ALYA	Computational mechanics	15.1	1–64
NAMD	Computational chemistry	25.9	1–64
QE[b]	Computational chemistry	17.7	1–64
BQCD	Particle physics	14.4	4–64
GENE	Plasma physics	16.2	4–64
CP2K	Computational chemistry	17.0	8–64

[a] Per-node memory usage when application runs on the minimum number of nodes.
[b] QE stands for Quantum Espresso application.

quirements. We also list per-node memory requirements for the benchmarks running on the minimum number of nodes.

5.2.3 Power and energy measurements

The node power consumption was measured using IBM Active Energy Manager power modules, which monitor the voltage and current at the node power supply [40]. Active Energy Manager is part of the Integrated Management Module II in the firmware of the System x iDataPlex dx360 M4 [55]. The modules measure the node's total power consumption, including power supply, motherboard with all its components, CPUs, and memory. The MareNostrum node firmware samples the power consumption every second, and it computes the energy consumption by multiplying the measured power sample by the interval length of one second. Finally, the LSF batch job manager [38] sums the energy measurements during the whole execution of a job, and it reports the total in the job execution log file.

We estimate the energy consumption of the interconnect. Measurements from the node power modules already include the network interfaces in the node, so we focus on the energy consumption of the switches. Current network components are observed to have close-to-constant power demand, independent of load, with a deviation of less than 5% [81, 104]. This means that the total switch power consumption can be determined by adding up the vendor's figures for typical use, and, since power consumption is independent of activity, the total can be attributed to the nodes equally. We calculate a constant 7.0 W/node for the top-of-rack switches, 15.3 W/node for the core switches, and 4.8 W/node for the Ethernet storage and management networks, giving a total of 27.1 W per node. For a given job, the interconnect energy consumption can therefore be calculated assuming a constant power of 27.1 W per node.

5.3 Scaling-in on standard nodes

5.3.1 Execution time *vs.* node-hours *vs.* energy

Before running any experiment on an HPC machine, the user must choose to run the application on a particular number of nodes. This scenario, of a fixed problem to solve on a variable number of nodes, is known as strong scaling. The largest number of nodes is limited by the machine size and application's scalability: beyond a certain point, adding further nodes delivers diminishing returns. The *smallest* number of nodes is constrained by the amount of memory needed by the application: scaling-in the application too far would require more memory per node than is available.

Until now, the number of nodes has been chosen as a trade-off between execution time and node-hours, where the latter is the main "cost" exposed to the user, and is given by the number of nodes multiplied by the job's execution time. Figure 5.2 shows this trade-off for the ALYA application. As the number of nodes, on the x-axis, is increased from 1 to 64, the execution time drops by a factor of 27 (1/0.04),[1] while the node-hours increase by a factor of 2.37. At the same time, the energy consumption increases by a factor of 2.07. The energy consumption is about 90% compute nodes and 10% switches, and this ratio was roughly the same in all our experiments.

Figure 5.3 summarizes the energy results for all the applications under study.[2] Increasing the number of nodes above the minimum always leads to significant energy overheads, between 1.25× and 2.07×, with an average of 1.6×.

To understand these results in the context of capacity computing, we analyze how execution time, node-hours and energy are affected by scale-in and scale-out, for a single job and for many jobs. The upper half of Table 5.2 refers to the single-job experiments, discussed in the previous paragraph. In this case, scale-in greatly increases the execution time, since it reduces the use of concurrent hardware resources (to 1 node instead of 64). When we move to 64 jobs, however, as illustrated in the bottom half of the table the analysis changes. The number of 64 jobs is selected to simplify the illustration of the phenomena; the conclusions are applicable for *any* large number of jobs. Scale-in executes the set of jobs across all 64 nodes, with an independent job

[1]We report only the execution time of the HPC job. Time waiting in the job queue and/or moving the results to an interactive server for post-processing can significantly reduce the effective speed-up.

[2]Although QE processing Test Case A should scale up to 64 nodes [89], we observed good scalability up to 16 nodes but slowdowns for 32 and 64 nodes. We therefore report results for the range 1–16 nodes for QE.

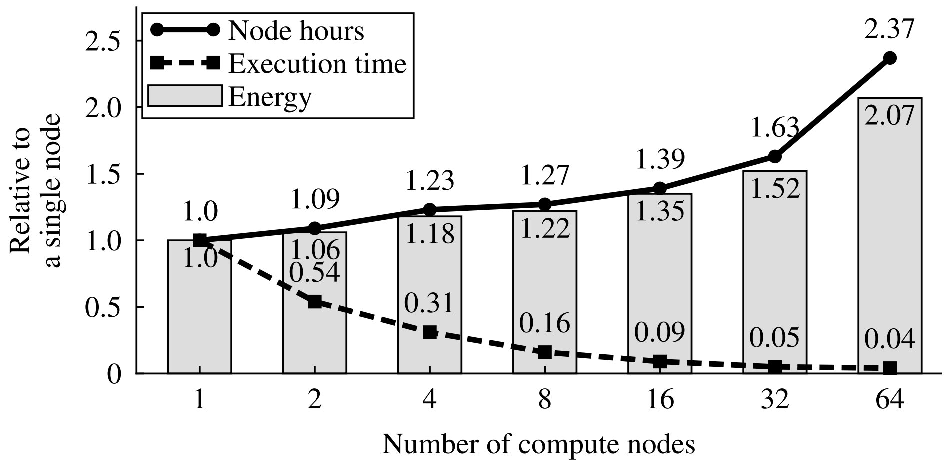


Figure 5.2: ALYA, 1–64 nodes: Increasing the number of nodes increases both energy and node-hours, with strong correlation.

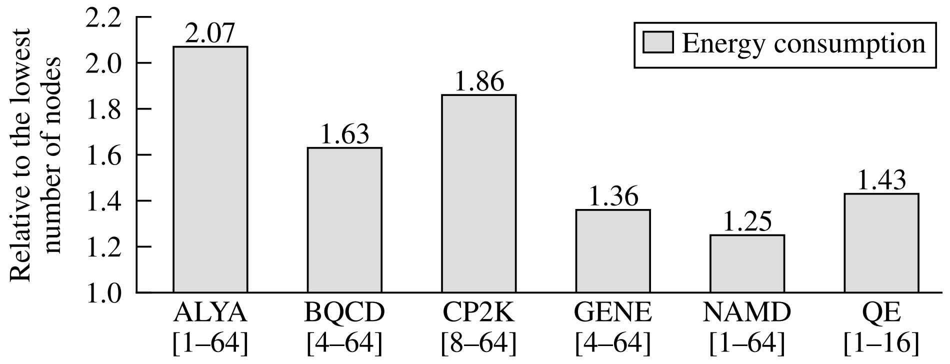


Figure 5.3: UEABS applications: Increasing the number of nodes causes significant energy overheads.

on each node. Although the total experiment size increases by a factor of 64, the execution time remains the same. In the scale-out approach, however, since each job already executes across all 64 nodes, the jobs execute one after another, and the execution time increases by a factor of 64. The scale-in approach is 27× slower for a single job, but 2.37× *faster* for 64 jobs. In both cases, scale-in approach reduces energy consumption by 2.07×.

Therefore, in capacity computing, where there are many smaller jobs, scaling-in improves all three metrics: execution time, node-hours, and energy consumption. Although important, this fact is overlooked by most of HPC research. To the best of our knowledge, this is the first study that *quantifies* improvements of scaling-in of large-scale HPC applications. We believe that the presented results will motivate further research in this direction and

Table 5.2: ALYA, 1 vs. 64 jobs: The scale-in approach is 27× slower for a single job, but 2.37× *faster* for 64 jobs. In both experiments scaling-in reduces node-hours by 2.37× and energy consumption by 2.07×.

	Nodes per job	Nodes	Exe time [min]	Node-hours	Energy [kWh]
1 job					
Scale-out	64	64	**1.14**	1.21	0.30
Scale-in	1	1	30.62	**0.51**	**0.14**
Ratio			0.04	2.37	2.07
Better approach			*Scale-out*	*Scale-in*	*Scale-in*
64 jobs					
Scale-out	64	64	72.71	77.56	19.10
Scale-in	1	64	**30.62**	**32.66**	**9.25**
Ratio			2.37	2.37	2.07
Better approach			*Scale-in*	*Scale-in*	*Scale-in*

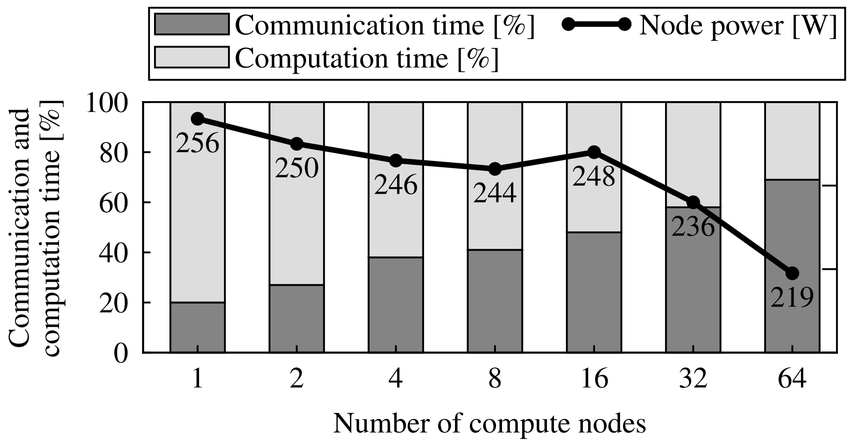


Figure 5.4: ALYA, 1–64 nodes: Scaling-out decreases power per node, since nodes spend more time in communication.

impact the policies for operational use of large HPC clusters.

5.3.2 Understanding energy *vs.* node-hours

For all the applications, as for ALYA shown in Figure 5.2, there is a clear correlation between the increments in node-hours and energy; however the two curves do not grow at the same pace. In fact, the node-hours curve always exceeds the energy curve. Since interconnect switch energy is proportional to node-hours (constant 27.1 W/node), the gap between the curves must come from a reduction in per-node power consumption. Figure 5.4 explores the node power consumption for the ALYA application, with each data point being the average of ten experiments. As the number of nodes, on the x-axis, is increased from 1 to 64, the per-node power reduces from 256 W to 219 W, a drop of 15%. The trend is not followed precisely, but the results are repeatable, as sample standard deviation was negligible.

The per-node power reduction comes due to changes in the behavior of HPC applications when scaling-out, mainly because of the increase in the communication-to-computation ratio. We trace the applications and measure the time spent in communication and computation with the Limpio instrumentation tool [85]. As shown in Figure 5.4, in the case of ALYA, increasing the number of nodes from one (16 processes) to 64 (1,024 processes) increases the proportion of time spent in communication from 20% to 69%. Since the power consumption of the communication (MPI functions) ranges between 200 W and 220 W, compared with about 280 W for computation, increasing the time spent in communication would pull down the average power consumption. In summary, for all applications under study, the higher the number of nodes, the higher the proportion of time that is spent in communication, and the lower the average per-node power consumption.

5.3.3 Implications and impact

The number of compute nodes to use in a given experiment impacts the application's execution time, node-hours and energy consumption, and, in aggregate, the throughput of the whole HPC system. This topic has not yet been thoroughly explored in the context of capacity computing. This is perhaps because HPC was traditionally biased to large public research centers and academia, which are heavily focused on capability computing.

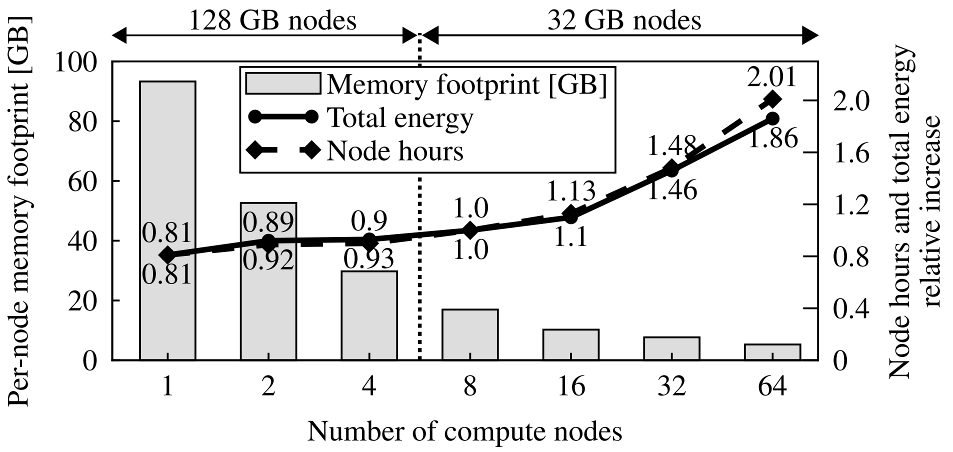
In recent years, however, HPC has entered industry, including small and medium enterprises, and many users now pay for their time on rented HPC resources. With this change, the cost of HPC experiments has become highly visible, and therefore of prime importance, and scaling-out of HPC applications is now a serious trade-off between execution time and cost. In addition to this, energy efficiency has become an important consideration in state-of-the-art HPC systems, and it is one of the main limitations in the design of future ones [5].

Considering these recent changes in HPC, and future requirements and limitations, it is important to rethink the scaling-out of HPC applications. The results in the previous section show that application scaling-out increases energy consumption on average by a factor of 1.6. Equivalently, from the point-of-view of current practice, scaling-in reduces the energy consumption by 36%, on average. To the best of our knowledge, this study is the first to emphasize how the number of nodes impacts energy consumption and to quantify the potential energy savings.

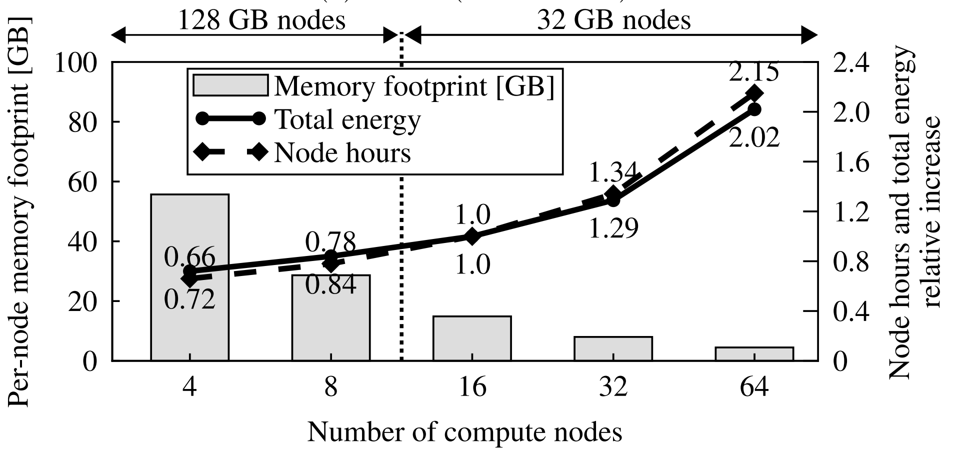
5.4 Large-memory nodes for energy efficiency

As described in previous section, reducing the number of nodes improves energy efficiency but it increases the memory demand per node, with the result that scaling-in is limited by the nodes' memory capacity. In Table 4.1 it was shown that CP2K, for example, requires at least eight MareNostrum nodes to fit the problem size. This is explained further in Figure 5.5. CP2K results are presented in Figure 5.5a, and they show how reducing the number of nodes, shown on the x-axis, to four or fewer causes the per-node memory footprint to exceed the standard node memory capacity of 32 GB. Figures 5.5b and 5.5c show the same trend for ALYA and QE applications processing Test Case B, the larger input dataset intended for Tier-0 HPC systems. ALYA application exceeds the 32 GB per node memory footprint on eight or fewer nodes, while QE requires at least 64 standard nodes to fit into the available main memory.

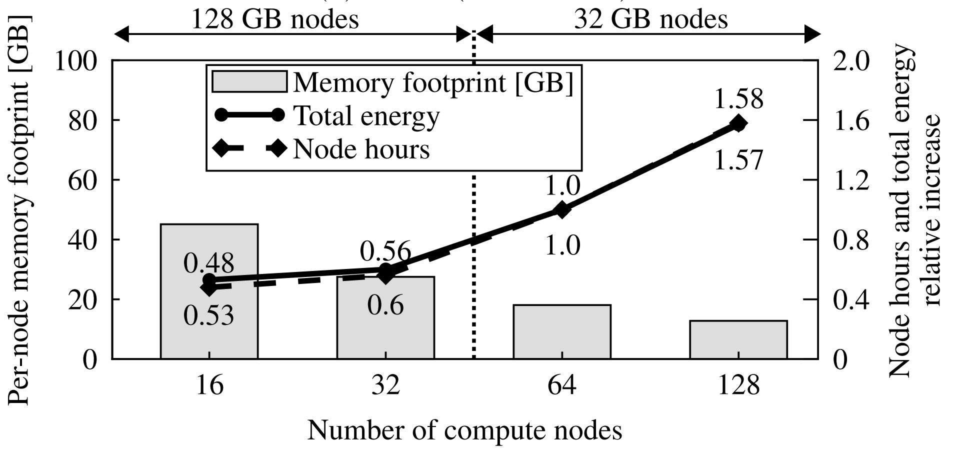
In addition to the memory footprint, Figure 5.5 also plots the node-hour and energy consumption curves. For CP2K in Figure 5.5a, the experiments with eight or more nodes use standard nodes, whereas the experiments with four or fewer nodes by necessity use large-memory nodes. Results are normalized to the eight-node experiment, which is the best result on standard nodes. It is clearly seen that, for the CP2K application, scale-in to large-memory nodes further improves the energy efficiency. Moving from eight standard nodes to a single large-memory node leads to 19% savings in both node-hours and energy consumption. In Figures 5.5b and 5.5c node-hours and energy curves follow the same trend as for CP2K, and the savings are even higher. For ALYA, scale-in from 16 standard to 4 large-memory nodes led to 28% energy and 34% node-hours savings, while QE saved 47% of energy and 52% of node-hours when moving from 64 to 16 nodes. We also analyze GENE running larger Test Case B. For GENE, shift from 128 standard to



(a) CP2K (Test Case A)



(b) ALYA (Test Case B)



(c) QE (Test Case B)

Figure 5.5: Scaling-in increases memory requirements and energy efficiency of HPC applications. Node-hours and energy are shown relative to the experiment on the minimum number of standard nodes.

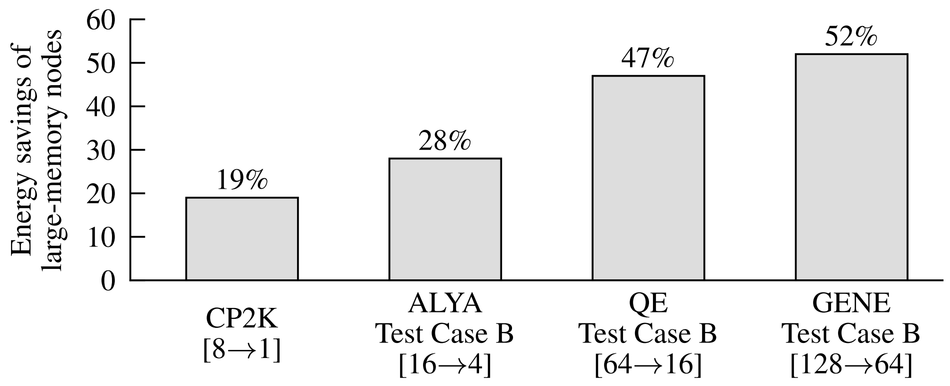


Figure 5.6: Summary of energy savings enabled by using large-memory nodes. $[a \rightarrow b]$ refers to a shift from a standard to b large-memory nodes.

64 large-memory nodes saved 52% of energy and 55% of node-hours.[3]

Figure 5.6 summarizes energy savings enabled by running the experiments on large-memory nodes. We detect energy savings from 19% for CP2K, up to 52% for GENE running Test Case B, with an average of 36%, a huge figure.

5.4.1 Large-memory nodes for capacity computing

To understand benefits of using large-memory nodes in the context of capacity computing, we analyze how execution time, node-hours and energy are affected on standard and large-memory nodes, for a single job and for many jobs. The upper half of Table 5.3 refers to the single-job experiments. In this case, using large-memory nodes increases the execution time by 6.5×, since it reduces the use of concurrent hardware resources (to one node instead of eight). When we move to eight jobs, however, as illustrated in the bottom half of the table, the analysis changes. Scale-in approach on large-memory nodes executes the set of jobs across all eight nodes, with an independent job on each node. Although the total experiment size increases by a factor of eight, the execution time remains the same. In the scale-in approach on standard nodes, however, since each job already executes across all eight nodes, the jobs execute one after another, and the execution time increases by a factor of eight. The scale-in on large-memory nodes approach is 6.5× slower for a single job, but 1.23× *faster* for eight jobs. In both cases, the scale-in on large-memory nodes reduces energy consumption by 1.24×. Therefore, in capacity computing, where there are many smaller jobs, using large-memory

[3]GENE is excluded from Figure 5.5 as it has only two data points, for 128 standard and 64 large-memory nodes.

Table 5.3: CP2K, 1 vs. 8 jobs: Execution on large-memory nodes is 6.5× slower for one job, but 1.23× *faster* for eight jobs. For both job sizes, node-hours and energy reduce when using large-memory nodes.

	Nodes per job	Nodes	Exe time [min]	Node-hours	Energy [kWh]
1 job					
Standard	8	8	**25.9**	3.45	1.07
Large-mem	1	1	168	**2.8**	**0.86**
Ratio			0.15	1.23	1.24
Better approach			*Standard*	*Large-mem*	*Large-mem*
8 jobs					
Standard	8	8	207.2	27.6	8.56
Large-mem	1	8	**168**	**22.4**	**6.88**
Ratio			1.23	1.23	1.24
Better approach			*Large-mem*	*Large-mem*	*Large-mem*

nodes improves all three metrics: execution time, node-hours, and energy consumption.

5.4.2 Large-memory node cost-benefit analysis

Finally, this section explores whether large-memory nodes are worthwhile from a financial point-of-view; i.e. whether the decrease in electricity costs would be sufficient to recover the cost of the extra memory. This analysis concentrates only on the financial return. Large memory nodes also increase system throughput, providing an extra benefit beyond that evaluated in this section.

In Table 5.4, each entry indicates the percentage payback, over a five-year system lifetime, from the reduced electrical costs delivered by large-memory nodes. Entries that exceed the break-even point of 100% are indicated in bold. The electricity cost for the U.S. is the average industrial price from August 2015 [115], whereas for the U.K, Germany and France industrial prices are from 2014 [33]. The memory upgrade from 32 GB to 128 GB was estimated to cost around $600 per node [88]. The benefit clearly depends on the mix of applications ran on the large-memory nodes, as different applications obtain greater or lesser energy savings. For CP2K, the 20% reduction in energy is not sufficient to recover the costs. For ALYA, a 30% energy

Table 5.4: Payback from large-memory nodes over five-year system lifetime [%].

Country	$/kWh	Reduction in energy consumption					
		10%	20% (CP2K)	30% (ALYA)	40%	50% (QE/GENE)	60%
U.S.	0.07	16	32	48	64	80	97
U.K.	0.15	33	66	99	**132**	**165**	**198**
Germany	0.17	38	75	**112**	**150**	**188**	**225**
France	0.10	22	44	66	88	**110**	**132**

saving means that the costs would be recovered in Germany. For QE and GENE, which both obtained roughly 50% reduction, the investment would be recovered in France, Germany and the U.K.

5.5 Related work

Significant industrial and academic research has been invested into energy-saving mechanisms for HPC components, such as CPUs, interconnects and memories. Several studies investigate how to employ CPU low-power modes in HPC. Current practice is to run the CPUs at the maximum voltage and frequency even while busy-waiting for an MPI message. Freeh et al. [34] investigate the tradeoff between energy and performance in MPI programs using DVFS. Using the NAS Benchmark Suite, they show that on one node it is possible to use 10% less energy while increasing time by only 1%. Lim et al. [57] propose an MPI runtime system that dynamically reduces the CPU performance during communication phases in order to minimize the energy-delay product (EDP). They show an average reduction in EDP of 10% across the NAS benchmarks suite.

Laros et al. [52] study how to combine CPU frequency scaling (for computation) and network bandwidth scaling (for communication) to reduce the energy consumption. On a set of Department of Energy (DOE) production applications running at large scale, they measure energy savings of up to 39%, with little or no impact on runtime performance. Their results also indicate that each application has a sweet spot based on its computation and communication requirements.

Regarding HPC interconnect energy consumption, Dickov et al. [23] reduce InfiniBand link energy by 21% by powering down the network links

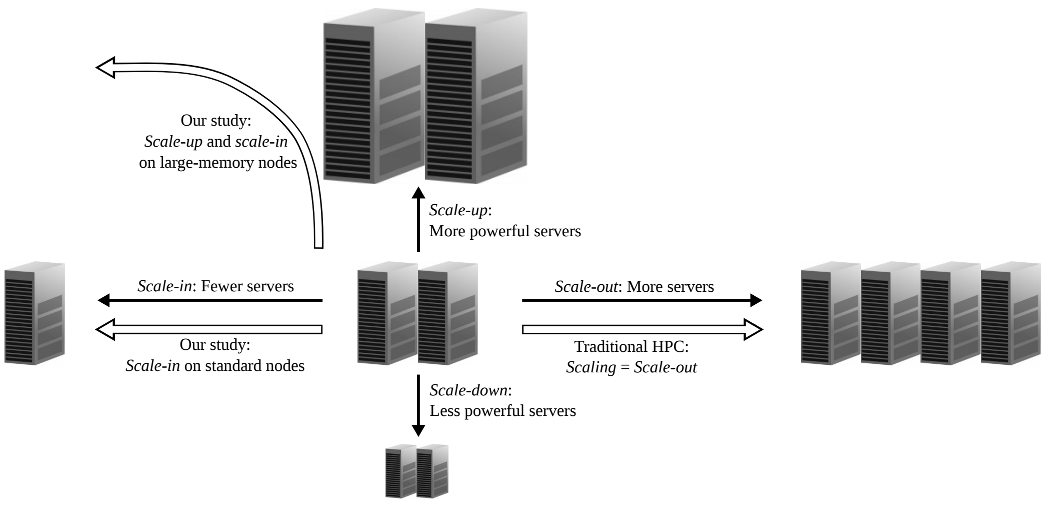


Figure 5.7: System scaling can be horizontal (*scale-in* or *-out*) and vertical (*scale-up* or *-down*). Traditionally, HPC community is focused mainly on *scale-out*, referring to it simply as *scaling*. Our study analyzes *scale-in* on standard nodes, and a combined *scale-up* and *scale-in* approach on large-memory nodes.

during the computation phases and using prediction to ensure that they are powered up in time for the next communication phase. Karthikeyan et al. [95] use prediction and an adaptive stall timer to reduce Ethernet link energy by 68%, while respecting a 1% bound on the increase in execution time.

Several previous studies deal with the energy efficiency of DRAM memory, through different memory management policies, intelligent data placement, and by creating opportunities to transition between power states [24, 62, 112]. Malladi et al. [61] use mobile DRAM devices in order to trade bandwidth for energy efficiency. These studies are validated for datacenter workloads, and it would be interesting to see to what extent their results are applicable to HPC.

In this chapter, we show that upgrading the memory capacity in HPC systems for capacity computing is a simple approach to save energy and reduce node-hours. In contrast to most of the prior research, our approach can be applied immediately, and it requires no changes to the system architecture, Operating System, system software or applications.

5.6 Second thoughts on scalability

The big data community distinguishes between two dimensions of system scaling — *horizontal*, which refers to the number of compute units, and *vertical*, referring to the hardware capabilities of each compute unit (see Figure 5.7). There are main two corresponding approaches for the analysis of huge data volumes: *scale-out* and *scale-up*. Scale-out means using more servers in parallel to spread out the workload, while scaling-up means using larger and faster servers to each handle a greater workload. The big data community is very active in analyzing the trade-offs between these two approaches, and whether both of them should co-exist within the same cluster [3].

In HPC, the dominant approach for addressing ever increasing HPC problems is scale-out. Actually, the community uses a general term *scalability* or *scaling* to refer to scale-out; while the more precise terms *scale-up/out*, *horizontal* and *vertical* scaling are rarely used or not used at all.

In modern HPC, the cost and energy consumption of the experiments has become highly visible and of prime importance. Our study demonstrates that simple but unconventional approaches of scale-in (standard node) or scale-up and scale-in (large memory nodes) can lead to significant savings in cost and energy, and improvements in throughput. Therefore, we hope that the study will motivate the community to consider the trade-offs between horizontal and vertical scaling when provisioning and using HPC clusters. Maybe we could start this journey with some second thoughts about the way that we use the word *scalability*.

5.7 Summary

The importance of energy consumption of current and future HPC machines means that significant research effort has been spent on advanced energy-saving techniques in HPC components. Despite this investment, the simple measure of scaling-in applications to reduce energy consumption has received little attention.

Scaling-in is most appropriate in the context of capacity computing, where a large number of mid-size or smaller problems have to be solved at the lowest cost, and the users are less interested in the execution time of a single job. We therefore advocate upgrading the memory capacity that allows further scaling-in in capacity computing. We validate this approach on a set of large-scale HPC applications running on a production system, and obtain average energy savings of 36%, a huge figure. Finally, we investigate the economical benefits of this approach, and show that the investment in upgrading the hardware would be typically recovered in less than five years.

Overall, we believe that this study will motivate further analysis of the trade-offs between horizontal and vertical scaling in HPC, especially in application domains that are on the border between HPC and big data analytics.

CHAPTER 6

DRAM errors in the field

Field studies of DRAM errors are essential for steering academic research and industrial practice in the most productive directions. This chapter summarizes our study of corrected and uncorrected errors on the MareNostrum 3 supercomputer, covering 2000 billion MB-hours of DRAM in the field.

The study clearly distinguishes between two different approaches for the DRAM error analysis. The first approach is to compare the errors at the DIMM level, and to partition the DIMMs into various categories, e.g. based on whether they did or did not experience an error. The second approach is to analyze the error rates, i.e., to present the total number of errors relative to other statistics, typically the number of MB-hours or the duration of the observation period. We show that although DRAM error analysis may be performed with both approaches, they are not interchangeable and can lead to completely different conclusions.

In addition to providing exploratory analysis, we perform statistical significance tests for each finding that we present. We show that various widely-accepted approaches for DRAM analysis may provide data that appear to support an interesting conclusion, but are not statistically significant, meaning that they could merely be the result of chance.

Overall, we believe that our study of methods for DRAM error analysis and reporting statistical significance of the results will become a standard for any future analysis of DRAM errors in the field.

6.1 Introduction

In large-scale compute clusters, main memory is one of the principal causes of hardware failures [36]. As memory capacities increase and DRAM processes shrink, it is thought that larger numbers of smaller transistors will have higher DRAM failure rates, impacting future system reliability [19]. It is especially important to understand reliability in high-performance computing (HPC), where a single tightly-coupled job may execute for days on thousands of nodes. If one of these nodes fails, the whole job is terminated. Component reliability therefore becomes an important limit on the ability to scale to larger systems.

This chapter summarizes our study of correctable and uncorrectable errors on the MareNostrum 3 supercomputer [12], covering 2000 billion MB-hours of DRAM in the field. MareNostrum 3 is one of six Tier-0 HPC systems in Europe; it comprises 3056 servers and more than 25,000 memory DIMMs. The study covers a period of more than two years, from October 2014 to November 2016, during which we detected 4.5 million corrected and 71 uncorrected DRAM errors. The results include all three major memory manufacturers and three different DRAM technologies.

In MareNostrum 3, and in the server domain in general, main memory is protected with error correcting codes (ECC). In modern HPC systems, sophisticated ECCs are able to correct multiple corrupted bits in a data word, and even handle cases where an entire DRAM chip is corrupted [21]. Data correction is performed in parallel with data read, so corrected errors effectively have no impact on system performance and reliability. But, if the ECC cannot correct a given DRAM error, the job typically has to be terminated and the server is shut down. The server is not operational until the DIMM is replaced and the node has been tested. The overall impact is lower reliability, lower system throughput and worse system availability.

Due to the requirement for high system reliability, original equipment manufacturers (OEMs) thoroughly test DIMMs from various manufacturers to certify that they can be used in production. It is usual practice, however, to quantify DIMM reliability using correctable errors, rather than the more important uncorrectable errors. Likewise, most of the DRAM error field studies focus their analysis on correctable errors, although *only* uncorrectable errors have an impact on system reliability.

Quantitative analysis of DRAM errors in the field can be performed with two approaches, which are often used interchangeably, although they differ greatly in stability and often lead to different conclusions. The first approach is to compare the errors at the DIMM level, and to partition the DIMMs into various categories, e.g. based on whether they *did* or *did not* experience an error. This first approach does not consider the number of errors that occurred on a given DIMM; the DIMM is categorized as soon as the first error is detected and any further errors do not change the DIMM's category. It is typically used to show the proportion of the DIMMs that experienced errors, or that were retired from the system. The second approach is to analyze the error rates. In this approach, the total number of errors is presented relative to other statistics, typically the amount of the MB-hours or the duration of the observation period. To the best of our knowledge, our study is the first to employ both approaches for DRAM error analysis and to clearly distinguish between them.

Categorical analysis: Similar to previous studies we detect that the percentage of DIMMs that experience uncorrectable errors is very small, and we notice some differences among manufacturers and DRAM technologies. However, to the best of our knowledge, we are the first to use statistical tests to validate these findings, and the first to show a lack of their statistical significance. We repeat the analysis for the corrected errors, and show a strong statistically significant difference between DRAM manufacturers and technologies. Contrary to the common belief that scaling down the technology reduces the DRAM reliability, our measurements show that the proportion of DIMMs that experience errors is reduced significantly in each DRAM generation. Finally, we show a strong dependency between DIMMs that experienced corrected and uncorrected DRAM errors. This validates the use of corrected errors as an indirect indicator of the memory systems reliability.

Error rates: First, we show that the findings based on the average errors rates, errors per MB-hour and MTBF, may be volatile and unreliable. Our results show that these findings may be completely different depending on the moment in which the measurements are taken, even after monitoring of more than 2000 billion MB-hours of DRAM in the field. Second, we show that using the correctable errors and faults rates as a DRAM reliability indicator is misleading because the uncorrected error trends can be completely different. Finally, after carefully considering the options, we conclude that there is no statistical test that can be used to reliably conclude statistical significance of the error rates results. Our study opens various doubts about the stability and usefulness of the DRAM error rate analysis especially if the results are based on the correctable errors and faults. Clarification of these doubts is very important because correctable errors and faults rates are the current standard for measuring DRAM reliability in both academia and industry.

Overall, we believe that our study will help the community to define standards for any future analysis of the DRAM errors in the field: focus on measurements with a practical value, and select a proper analysis method that provides stable results, ideally supported with statistical significance.

6.2 Background

In the last decade several studies have analyzed field DRAM errors. These studies have quantified the variations in error rates among DRAM manufacturers and technologies and analysed the nature of DRAM errors, including their temporal and spatial distributions. It is not easy, however, to compare the findings of different studies or to combine their findings into a clear overall understanding of memory system reliability, for two main reasons. First, the studies use non-unified terminology, especially when classifying the error types. Second, the studies give quantitative results without reporting whether or to what degree the reported results are statistically significant.

6.2.1 Taxonomy: Are correctable DRAM errors failures?

Most of the previous studies use the definitions of errors, faults and failures from Avizienis et al. [8]:

- **Failure** is an event that occurs when the delivered service deviates from correct service. For example, it is expected that a data read from memory delivers correct data stored on a given address. Deviation from this service is a failure.
- **Error** is the deviation of the system state (seen externally) from its correct service state. For example, the fact that a DIMM delivers to the memory controller data that do not match the ECC is the DRAM error.
- **Fault** is the adjudged or hypothesized root cause of the error. The cause of a DRAM error could be a particle impact, or a defect in the memory cell or circuit.

The problem is that the definitions of failures and errors are tightly coupled with the scope of the target system and its boundaries. For example, the memory system comprises the memory controller, DRAM devices and all the circuitry between them [43]. DRAM errors that are corrected by ECC in the memory controller **are not** errors or failures of the memory system, because the memory system still delivers correct data. Such correctable DRAM errors therefore have no impact on the service provided by the server and the overall HPC system. DRAM errors that cannot be ECC-corrected, however, propagate over the boundaries of the memory system, so they **are** also memory system errors and failures. In current HPC systems, such errors propagate even further, causing failures of the whole server and the affected

HPC job(s). The essence of the error classification problem is whether or not to categorize corrected DRAM errors as failures.

Although most previous studies categorize them as such, we believe that reporting corrected DRAM errors as failures, and using them to compute statistics such as the *failure rates* or the *mean time between failures (MTBF)* could be highly misleading as it exaggerates the problem of HPC system reliability. For example, a statement that the MTBF in MareNostrum 3 during the observation period was 14 seconds, based on the correctable error count in this study, could suggest that the system suffered frequent service interruptions. However, the provided number only states that at an average rate of once every 14 seconds, one out of eight memory controllers in one out of 3056 servers performs an ECC correction. The service is not interrupted and performance is not affected. So, it is very difficult to understand the practical value of this number. On the other hand, the mean time between uncorrected DRAM errors in MareNostrum 3 was 10 days (approximately 1 million seconds), meaning that on average every 10 days a single job is terminated, a single node is shut down and single DIMM is replaced.

Overall, it is important that DRAM error studies and the research motivated by them are clear as to whether the presented failure rates and MTBF values are based on correctable or uncorrectable errors, or both. On MareNostrum 3, the difference between MTBF values calculated using correctable vs. uncorrectable failures was five orders of magnitude; i.e. 14 seconds vs. 10 days. And we would strongly suggest to present numbers that have a practical value.

6.2.2 Statistical significance

Statistical significance means that a result from testing or experimenting has a low probability of occurring randomly or by chance, allowing us to conclude with confidence that it is likely to have a specific cause. Previous field studies of DRAM errors often claim that their findings are statistically significant because the analysis covers years of data on machines with thousands of servers, totalling thousands of billions of MB-hours.

Unfortunately, these claims are misleading. Statistical significance has to be confirmed or rejected using a carefully designed statistical test that considers the type and distribution of the data under study. As we show in this chapter, a large-scale experiment with a large number of observations, e.g. millions of corrected DRAM errors, does not *per se* guarantee statistical significance.

In addition to providing exploratory analysis, e.g. plotting the error rates for different memory manufacturers, we perform statistical significance tests for each finding that we present. Our analysis shows that various widely-accepted approaches for comparing DIMMs from different categories, e.g. different manufacturers, provide data that appear to support an interesting conclusion, but are not statistically significant, meaning that there is insufficient evidence to conclude that it is not merely the result of chance. We hope that these conclusions will encourage future work to analyse their data using formal statistical methods.

6.3 Environment description

6.3.1 MareNostrum 3

Our analysis is based on measurements of the memory errors on the MareNostrum 3 supercomputer [12] over a period of more than two years, from October 2014 to November 2016. Detailed explanation of MareNostrum 3 can be found in Chapter 3. MareNostrum 3 includes more than 25,000 DDR3-1600 DIMMs, and during the observation period we collected measurements on more than 2000 billion MB-hours. The main workloads executed on MareNostrum 3 are large-scale scientific HPC applications and the typical system utilization exceeds 95%.

We analyze DIMMs from all three major memory manufacturers, built in three different DRAM technologies. All the DIMM manufacturers presented in this study have been anonymized to protect the interested parties. In this chapter, we will refer to the different memory manufacturers as *Manufacturer A*, *B* and *C*.[1] Similarly, technologies in the DIMMs under study are also anonymized, and we show only the first of two digits of the nanometer technology. $\overline{3x}$ nm, $\overline{2y}$ nm and $\overline{2z}$ nm represent three different DRAM technologies in descending order, i.e., $\overline{3x}$ nm $>$ $\overline{2y}$ nm $>$ $\overline{2z}$ nm.

Each MareNostrum 3 DIMM contains two ranks. Most of the DIMMs, more than 90% of the DIMMs in our system, have nine DRAM chips in each rank, and each chip provides eight data (DQ) signals. These chips are usually referred to as x8 DRAM chips. The remaining DIMMs have 18 DRAM chips in each rank with four DQ signals (x4 DRAM chips). In both types of DIMMs, 72 bits in total comprise 64 data bits and 8 Error-Correcting Code (ECC) bits. MareNostrum 3 uses Single Device Data Correction (SDDC) ECC scheme for x4 devices, which provides single x4 error correction and double x4 error detection. This means that for x4 devices it provides Chipkill error correction, i.e. ECC can correct all errors coming

[1]There are 6717, 13,419 and 5247 DIMMs from anonymized *Manufacturers A*, *B* and *C* respectively.

from a single x4 device. For x8 devices it can correct consecutive 4-bit errors coming from the same DRAM chip.

6.3.2 Data collection

In Intel server architectures, the memory errors that are corrected by the ECC are recorded in the machine-check architecture (MCA) registers [47]. To log the **corrected DRAM errors**, we designed a daemon, based on the mcelog Linux kernel module [47], that periodically (each 100 ms) accesses the MCA registers, extracts the information of interest and logs them into a file. The log file contains the information about the error time stamp, server and DIMM id, and the exact physical location of the error in the DIMM including rank, bank, row, column and DQ pin. Also, the daemon can distinguish whether the correction was done on application memory read or patrol scrubbing.[2] If more than one error occurred in the 100 ms time interval, the MCA registers record the number of errors, but provide detailed information only for one error in the interval. Therefore, our logs contain the exact number of corrected errors that occurred in our system, while the detailed error information is available for a statistical sample (sampled in time) of all the errors.

If various DRAM errors occurred in the exactly the same physical location, they are counted as a single **fault**. The faults can be extracted only from the errors with known exact physical location. Increasing the frequency of daemon access to the MCA registers would increase the sample of errors with detailed information and the sample of observed faults. However, this would also increase the performance penalty of the error logging daemon. The 100 ms time interval was selected as the shortest time interval that causes less then 1% overhead to the production applications. Previous studies perform similar readings of the memory error registers with a period of a few seconds [105, 106, 107] or once per hour [56].

On a node restart, the daemon logs the DIMM locations, manufacturer information, and a serial number unique for each DIMM. This information enables us to keep the DIMM error and fault history, even if the servers or the DIMMs are moved.

Uncorrected errors are logged by the IBM firmware [39], which is part

[2]Patrol scrubbing is a technique for increasing memory system reliability. It periodically traverses the whole physical memory, performing an ECC check on each location. If the scrubber detects any errors that are correctable by the ECC, it fixes the errors and writes the correct data back to the same memory location. The main objective of patrol scrubbing is to deal with DRAM errors while they can still be corrected by ECC, and before they evolve to uncorrectable DRAM errors that would lead to system failures [94].

of the MareNostrum 3 monitoring software. For each uncorrected error, the log specifies the DIMM that failed and the cause of the error, i.e. whether the error happened during an application memory read or patrol scrubbing.

After an uncorrected error is reported, the corresponding DIMM is removed from production and exposed to a stress test.[3] If additional errors are detected during testing, the DIMM is retired or replaced. If no errors are detected, the DIMM is returned to production. In our study, we detected 71 uncorrected errors from 51 DIMMs.[4]

6.4 Categorical analysis

This section analyzes the percentage of DIMMs that experience errors, and evaluates whether there is a significant difference among the manufacturers and DRAM technologies. The presented analysis is formally referred to as a *categorical analysis* because the population of all DIMMs is divided into different categories based on, e.g., whether they *did* or *did not* experience an error.

6.4.1 Uncorrected errors

The results for the uncorrected errors are summarized in Figure 6.1. Figure 6.1(a) compares different DRAM manufacturers. Only 0.15% of *Manufacturers A* and *C* DIMMs experience uncorrected errors. For *Manufacturers B*, this percentage is somewhat higher, 0.25%. Figure 6.1(b) shows the technology comparison. The $\overline{3x}$ nm technology shows the best reliability with 0.14% of DIMMs experiencing uncorrected errors. For $\overline{2y}$ nm and $\overline{2z}$ nm technology the percentage of DIMMs with errors increases to 0.19%.

Overall we could conclude that the percentage of DIMMs that experience uncorrected errors is low, with some differences among the manufacturers and DRAM technologies. However, based solely on the results presented in Figure 6.1, we have no evidence as to whether these differences are *statistically significant*. In other words, we cannot know whether we should conclude that there really is a difference; e.g. the DIMMs from *Manufacturer B* have a higher probability of an uncorrectable error, or whether these results show

[3]The DIMMs is moved to stress-test server that executes the High-Performance Linpack for one week.

[4]A small number of DIMMs were exposed to a stress test only after experiencing more than one uncorrected error, and some returned to production and then experienced additional uncorrected errors.

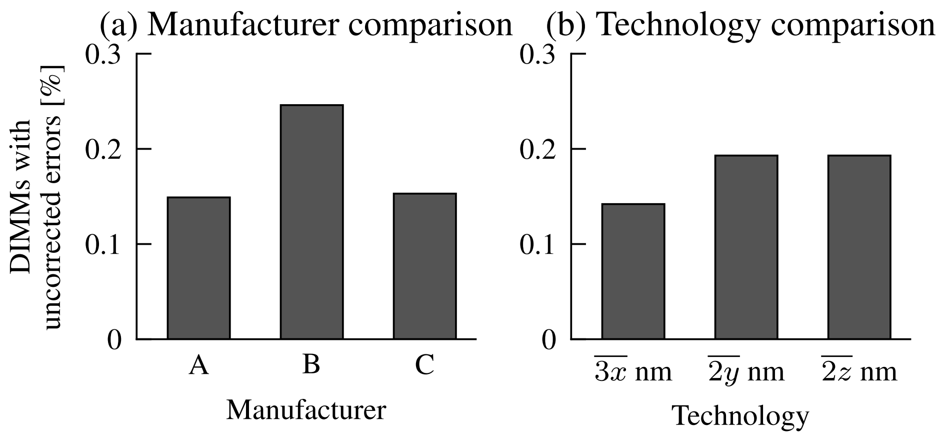


Figure 6.1: Percentage of DIMMs with uncorrected errors: Manufacturer and technology comparison.

	DIMMs w/ UEs	DIMMs wo/ UEs
Manufacturer A	10	6,707
Manufacturer B	33	13,386
Manufacturer C	8	5,239

Table 6.1: Contingency table: Dependency between the number of DIMMs that experienced uncorrectable error (UE) and the DIMM manufacturer. The statistical test indicates no dependency, p-value = 0.24, so we cannot claim any statistically significant difference in the probability that DIMMs from *Manufacturers A*, *B* and *C* will experience uncorrectable errors.

the typical variations due to chance that would be expected even without differences among the manufacturers and DRAM technologies.

The statistical significance of our results can be verified using independence tests that analyze the relation between two *categorical variables*. For example, we verify whether there is a statistically significant dependency between the number of DIMMs that experienced the errors (first categorical variable), and the DIMM manufacturer (second variable). Once that the categorical variables are defined, each DIMM is classified into one of the categories. In our example, the DIMMs can be classified into six categories based on the DRAM manufacturer (*Manufacturer A, B* or *C*) and the error occurrence (the DIMM *did* or *did not* experience an uncorrectable error), as illustrated in Table 6.1. Table 6.1 is referred to as a *contingency table*, and it shows the number of DIMMs that belong to each category. The contingency table is the input to a statistical test of independence that determines whether there is a significant relationship among the categorical variables. The test assumes the *null hypothesis* that the categorical variables are independent. The test output is the *p-value*, which is the probability of obtaining a result equal to or more extreme than what was actually observed, assuming the null hypothesis is true. If p-value is small, then we can conclude that the null hypothesis can be rejected, i.e., there is enough evidence to claim dependency between the variables. We use an $\alpha = 0.05$ cutoff for accepting or rejecting the null hypothesis, which is a standard value used in academia. In our study, we use **Pearson's chi-square test**, the most widely used test for the independence between two categorical variables.[5]

The test applied to our data shows no statistically significant dependence between the number of DIMMs that experienced uncorrectable error and the memory manufacturer, p-value $= 0.24$, meaning that although the results may seem to provide convincing evidence of a difference, we would expect similar or more extreme results to appear 24% of the time by chance, even if there were no differences at all. Also, in contrary to the common belief in the community is that as the technology scales down, the DRAM reliability decreases, measurements show no statistically significant decrease in the reliability for the three generations of DIMM technology used in our system, p-value $= 0.93$.

6.4.2 Corrected errors

The results for the corrected errors are summarized in Figure 6.2. Figure 6.2(a) compares different DRAM manufacturers. *Manufacturers B* and *C* have 3.3% and 5.1% percent of DIMMs with corrected errors, while for *Manufacturers A* it reaches 16%. Figure 6.2(b) shows the technology comparison. Contrary to the common belief in the community is that as the technology scales down the DRAM reliability decreases, our measurements show the opposite trend: $\overline{3x}$ nm technology has the highest percent of DIMMs with errors, followed by $\overline{2y}$ nm and $\overline{2z}$ nm technology, respectively.

As for the uncorrected errors, we use the statistical test of indepen-

[5]If cell values in the contingency table are small, it is recommended to use **Fisher's exact test**. Fisher's exact test is similar to Pearson's chi-square test, and a rule of thumb is to use it instead of a chi-squared test if more than 20% of the values in contingency table are lower than five. In all our results, we employ both Pearson's chi-square test and Fisher's exact test, using the `chisq.test()` and `fisher.test()` functions from the R programming language, respectively. Even for small cell values both tests have the same conclusions about accepting or rejecting the null hypothesis. Therefore, in the rest of the chapter, we report p-values from Pearson's chi-square test.

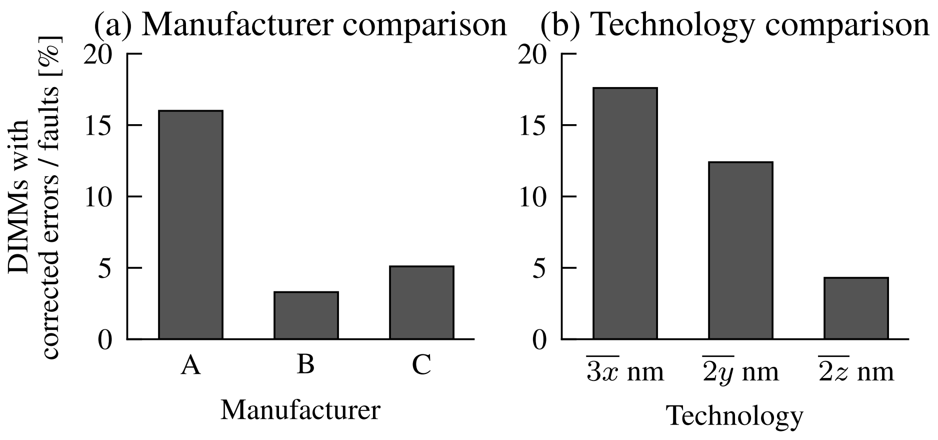


Figure 6.2: Percentage of DIMMs with corrected errors: Manufacturer and technology comparison.

dence to validate whether the detected differences between the manufacturers and technology are statistically significant. The test applied to our data confirms a statistically significant difference among DRAM manufacturers (p-value $< 2.2 \times 10^{-16}$) and technologies (p-value $= 1.51 \times 10^{-15}$).

6.4.3 Corrected vs. uncorrected errors

System reliability is affected only by uncorrectable memory errors. However, current practice in academia and industry is to analyze corrected errors and faults as DIMM reliability indicator [98, 56, 106, 107, 105, 99, 70], This practice is accepted because it might be intuitive that the DIMMs that experience corrected errors are more likely to also have uncorrected errors. However, a thorough study on this dependency has not yet been performed. In this section, we perform statistical tests to analyze the dependency between the DIMMs that experienced corrected and uncorrected errors. Validating this dependency would practically mean validating the use of corrected DRAM errors as a system reliability indicator.

The contingency table for this statistical test of independence is shown in Table 6.2. In different rows of the table, we show the number of DIMMs *with* and *without* an uncorrectable error (UE). Similarly, the table columns show the number of DIMMs *with* and *without* at least one correctable error (CE). The independence test indicates a strong dependency between correctable and uncorrectable errors, p-value $< 2.2 \times 10^{-16}$. We repeated the test for each manufacturer separately and observed the same conclusion. For *Manufacturers A, B* and *C*, the p-values are 0.013, less than 2.2×10^{-16} and 0.006, respectively.

	DIMMs w/ CEs	DIMMs wo/ CEs
DIMMs w/ UEs	23	28
DIMMs wo/ UEs	1,764	23,722

Table 6.2: Contingency table: Dependency between correctable (CE) and uncorrectable (UE) errors (All manufacturers). The statistical test indicates strong dependency, p-value $< 2.2 \times 10^{-16}$; i.e. we can claim that DIMMs that experienced CEs have higher probability of also experiencing UEs.

6.4.4 Errors vs. Faults

A couple of studies [105, 106, 107] argue that the DIMMs should be compared in terms of DRAM faults rather than errors. The categorical analysis presented in this section would directly apply to the faults as well. This is due to the inherent dependency between the errors and faults—a DIMM that experienced an error at the same time experienced a fault; while a DIMM with no errors, has no faults neither. Actually, all the contingency tables, p-values and conclusions for the DRAM faults would remain precisely the same as the ones that we presented for the errors.

6.4.5 Summary

In this section, we analyzed the percentage of DIMMs that experience uncorrected or corrected errors, and evaluated whether there is a significant difference among the manufacturers and DRAM technologies.

Similar to previous studies (see Section 6.6) we detect that the percentage of DIMMs that experience uncorrectable errors is very small, and we notice some differences among manufacturers and DRAM technologies. However, to the best of our knowledge, we are the first to use statistical tests to validate these findings, and the first to show a lack of their statistical significance.

We repeat the analysis for the corrected errors. The results indicate some differences among manufacturers and DRAM technologies, and, unlike for the uncorrected errors, the independence tests confirm with high confidence that these differences are statistically significant. Contrary to the common belief that scaling down the technology reduces the DRAM reliability, our measurements show that the fraction of DIMMs that experience errors has reduced significantly in each DRAM generation.

Finally, we show a statistically significant dependence between corrected and uncorrected DRAM errors, in this case we find that the probability of an uncorrected error is higher if the DIMM previously experienced corrected errors. This validates the use of corrected errors as an indirect indicator of the memory system's reliability. We hope that by showing that it may be possible to predict upcoming uncorrected DRAM errors based on preceding corrected errors, we will motivate future work on pre-failure detection.

6.5 Error rate analysis

In addition to the categorical analysis, DIMM reliability can be quantified and compared using the error rates: *errors per MB-hour* or *mean time between failures (MTBF)*. The errors per MB-hour metric considers the total number of errors, and the capacity and production time of each DIMM:

$$\text{Errors per MB-hour} \quad = \quad \frac{\text{Total number of errors}}{\sum \text{DIMM capacity [MB]} \times \text{Production time [hours]}}$$

The mean time between failures (MTBF) is computed as the ratio of the time in production divided by the total number of detected failures. The word *failure* can be used to refer to a fault, corrected or uncorrected error:

$$\text{MTBF [hours]} = \frac{\sum \text{Production time [hours]}}{\text{Total number of failures}}$$

Although errors per MB-hour and MTBF are both standard metrics for quantifying the DIMM reliability, to the best of our knowledge, no study has confirmed that they provide stable and meaningful results. Also, no study has supported their findings based on these metrics with a statistical significance.

In this section, we analyze the distributions of errors per MB-hour and MTBF, over the course of the 25-month observation period. This analysis is necessary in order to choose a test to determine the statistical significance of any finding based on these metrics. As in Section 6.4, we analyze both corrected and uncorrected errors, and the link between them.

6.5.1 Uncorrected error characterization

Distribution

Figure 6.3 shows the incidence of uncorrected errors over time. The x-axis shows time, in months from the beginning of the study (October 2014) and the y-axis shows the number of uncorrected errors per day, across all DIMMs of a single given manufacturer. First, we notice that the number of observed uncorrected errors is very low; we detect only 71 errors during the observation period of 25 months.[6] On most days we detect no errors, on a few days we detect a single error, and very occasionally we detect two or more (up to three) errors on the same day. Given the low incidence of errors, it requires a long observation period to converge to a stable mean error rate, which can change significantly each time that we detect a new error.

Figure 6.4 illustrates the volatility of the empirical mean error rates for the three manufacturers over time. The x-axis is again the time, in months from the beginning of the study. The y-axis is now the average number of uncorrected errors per billion MB-hour, based on the measurements done until that point.

Figure 6.4 shows that the errors per MB-hour evolve in time as an *impulse and down-ramp* function. The impulses in the average errors per MB-hour are caused each time a new error is detected. For example, if we observe *Manufacturer A*, the impulses in months 1, 8, 9, 12, etc., are perfectly aligned with the errors detected in Figure 6.3. This happens because the total number of errors is very small. For example, *Manufacturer A* experiences the first error in the first month of the observation. At the eight month, the second error is detected, which causes both the total error count and the errors per MB-hour to double. In the ninth month, when the number of observed errors changes from two to five, the total number of errors and the errors per MB-hour are both multiplied by 2.5.

The down-ramp segments observed in the plot of errors per MB-hour correspond to periods in which we detect no errors. In these periods the cumulative number of errors remains constant while the observation time increases. Therefore, the shape of the errors per MB-hour function is proportional to $1/t$, where t is the observation time. This is well illustrated for *Manufacturer A* between the first and eighth months of the observation period and for *Manufacturer B* between the second and eleventh months.

Figure 6.4 clearly shows the volatility when calculating the mean errors per MB-hour. The error rates can vary significantly, by tens of percents each time a new error is detected. As a consequence of the high variability in the error rates, the ranking of manufacturers switches several times. For example, 12 months into the study *Manufacturer A* had 90% higher error rate than *Manufacturer C*. At the end of the study, at month 25, *Manufacturer C* now had 60% higher error rate than *Manufacturer A*.

Overall, our results show that comparing different DRAM manufacturers based on the errors per MB-hour may support different conclusions depending on the moment in which the measurements are finalized. It is intuitive to conclude that we have little confidence in how the results would have looked

[6]These 71 errors were experienced by 51 faulty DIMMs.

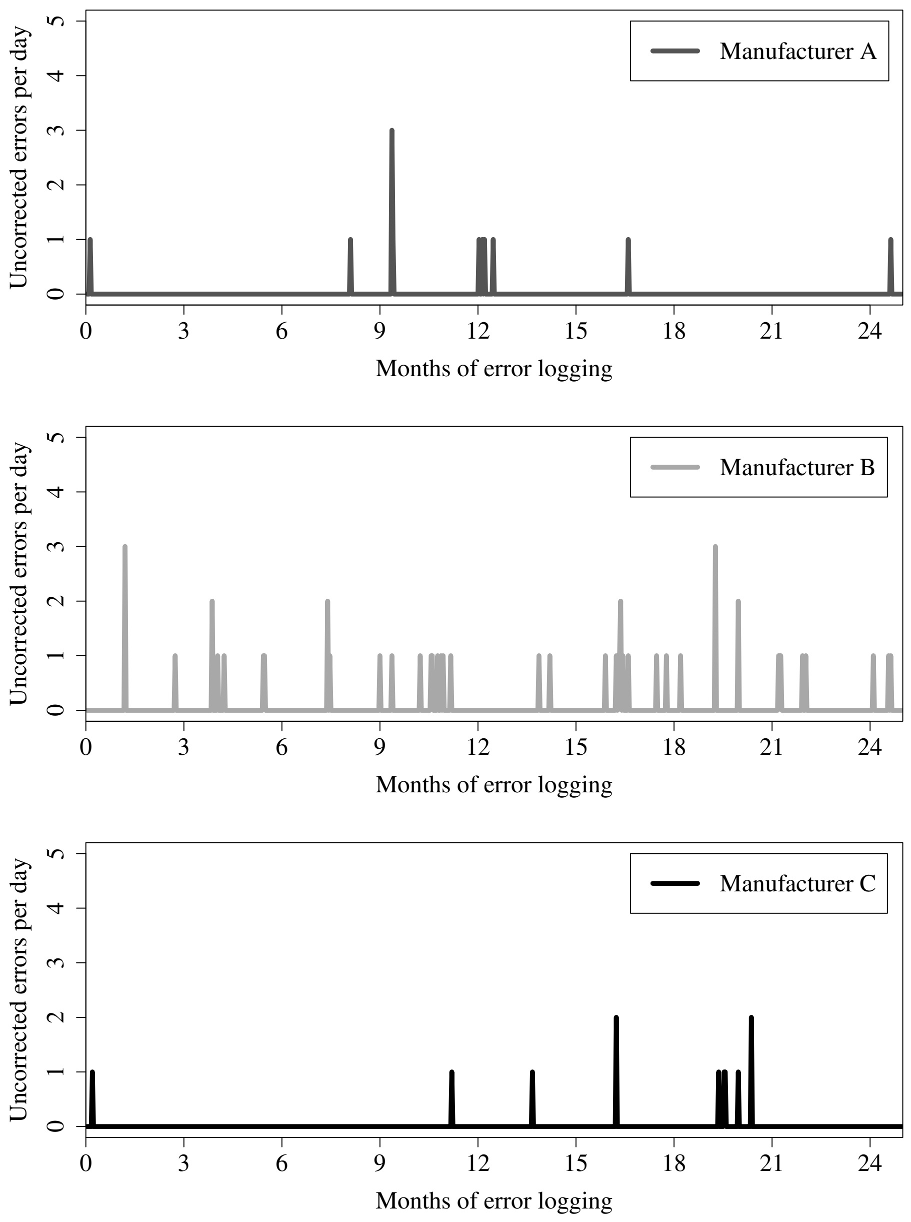


Figure 6.3: Uncorrected errors per day. The bars show the sum of the UEs for all the DIMMs of a given manufacturer.

if we were able to continue the study for another year.

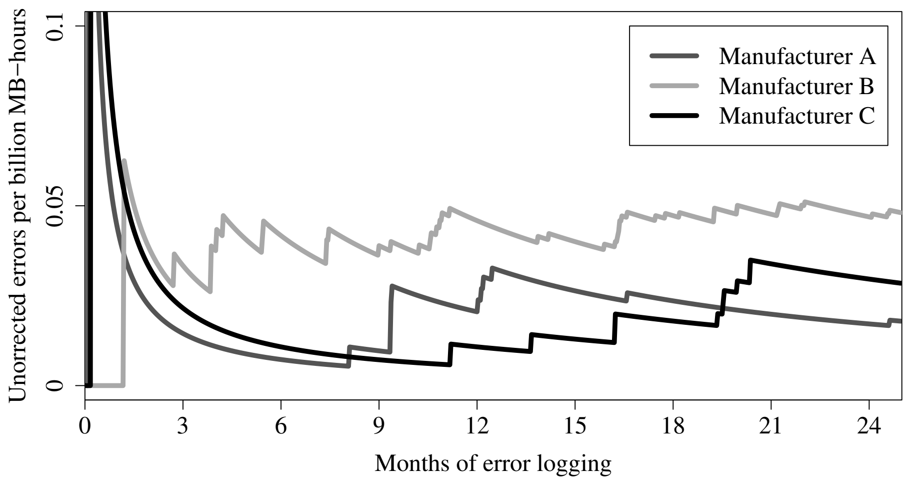


Figure 6.4: Average uncorrected errors per MB-hour: each point is the running average number of uncorrected errors per MB-hour observed up to that point. The running average is an *impulse and down-ramp* function. Depending on the moment of observation, we reach different conclusions about the ranking of the different DIMM manufacturers. During the observation period, *Manufacturer A* and *C* switched order twice.

Further discussion

Figure 6.4 illustrates the volatility of the conclusions when the error rates are used for comparison of different DRAM manufacturers. We repeat the whole analysis by looking into other DRAM categories, MTBF metric and the DRAM faults.

DRAM categories: We repeated the analysis for the three different DRAM technologies (rather than manufacturers) and reach the same conclusions. Since the total number of DRAM errors is the same as before, the number of non-zero observations contributing to the calculation of the mean is still small. We strongly argue that the same problem is present when the error rates are compared for various DIMMs categories, such as DIMMs located in different datacenters, different racks or servers, or DIMMs running different workloads.

MTBF: We used the same approach to test the MTBF metric, and the conclusions are the same—we detect huge variability in the MTBF, even after long periods of error logging. Since the essence of the problem is the small number of uncorrected errors, all statistics based on the error counts and means have high variability.

6.5.2 Corrected error characterization

We repeat the above analysis for the corrected errors. Although corrected and uncorrected DRAM errors have fundamentally different distributions, we obtain precisely the same conclusions: the average corrected errors per MB-hour has a large variance even after more than two years of error logging.

Distribution

Figure 6.5 shows the number of corrected errors over time. As before, the x-axis shows time, in months from the beginning of the study, and the y-axis shows the total number of correctable errors per day, for a single manufacturer. This figure clearly illustrates the error distribution: on most days there are zero, or close to zero, corrected errors, but on a few days there are very large numbers of correctable errors, up to about 110,000 (*Manufacturer B*, Month 7). When repeated at a finer granularity, by measuring not per day, but per hour, minute or second, we detected the same trend: >99% of the observations had no errors, and again a very small proportion of observations had very high values. In statistics, such distributions are referred to as *long-tailed* distributions. A classic problem with long-tailed distributions is that it can be difficult to determine the mean, which may be far from typical values and dominated by very infrequent observations.

Similarly to the uncorrected error case, to illustrate the difficulty in measuring the mean, Figure 6.6 plots the empirical mean error rates, per MB-hour, for the three manufacturers over time. This figure shows that corrected errors per MB-hour also evolve in time as an *impulse and down-ramp* function. An intensive error burst, as seen in Figure 6.5, *significantly increases* the number of detected errors in a short time interval, causing an impulse in the average errors per MB-hour function. For example, if we observe *Manufacturer C*, the impulses in Figure 6.6, e.g. in the observation months 2, 8, 11, 15, etc., are perfectly aligned with the intensive error burst in Figure 6.5. The down-ramp segments of the errors per MB-hour function, correspond to the periods in which we detect few errors, similarly to Figure 6.4.

Figure 6.6 clearly shows the volatility when calculating the mean errors per MB-hour. The error rates can vary significantly, by tens of percents in just a few days. We detect this behavior for all three manufacturers. Also, we detect this behavior not only at the beginning of the study, where it might be expected due to the small observation period, but also after long periods e.g. well after one year of the study. As a consequence of the

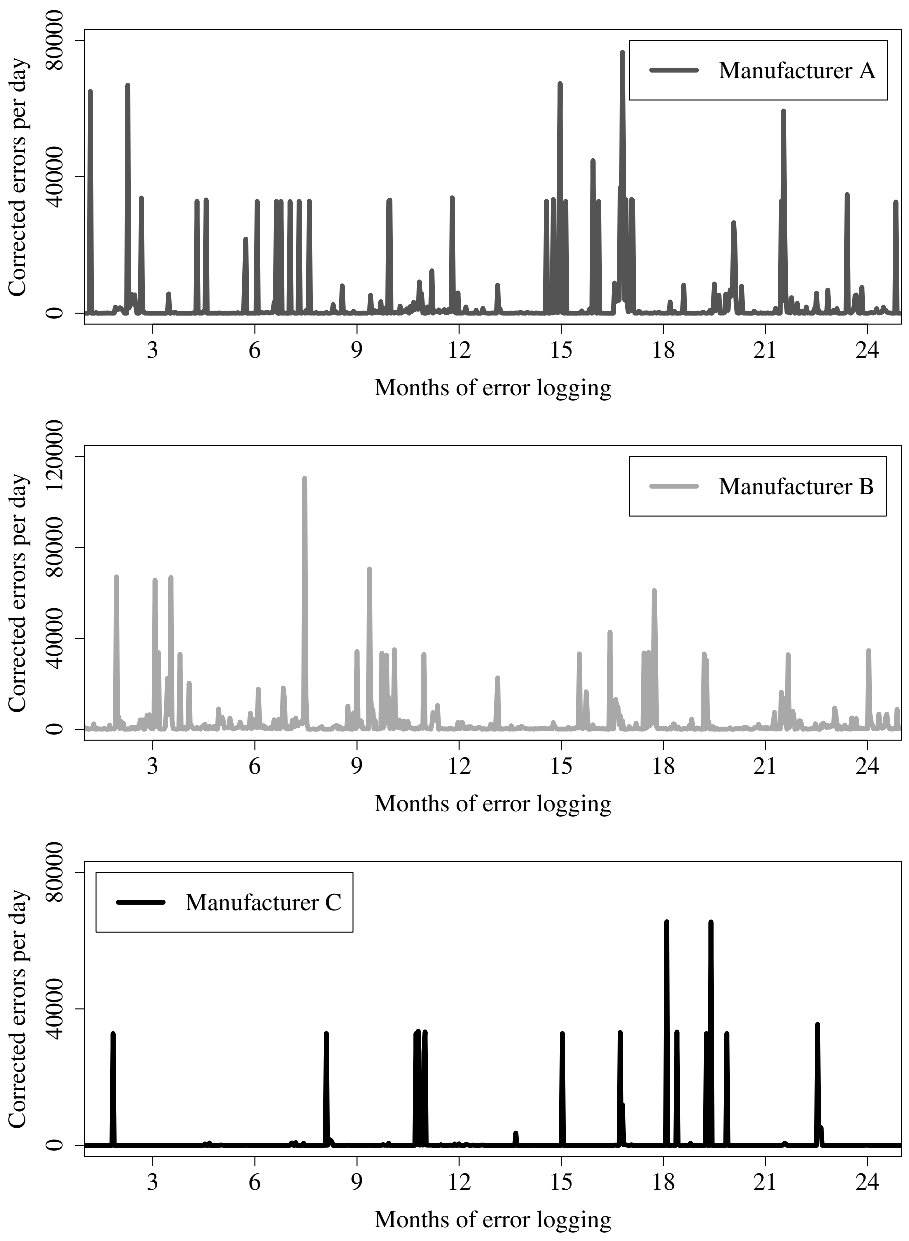


Figure 6.5: Corrected errors per day. On most days we detect zero or close to zero correctable errors; but on a few days there are very large numbers of correctable errors, up to about 110,000 errors.

high variability in the error rates, the ranking of manufacturers switches several times. Actually, during the observation period, *Manufacturer A* and *B* switched order eight times. At month 4, *Manufacturer B* had 40% higher

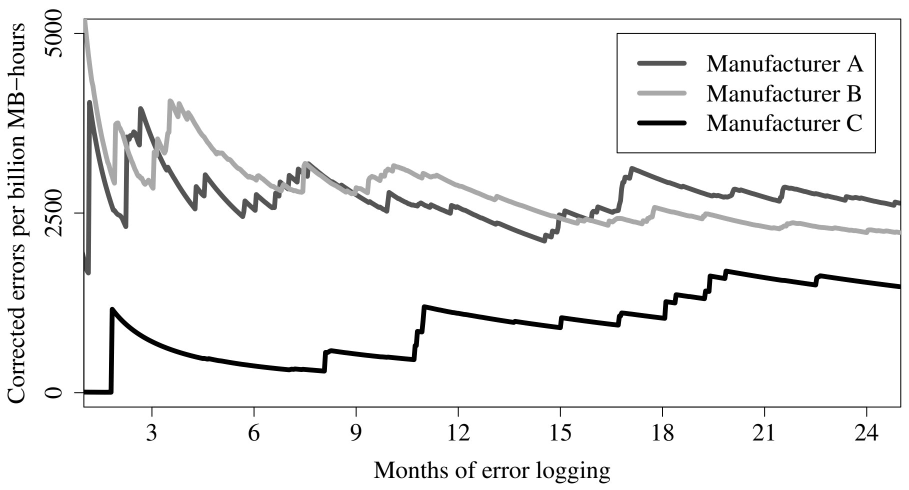


Figure 6.6: Average corrected errors per MB-hour: each point is the running average observed up to that point. The running average is an *impulse and down-ramp* function. Depending on the moment of observation, we reach different conclusions about the ranking of the DIMM manufacturers. During the observation period, *Manufacturer A* and *B* switched order eight times.

error rate than *Manufacturer A*, but at month 17, *Manufacturer A* had 25% higher error rate than *Manufacturer B*.

Overall, our results show that comparing different DRAM manufacturers based on the errors per MB-hour may support a different conclusions depending on the moment in which the measurements are finalized.

Further discussion

As for the uncorrected errors, we repeat the analysis for **DRAM technologies**, **MTBF metric** and the **DRAM faults** and the conclusions are the same—the error rates can vary significantly, by tens of percents in just a few days, not only at the beginning of the study, but also after long periods. Whereas uncorrected error rates (Section 6.5.1) had high volatility due to the small number of uncorrected errors, corrected error rates have high volatility due to a long-tailed distribution.

6.5.3 Statistical significance

With an understanding of the distributions of the uncorrected (Section 6.5.1) and corrected (Section 6.5.2) error rates per MB-hour, we consider the choice of statistical test to determine statistical significance. Whereas the categorical analysis using Pearson's chi-square test in Section 6.4 only needed to assume that the observations are random, independent and identically distributed, analysis of numerical data usually requires assumptions about the underlying distribution.

The most common tests for statistical significance in numerical data are the t-test (for two sets of data) and ANOVA (its generalization to three or more sets of data). Both tests assume that the sample means have a normal distribution. In many circumstances, this assumption is justified, either because the population itself is close to normal or because the average of a large sample is close to normal by the Central Limit Theorem. Unfortunately, however, if the data has a long-tailed distribution, convergence of the sample mean to normal is very slow, requiring an extreme sample size. Our experiments indicate that even with 2000 billion MB-hours and two years, the sample mean of the corrected errors is not normal, meaning that neither the t-test nor ANOVA should be applied. Due to the small number of non-zero samples for the uncorrected errors, it is not possible to determine whether or not this is also the case for uncorrected errors.

There are, however, additional statistical tests that do not assume a normal distribution, or any parametrized distribution (such as normal, whose parameters are the mean and variance). Such *non-parametric* tests include the Mann–Whitney U test (for two sets of data) and Kruskal–Wallis (its generalization to three or more sets of data). These tests are commonly thought to compare population medians, rather than means, but this is not strictly true. In our case comparing population medians would be useless since, for all DRAM manufacturers and technologies, >99% of MB-hours had zero (un)correctable errors, so the median number of (un)correctable errors per MB-hour is zero.

We applied the Kruskal–Wallis test and could not conclude that there is any statistically significant difference among the distributions (p-value $< 2.2 \times 10^{-16}$). It is important to realise that even if we had found a statistically significant difference, the strongest conclusion that we could have made would have been that the DRAM manufacturers have different distributions, not that one has a higher mean or median than another. To conclude the latter would have required an assumption of statistical dominance; i.e. that the cumulative distribution functions do not cross, but our experiments (not presented) show that it is quite likely that they do.

6.5.4 Corrected vs. uncorrected errors

Next, we compare the errors per MB-hour and MTBF metrics based on DRAM faults, correctable errors and uncorrectable errors. In case that these trends were similar, we could conclude that MTBF based on the correctable errors and faults might be used as an indirect indicator of the DRAM reliability. Our results clearly show that the measurements based on the corrected errors and faults indeed show similar trends, that are however **completely different** from the uncorrected errors.

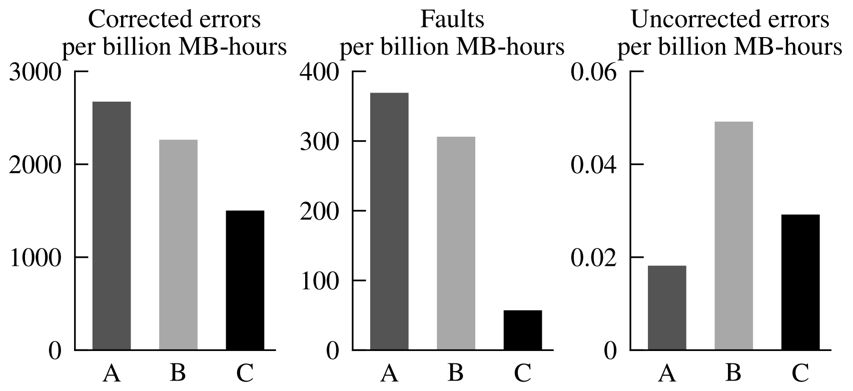
Errors per MB-hour

In Figure 6.7(a), we compare the errors per MB-hour for various manufacturers. Corrected errors, faults and uncorrected errors results are presented in separate charts, while different bars refer to the different DRAM manufacturers. Note that because of the big difference in error rates, charts in Figure 6.7(a) have different scales on the y-axis. When counting the corrected errors, the highest error rate, 2665 errors per billion MB-hours, is measured for *Manufacturer A*, followed by *Manufacturers B* and *C* with 15% and 44% lower error rates, respectively. Fault rates follow a similar trend, *Manufacturer A* has the highest fault rate followed by *Manufacturers B* and *C*. The uncorrected error rates, however, follow a **different trend**: *Manufacturer A* shows the lowest rate, followed by *Manufacturer C* (1.6$\times$ increment) and *Manufacturer B* (2.7$\times$ increment).

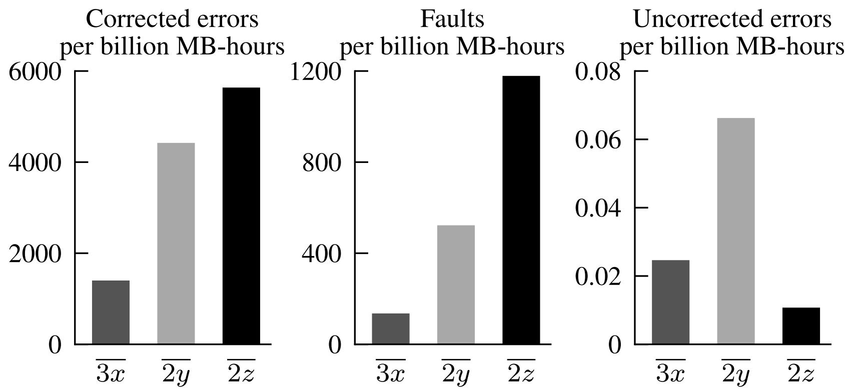
We get the same conclusion when comparing DIMMs with different technologies, see Figure 6.7(b). The corrected error and fault rates increase as the technology scales down from $\overline{3x}$ nm to $\overline{2y}$ nm and $\overline{2z}$ nm. The uncorrectable error rates, again, follow a **different trend**: the smallest technology, $\overline{2z}$ nm shows the lowest error rates followed by the $\overline{3x}$ nm (2.3$\times$ increment) and $\overline{2y}$ nm technology (6.3$\times$ increment).

Mean time between failures (MTBF)

Similarly to the analysis of errors per MB-hour, we compare the MTBF metric based on DRAM faults, correctable and uncorrectable errors. We perform the analysis for different DRAM manufacturers and DIMM technology, presented in Figures 6.8(a) and 6.8(b), respectively. As in the errors per MB-hour analysis, MTBF based on the correctable error and fault show similar trends, that are **completely different** from the uncorrectable errors.



(a) DRAM manufacturer comparison



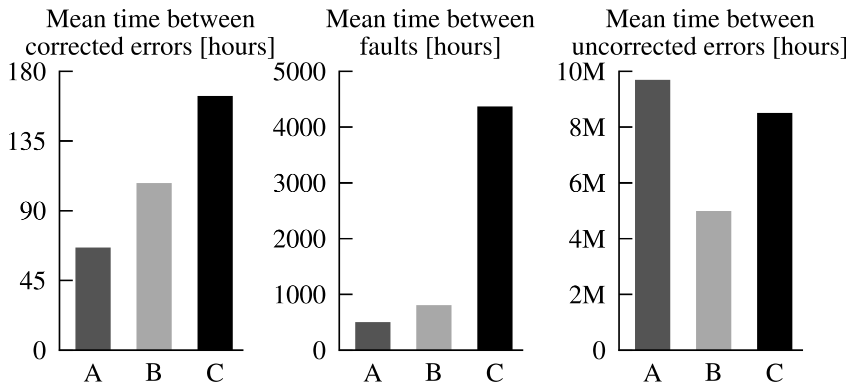
(b) DRAM technology comparison

Figure 6.7: Corrected errors, faults and uncorrected errors per billion MB hours. The corrected error and fault rates have the same trend, but the uncorrected error rates exhibit a **different trend**.

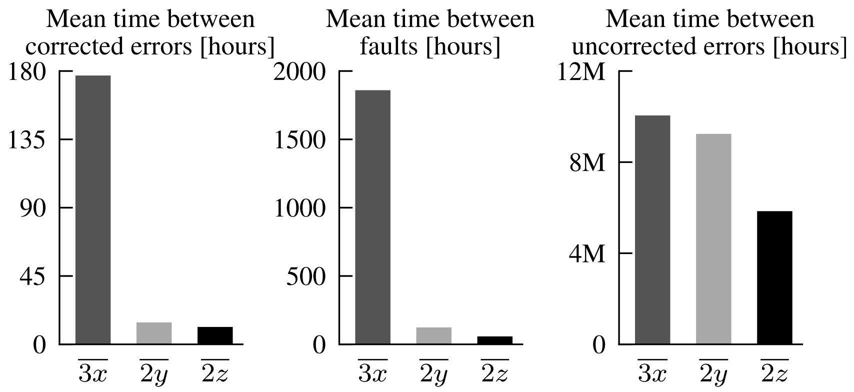
6.5.5 Error rates vs. Categorical analysis

As the final step of our study, we perform the analysis of the same DRAM error data with different approaches, categorical and the error rates, and we compare the findings that are the outcome of each of them. Our results clearly show that although quantitative DRAM error analysis may be performed with both approaches, they are **not interchangeable**. Actually, the conclusions of the analysis could be **completely different** from one approach to the other.

We illustrate this with two examples. Figure 6.9 compares the uncorrectable DRAM errors for three technologies under study, $\overline{3x}$ nm, $\overline{2y}$ nm and $\overline{2z}$ nm. Figure 6.9(a) shows the categorical analysis, the percent of DIMMs



(a) DRAM manufacturer comparison



(b) DRAM technology comparison

Figure 6.8: Mean time between corrected errors, faults and uncorrected errors. The corrected error and fault rates have the same trend, but the uncorrected error rates exhibit a **different trend**.

that experienced an uncorrectable error, while Figure 6.9(b) shows the uncorrectable error rates per MB-hour. It is clear that the trends on the figures are completely different, e.g. $\overline{2z}$ nm technology has **the highest** percent of the DIMMs with uncorrectable errors, while it has **the lowest** rate of the errors per MB-hour.

Figure 6.10 illustrates the same for the corrected errors. The trends on the Figures 6.10(a) and (b) are completely different depending on whether we compare the DRAM technologies based on the error rates per MB-hour or the percent of DIMMs that experienced an error.

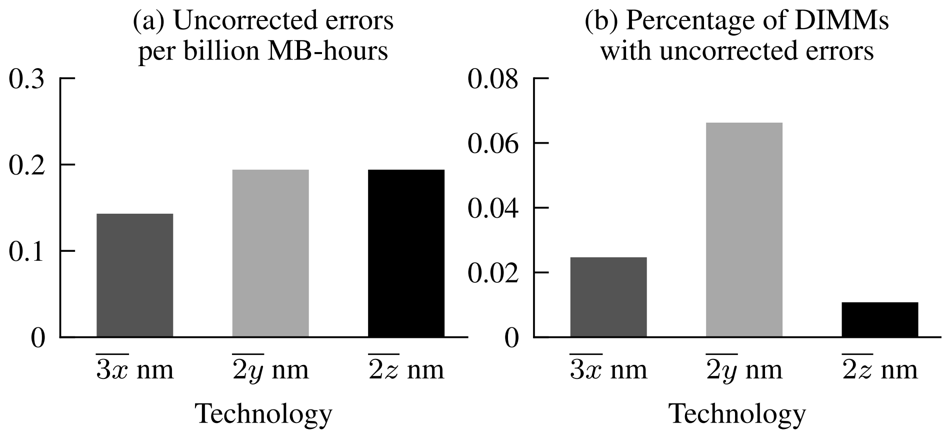


Figure 6.9: Technology comparison ambiguity, uncorrected errors: Errors per MB-hour vs. Percentage of DIMMs with errors.

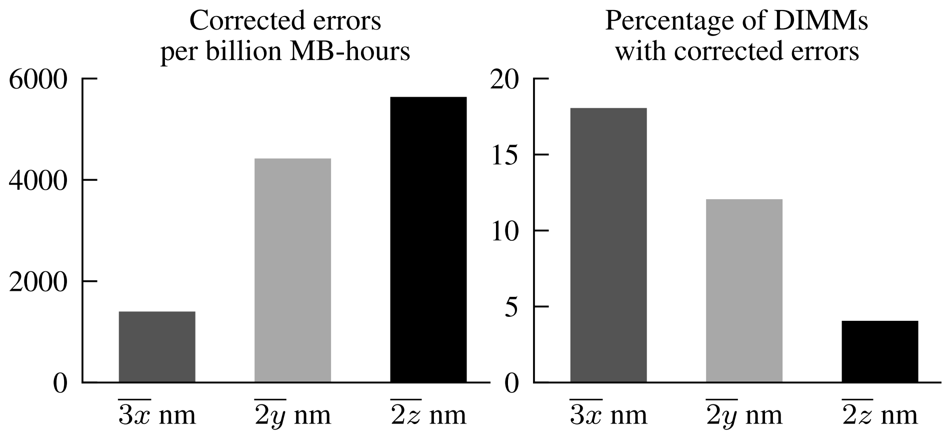


Figure 6.10: Technology comparison ambiguity, corrected errors: Errors per MB-hour vs. Percentage of DIMMs with errors.

6.5.6 Summary

In this section, we analyzed the DRAM error distributions and the variability of errors per MB-hour and MTBF over the course of the 25-month observation period.

First, we show that average errors rates, errors per MB-hour and MTBF, have a **large variance** even after **more than two years** of the error logging. The findings are the same for corrected and uncorrected errors and for both, DRAM manufacturer and technology comparison. We also show that errors per MB-hour and MTBF show different conclusions depending on the moment in which the measurements are taken. It is intuitive to conclude that we have **little confidence** in how the results would have looked if we were able to continue the study for another year.

Second, we show that using the correctable errors and faults rates, errors per MB-hour or MTBF, as an indicator of DRAM reliability is **misleading** because the uncorrected error trends can be **completely different**. This is one more example that shows how important it is to understand the relation between DRAM faults, correctable and uncorrectable errors. Any metrics based on correctable errors or faults should be used as a DRAM reliability indicator **only** if there is a clearly understood relationship with the uncorrected errors.

Finally, after carefully considering the options, we conclude that there is **no statistical test** that can be used to reliably conclude statistical significance of the error rates results for corrected, uncorrected errors and faults.

These findings are very important because precisely the correctable errors and faults rates are the **current standard** for measuring the DRAM reliability in both, academia and industry. Therefore, it is essential to question the current practice in quantifying DRAM reliability and to select a proper analysis approach. Our strong suggestion would be to use the method that provides stable and (ideally) statistically significant results. Another important requirement is that the selected method provides the numbers with a practical value that could be easily related to the physical properties of the system and its reliability.

6.6 Related Work

In recent years, various studies have analyzed correctable and uncorrectable DRAM errors and faults in the field.

6.6.1 Uncorrectable DRAM errors and whole system resiliency

Schroeder et al. [96] and Martino et al. [64] analyze the impact of DRAM errors on the resiliency of large-scale compute clusters. The authors consider numerous causes of the system failures including hardware components, software and environment. The analysis of the DRAM errors is only a small part of their studies.

Schroeder et al. [96] analyze failures of the Los Alamos National Laboratory HPC systems between 1996 and 2005. The authors report that uncorrected memory errors account for 20% of all hardware failures, and were the root cause of 30% of the server failures. Martino et al. [64] analyze failures of the Blue Waters supercomputer during 261 days. The supercomputer includes general purpose computing nodes with chipkill protected DDR3 and GPU accelerators with SEC-DED protected DDR5. The authors detect 1.5 million corrected and 28 uncorrected DRAM errors, and report that DRAM is the cause of 44% of all server failures.

These two studies are very important for two reasons. First, the studies show that DRAM is one of the main causes of hardware failures, and they quantify the impact of these failures on system reliability. This positions DRAM failures in the overall picture of large-scale compute cluster reliability. Second, when quantifying system reliability, the studies focus on uncorrected DRAM errors. Although the message is not as explicit as it could be, it is very clear: system reliability is driven by uncorrected errors not corrected errors. In our study, we emphasize this message and we question the practical value of any analysis focused on corrected DRAM errors. We also show that a quantitative analysis of corrected DRAM errors could be misleading, as it could give a wrong impression about the memory system reliability.

6.6.2 Correctable DRAM errors

Most DRAM error studies focus their analysis on corrected errors. These studies cover various large-scale compute systems, with DDR1, DDR2, DDR3 and FBDIMM DIMMs.

Schroeder et al. [98] present the first large-scale study of DRAM memory errors in the field. The study covers 2.5 years (Jan 2006–June 2008) of DRAM error logging of the Google fleet with six different platforms using DDR1, DDR2 and FBDIMM memory with SEC-DED and chipkill ECC. The study analyzes corrected and uncorrected error probabilities and rates, and correlates them with different factors, such as chip capacity, temperature, utilization, aging and DIMM generation.

Li et al. [56] report on nine months of DRAM error collection from various platforms with a total of 800 GB of DDR2 memory. The authors pay special attention to a comparison of transient and non-transient errors, and the study discovers a significant number of non-transient errors, with multiple errors often occuring in the same row or column.

Sridharan and Liberty [106] analyze 11 months (Nov 2009–Oct 2010) of DRAM error logs from the Jaguar supercomputer located at the Oak Ridge National Laboratory. The study covers DDR2 DIMMs with chipkill ECC and presents detailed analysis of the corrected errors and fault types: permanent, transient, single-bit, multi-bit, row, column, bank, multi-bank and multi-rank. Sridharan et al. [107] extend this study with the analysis of the error logs from the Cielo supercomputer located at the Los Alamos National Laboratory. These logs observe 15 months (mid-2011–early-2013) of chipkill-protected DDR3 DIMMs. This study focuses on DRAM faults (corrected errors faults) and presents a detailed analysis of fault types, similarly to Sridharan and Liberty [106]. The study also analyzes fault rates as a function of the DRAM vendor, physical location of the fault in the DRAM device, location of the DRAM device in the data-center, and the data-center altitude. Sridharan et al. [105] extend these studies with the 18 months (April 2011–January 2013) of the error logs from the Hopper supercomputer located at the Lawrence Berkley Labs. The study covers DDR3 DIMMs with a chipkill ECC scheme. These three studies [106, 107, 105] strongly argue that "system health" should be measured in terms of DRAM faults rather than errors. The term "system health" is not explicitly defined, but if it is a synonym for system reliability, then we argue instead that it should be quantified by uncorrected DRAM errors, rather than corrected errors or faults.

Siddiqua et al. [99] analyze DRAM errors logs collected from 30,000 servers over a period of three years. This is the first study that uses the pattern of the error addresses to distinguish between errors caused by the memory module, memory controller, memory channel and bus. The authors conclude that memory module faults are by far the most dominant fault type. Meza et al. [70] extend this work, and distinguish between errors caused by the DIMM bank, row, column and cell. They analyze 14 months of DRAM error logs from the Facebook fleet comprising DDR3 DIMMs, and conclude that 85% of memory errors are not caused by the DIMM, but by the socket and memory channel, which is opposite to the conclusions of Siddiqua et al. [99]. Meza et al. [70] also analyze corrected error rates as a function of DIMM manufacturer, DIMM architecture, technology, workload characteristics, CPU and memory utilization. The study also shows that the number of corrected errors among servers that had at least one error follows a power law (Pareto) distribution, and that a small number of servers (e.g. 1%) account for most of the errors. The authors, therefore, question the practical value of reporting mean per-server error rates. This confirms our results and discussion in Section 6.5.2.

Corrected vs. Uncorrected DRAM errors: Only two studies mention the dependency between correctable and uncorrectable DRAM errors. The results of Schroeder et al. [98] indicate that two months after a corrected error the DIMM has higher probability to experience an uncorrected one. The authors also present the idea of an early replacement policy, where a DIMM is replaced after experiencing a significant number of corrected errors, rather than waiting for the first uncorrected one. Up to now, this idea has not been validated. Sridharan and Liberty [106] confirm that the probability of an uncorrected error may increase if the DIMM had preceding corrected errors, especially if the corrected errors affected various ranks and banks of the DIMM.

All the studies that report on the uncorrected DRAM errors agree that the probability that a given DIMM experiences an error is very low, in the order of 0.x% or 0.0x%. Therefore, it is essential that any analysis of the uncorrected error probabilities is supported by statistical tests that distinguish between findings that are statistically significant and those that show a typical variation due to chance. Our work, to the best of our knowledge, is the first study that uses a solid statistical approach to this analysis, and formally shows a strong dependency between correctable and uncorrectable DRAM errors. Even more important, we believe that the methodology, statistical tests and examples presented in our study make a solid base for any future analysis of the dependencies between different DIMM categories.

6.7 Summary

This study summarizes our two-year study of corrected and uncorrected errors on the MareNostrum 3 supercomputer, covering 2000 billion MB-hours of DRAM in the field.

To the best of our knowledge, this is the first study that clearly distinguishes between two different approaches for the DRAM error analysis, categorical and via error rates. Although DRAM error analysis is typically performed with both approaches, we show that the approaches are not interchangeable, and can lead to completely different conclusions.

Categorical analysis: Similar to previous studies we detect that the percentage of DIMMs that experience uncorrectable errors is very small, and we notice some differences among manufacturers and DRAM technologies. However, to the best of our knowledge, we are the first to use statistical tests to validate these findings, and the first to show a lack of their statistical significance. We repeat the analysis for the corrected errors, and show a strong statistically significant differences between different DRAM manufacturers and technologies. In contrary to the common belief that scaling down the technology reduces the DRAM reliability, our measurements show that percent of DIMMs that experience errors is reduced significantly in each DRAM generation. Finally, we show a strong dependency between DIMMs that experienced corrected and uncorrected DRAM errors. This validates the use of corrected errors as an indirect indicator of the memory system's reliability.

Error rates: First, we show that the findings based on the average errors rates, errors per MB-hour and MTBF, can be completely different depending on the moment in which the measurements are taken. The error rates have a large variance even after monitoring of more than 2000 billion MB-hours of DRAM in the field. It is intuitive to conclude that we have little confidence in how the results would have looked if we were able to continue the study, e.g. for another year. Second, we show that using the correctable errors and faults rates as a DRAM reliability indicator is misleading because the uncorrected error trends can be completely different. This shows, once more, how important it is to understand the relationships between DRAM faults, correctable errors and uncorrectable errors. Finally, after carefully considering the options, we conclude that there is no statistical test that can be used to reliably conclude statistical significance of the error rates results. Our findings open various doubts about the stability of the DRAM error rate analysis especially if the results are based on the correctable errors and faults. Clarification of these doubts is very important because precisely the correctable errors and faults rates are the current standard for measuring the DRAM reliability in both, academia and industry.

Overall, we believe that our study will help the community to define standards for any future analysis of the DRAM errors in the field. We hope that our analysis will help the future studies to: First, focus on the measurements with a practical value that can be easily related to the system reliability. And second, select proper analysis approach that provides reliable results, ideally supported with statistical significance. If we ignore these guidelines, we are in a great danger that the value of the large-scale DRAM error studies diminishes. And the larger the systems, longer the observation periods, and more detailed the error logs, the higher the probability that our conclusions will be no more than a misleading noise, merely the result of chance and randomness of the in-field studies.

CHAPTER 7

Conclusions

DRAM has been the dominant main memory technology in most computing systems for decades. But, the expected end of DRAM technology scaling prevents further improvements in memory capacity and in the cost of DRAM production.

This thesis studied capacity and reliability issues in current memory systems for high-performance computing. The contributions of the thesis can be classified in three groups. First, we presented the study of memory capacity requirements of important high-performance computing benchmarks and applications. Second, we analyzed the scaling-in of high-performance computing applications on large-memory nodes to reduce energy consumption. Finally, we presented a field study of DRAM errors. The following sections briefly summarize each of the contributions.

7.1 Thesis contributions

7.1.1 Memory capacity requirements of HPC applications

This thesis analyzed memory capacity requirements of important HPC benchmarks and applications. This analysis becomes increasingly important as 3D-stacked memories are hitting the market. These novel memories provide significantly higher memory bandwidth and lower latency, leading to higher performance and better energy-efficiency. However, the adoption of 3D memories in the HPC domain requires use cases that need much less memory capacity than currently provisioned. With good out-of-the-box performance, these use cases would be the first success stories for these memory systems, and could be an important driving force for their further adoption.

We detected that HPCG could be an important success story for 3D-stacked memories in HPC. With low memory footprints and performance directly proportional to the available memory bandwidth, this benchmark is a perfect fit for memory systems based on 3D chiplets. HPL, however, could be one of the main show-stoppers because reaching a good performance requires memory capacities that are unlikely to be provided by 3D chiplets.

In addition to this, we demonstrated that the analysis of memory footprints of production HPC applications requires an understanding of their scalability and target category, i.e., whether the workloads represent capability or capacity computing. The results show that most of the HPC applications under study have per-core memory footprints in the range of hundreds of megabytes — an order of magnitude less than the main memory available in the state-of-the-art HPC systems; but we also detect applications and use cases that still require gigabytes of main memory per core.

7.1.2 Large-memory nodes for high-performance computing

We analyzed scaling-in of HPC applications, i.e., decreasing the number of application processes and compute nodes to solve a fixed-sized problem, as a practical approach to reduce energy consumption. We showed that scaling-in is most appropriate in the context of capacity computing, where a large number of mid-size or smaller problems have to be solved at the lowest cost, and the users are less interested in the execution time of a single job. We therefore advocated upgrading the memory capacity that allows further scaling-in in capacity computing. We validated this approach on a set of large-scale HPC applications running on a production system, and obtained significant energy savings.

Furthermore, we investigated the economical benefits of this approach, and showed that the investment in upgrading the hardware would be typically recovered in less than five years.

7.1.3 DRAM errors in the field

Finally, we studied DRAM errors on the MareNostrum 3 supercomputer during a period of more than two years. The field studies of DRAM errors are essential for the understanding of the nature, rates and distributions of errors in memory systems and are the main requirement for quantifying the effectiveness of any error mitigation strategy.

Our study clearly distinguished between two different approaches for the DRAM error analysis: categorical analysis and the analysis of error rates. We showed that although DRAM error analysis may be performed with both approaches, they are not interchangeable and can lead to completely different conclusions.

In addition to providing exploratory analysis, we performed statistical significance tests for each finding that we present. We also demonstrated the importance of providing statistical significance and presenting results that have practical value and real-life use. We showed that various widely-accepted approaches for DRAM analysis may provide data that appear to support an interesting conclusion, but are not statistically significant, meaning that they could merely be the result of chance.

7.2 Future work

7.2.1 Memory capacity study

In this thesis, we analyzed memory capacity requirements of important HPC benchmarks and applications. A simple but important question *"How much memory do we need in HPC?"* has not been discussed thoroughly by the academic and computer architecture community, and we hope that our study will trigger further research and discussion on this topic.

We focused our analysis on systems built with high-end x86 cores, the architecture that dominates the HPC market. HPC applications executed on other CPU architectures, such as the Blue Gene systems based on PowerPC embedded cores, systems based on embedded ARM [92] or systems with GPU accelerating cores, may have significantly different behavior and memory-related requirements. Therefore, we believe that interesting avenue of future work would be to repeat the presented analysis on non-x86 HPC systems and compare the obtained results with the findings of our study.

As a part of the future work, we plan to analyze weak scaling of production HPC applications and its impact on HPC memory footprints. Weak scaling analysis is especially important to anticipate future HPC problems with significantly larger input datasets. This analysis, however, would require HPC production application benchmark suites that allow the problem size to be tuned in a similar way to HPL and HPCG, or at least provide a collection of comparable input sets with varying problem size.

The study presented in this thesis included the experiments performed on a particular machine (MareNostrum supercomputer) and on a set of applications that represent the workloads running on current HPC systems.

Therefore, our analysis represents a set of isolated experiments that should be repeated on many different platforms and applications in order to confirm the conclusions. Another approach would be to analyze memory usage on existing HPC systems and correlate memory usage with the workloads. This analysis would require a software daemon which would log the memory usage and the type of workload being executed. Therefore, we would like to study the histograms of memory system usage of existing HPC clusters and understand whether, in every day usage of HPC systems, users take advantage of most of the installed memory.

We analyzed scaling-in of HPC applications as a practical approach to reduce energy consumption, and showed that scaling-in is most appropriate in capacity computing. In recent years, HPC has entered industry, including small and medium enterprises, and many users now pay for their time on rented HPC resources. It would be interesting to extend our study and to analyze industrial applications targeting capacity computing. We believe that applications from the industrial sector could fit well the analysis presented in Chapter 5. Users from industry run their applications on rented HPC systems and are usually interested in the performance of batch of jobs. If proven to HPC providers that scaling-in improves the energy efficiency, maybe it would lead to the scheduling policies in HPC systems that reward users that scale-in their applications.

7.2.2 Memory reliability study

We presented a study of DRAM errors, covering the data logged on a production HPC system during a period of more than two years. We showed that uncorrected errors are very rare. For this reason, we would need more field data logged during longer periods of time to have more statistical significance in our results. As a part of the future work, we would like to extend our analysis to include more production systems and clusters. Ideally, we would like to have logs from the whole lifetime of a production system. This way, we would have significantly more MB-hours for the analysis, and we could even observe the effect of aging, i.e., how reliability changes in the early and later phases of system production lifetime.

In our study, all memory DIMMs were DDR3 memory DIMMs, built from different memory manufacturers and in different nanometer technology. We would like to extend our study to include DDR4 DIMMs and even GDDR5 DIMMs used in graphics cards. This way, we could compare the rates and distributions of memory errors over several generations of the DDR standard.

We showed strong dependency between DIMMs that experienced corrected and uncorrected DRAM errors. This validates the use of corrected errors as an indirect indicator of the memory system's reliability. We believe that corrected errors and faults should be further analyzed in regards to predicting potential uncorrected errors on the DIMM. As a part of the future work, we would like to analyze patterns of the occurrence of corrected errors and faults on the DIMM, and try to recognize patterns that occur right before an uncorrected error. However, in order to use any of the patterns as a part of a pre-failure alert mechanism, the probability of uncorrected error should be significantly high. Only this way, we could compensate for the overhead induced by such pre-failure mechanism.

Finally, in our logs we had limited amount of information. We would like to extend our study and analyze how other parameters impact and correlate with the occurrence of uncorrected errors on the DIMM, such as temperature, application domain, memory bandwidth, memory usage, etc. This would enable much broader analysis, and possibly the detection of use-cases and domains that are more susceptible to memory errors.

7.3 Publications

In this section, we present a list of our research articles that are accepted for publication at conferences and journals. We also list the posters used to present our work at summits and forums, and publications on other topics that are not considered to be the contributions of the thesis.

7.3.1 Conferences

- Darko Zivanovic, Milan Radulovic, Germán Llort, David Zaragoza, Janko Strassburg, Paul M. Carpenter, Petar Radojković, Eduard Ayguadé. *Large-Memory Nodes for Energy Efficient High-Performance Computing.* In Proceedings of the Second International Symposium on Memory Systems (MEMSYS), Washington DC, US. October 2016.

  Best Paper Award.

7.3.2 Journals

- Darko Zivanovic, Milan Pavlovic, Milan Radulovic, Hyunsung Shin, Jongpil Son, Sally A. Mckee, Paul M. Carpenter, Petar Radojković, Eduard Ayguadé. *Main Memory in HPC: Do We Need More or Could We Live with Less?* In ACM Transactions on Architecture and Code Optimization (TACO), Volume 14 Issue 1, Article No. 3, April 2017.

— Invited for the presentation at the HiPEAC Conference, Manchester, UK. January 2018.

7.3.3 Posters

- Darko Zivanovic, Petar Radojković, Eduard Ayguadé. *Main Memory in HPC: Do We Need More or Could We Live with Less?* Fourth International BSC Severo Ochoa Doctoral Symposium. Barcelona 2017.
- Darko Zivanovic, Petar Radojković, Eduard Ayguadé. *Main Memory Provisioning for High-Performance Computing.* PRACEdays17, European HPC Summit Week 2017. Barcelona 2017.

7.3.4 Under submission

- Darko Zivanovic, Sergi Moré, Javier Bartolome, Paul M. Carpenter, Petar Radojković, Eduard Ayguadé. *DRAM Errors in the Field: The Signal and the Noise.* Under submission.

7.3.5 Other publications

- Milan Radulovic, Darko Zivanovic, Daniel Ruiz, Bronis R. de Supinski, Sally A. McKee, Petar Radojković, Eduard Ayguadé. *Another Trip to the Wall: How Much Will Stacked DRAM Benefit HPC?* In Proceedings of the International Symposium on Memory Systems (MEMSYS), Washington DC, US. October 2015.

Bibliography

[1] SPEC MPI 2007. `http://www.spec.org/mpi2007/`. [page 48]

[2] SPEC OMP 2012. `https://www.spec.org/omp2012/`. [page 48]

[3] Appuswamy, R., Gkantsidis, C., Narayanan, D., Hodson, O., and Rowstron, A. Scale-up vs Scale-out for Hadoop: Time to Rethink? In *Proc. of the Symp. on Cloud Computing (SOCC)* (Oct. 2013), pp. 20:1–20:13. [page 78]

[4] Asanovic, K., Bodik, R., Catanzaro, B. C., Gebis, J. J., Husbands, P., Keutzer, K., Patterson, D. A., Plishker, W. L., Shalf, J., Williams, S. W., and Yelick, K. A. The Landscape of Parallel Computing Research: A View from Berkeley. Tech. Rep. UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec. 2006. [page 48]

[5] Ashby, S., Beckman, P., Chen, J., Colella, P., Collins, B., Crawford, D., Dongarra, J., Kothe, D., Lusk, R., Messina, P., Mezzacappa, T., Moin, P., Norman, M., Rosner, R., Sarkar, V., Siegel, A., Streitz, F., White, A., and Wright, M. The Opportunities and Challenges of Exascale Computing. Tech. rep., 2010. [pages 65 and 72]

[6] Asifuzzaman, K., Pavlovic, M., Radulovic, M., Zaragoza, D., Kwon, O., Ryoo, K.-C., and Radojković, P. Performance Impact of a Slower Main Memory: A Case Study of STT-MRAM in HPC. In *Proc. of the International Symposium on Memory Systems (MEMSYS)* (oct 2016), pp. 40–49. [pages 4 and 20]

[7] Atkins, D. E., Droegemeier, K. K., Feldman, S. I., Feldman, S. I., Klein, M. L., Messerschmitt, D. G., Messina, P., Ostriker, J. P., and Wright, M. H. Revolutionizing Science and Engineering Through Cyberinfrastructure. Report of the National Science Foundation Blue-Ribbon Advisory Panel on Cyberinfrastructure, National Science Foundation, Jan. 2003. [page 32]

[8] Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing 1*, 1 (jan 2004), 11–33. [page 84]

[9] Barcelona Supercomputing Center. *Paraver: Parallel Program Visualization and Analysis tool*, oct 2001. [page 29]

[10] Barcelona Supercomputing Center. MareNostrum 3 System Architecture. `http://www.bsc.es/marenostrum-support-services/mn3`, 2013. [pages 23, 32, 35, and 66]

[11] Barcelona Supercomputing Center. *Extrae User guide manual for version 2.5.1*, Apr. 2014. [pages 28 and 35]

[12] Barcelona Supercomputing Center. *MareNostrum 3 User's Guide*, Apr. 2016. [pages 6, 82, and 86]

[13] Bautista-Gomez, L., Zyulkyarov, F., Unsal, O., and McIntosh-Smith, S. Unprotected Computing: A Large-scale Study of DRAM Raw Error Rate on a Supercomputer. In *Proc. of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (nov 2016), pp. 55:1–55:11. [page 6]

[14] Bienia, C., Kumar, S., Singh, J. P., and Li, K. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proc. of the International Conference on Parallel Architectures and Compilation Techniques (PACT)* (Oct. 2008), pp. 72–81. [page 48]

[15] Biswas, S., de Supinski, B. R., Schulz, M., Franklin, D., Sherwood, T., and Chong, F. T. Exploiting Data Similarity to Reduce Memory Footprints. In *Proc. of the IEEE International Parallel & Distributed Processing Symposium (IPDPS)* (May 2011), pp. 152–163. [pages 60 and 61]

[16] Black, B., Annavaram, M., Brekelbaum, N., DeVale, J., Jiang, L., Loh, G. H., McCaule, D., Morrow, P., Nelson, D. W., Pantuso, D., Reed, P., Rupley, J., Shankar, S., Shen, J., and Webb, C. Die Stacking (3D) Microarchitecture. In *Proc. of the IEEE/ACM International Symposium on Microarchitecture (MICRO)* (2006), pp. 469–479. [page 4]

[17] Bull, M. PRACE-2IP: D7.4 Unified European Applications Benchmark Suite - Final. Tech. rep., PRACE Seventh Framework Programme Research Infrastructures, 2013. [page 55]

[18] Cantalupo, C., Raman, K., and Sasanka, R. MCDRAM on 2nd Generation Intel Xeon Phi Processor (code-named Knights Landing): Analysis Methods and Tools. International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Nov. 2015. Tutorial. [page 34]

[19] Cappello, F., Al, G., Gropp, W., Kale, S., Kramer, B., and Snir, M. Toward Exascale Resilience: 2014 Update. *Supercomputing Frontiers and Innovations: an International Journal 1*, 1 (Apr. 2014), 5–28. [pages 6 and 82]

[20] Chou, C., Jaleel, A., and Qureshi, M. K. CAMEO: A Two-Level Memory Organization with Capacity of Main Memory and Flexibility of Hardware-Managed Cache. In *Proc. of the International Symposium on Microarchitecture (MICRO)* (Dec. 2014), pp. 1–12. [pages 34, 59, and 61]

[21] Dell, T. J. A White Paper on the Benefits of Chipkill-Correct ECC for PC Server Main Memory. Technical white paper 4AA4-3490ENW, IBM, Nov. 1997. [pages 16 and 82]

[22] Dell, T. J. System RAS Implications of DRAM Soft Errors. *IBM Journal of Research and Development 52*, 3 (May 2008), 307–314. [page 6]

[23] Dickov, B., Pericàs, M., Carpenter, P., Navarro, N., and Ayguadé, E. Software-Managed Power Reduction in Infiniband Links. In *Proc. of the Int. Conference on Parallel Processing (ICPP)* (Sept. 2014), pp. 311–320. [page 76]

[24] Diniz, B., Guedes, D., Meira, Jr., W., and Bianchini, R. Limiting the Power Consumption of Main Memory. In *Proc. of the Int. Symp. on Computer Architecture (ISCA)* (June 2007), pp. 290–301. [page 77]

[25] Dong, X., Xie, Y., Muralimanohar, N., and Jouppi, N. P. Simple but Effective Heterogeneous Main Memory with On-Chip Memory Controller Support. In *Proc. of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (Nov. 2010), pp. 1–11. [pages 34 and 61]

[26] Dongarra, J., Beckman, P., Moore, T., Aerts, P., Aloisio, G., Andre, J.-C., Barkai, D., Berthou, J.-Y., Boku, T., Braunschweig, B., Cappello, F., Chapman, B., Chi, X., Choudhary, A., Dosanjh, S., Dunning, T., Fiore, S., Geist, A., Gropp, B., Harrison, R., Hereld, M., Heroux, M., Hoisie, A., Hotta, K., Jin, Z., Ishikawa, Y., Johnson, F., Kale, S., Kenway, R., Keyes, D., Kramer, B., Labarta, J., Lichnewsky, A., Lippert, T., Lucas, B., Maccabe, B., Matsuoka, S., Messina, P., Michielse, P., Mohr, B., Mueller, M. S., Nagel, W. E., Nakashima, H., Papka, M. E., Reed, D., Sato, M., Seidel, E., Shalf, J., Skinner, D., Snir, M., Sterling, T., Stevens, R., Streitz, F., Sugar, B., Sumimoto, S., Tang, W., Taylor, J., Thakur, R., Trefethen, A., Valero, M., van der Steen, A., Vetter, J., Williams, P., Wisniewski, R., and Yelick, K. The International Exascale Software Project Roadmap. *International Journal of High Performance Computing Applications 25*, 1 (Feb. 2011), 3–60. [pages 2 and 5]

[27] Dongarra, J. J., and Heroux, M. A. Toward a New Metric for Ranking High Performance Computing Systems. Sandia Report SAND2013-4744, Sandia National Laboratories, June 2013. [pages 25, 42, and 44]

[28] Dongarra, J. J., Luszczek, P., and Heroux, M. A. HPCG: One Year Later. In *International Supercomputing Conference (ISC)* (June 2014). [page 43]

[29] Dongarra, J. J., Luszczek, P., and Petitet, A. The LINPACK Benchmark: Past, Present and Future. *Concurrency and Computation: Practice and Experience 15*, 9 (2003), 803–820. [pages 39, 40, and 60]

[30] Draper, J., Chame, J., Hall, M., Steele, C., Barrett, T., LaCoss, J., Granacki, J., Shin, J., Chen, C., Kang, C. W., Kim, I., and Daglikoca, G. The Architecture of the DIVA Processing-in-memory Chip. In *Proc. of the International Conference on Supercomputing (ISC)* (June 2002), pp. 14–25. [page 4]

[31] Dubey, P. Recognition, Mining and Synthesis Moves Computers to the Era of Tera. *Intel Technology Journal 9*, 2 (Feb. 2005). [pages 1 and 4]

[32] ETP4HPC. ETP4HPC Strategic Research Agenda Achieving HPC leadership in Europe, June 2013. [pages 48 and 65]

[33] EU Commission. European Commission Eurostat. `http://ec.europa.eu/eurostat/`, 2015. [page 75]

[34] Freeh, V. W., Pan, F., Kappiah, N., Lowenthal, D. K., and Springer, R. Exploring the Energy-Time Tradeoff in MPI Programs on a Power-Scalable Cluster. In *Proc. of the Int. Parallel and Distributed Processing Symp. (IPDPS)* (Apr. 2005), p. 4.a. [page 76]

[35] Geist, A. Supercomputing's Monster in the Closet. *IEEE Spectrum 53*, 3 (Mar. 2016), 30–35. [page 6]

[36] HP. How memory RAS technologies can enhance the uptime of HPE ProLiant servers. Technical white paper 4AA4-3490ENW, Hewlett Packard Enterprise, Feb. 2016. [pages 6, 16, and 81]

[37] Hybrid Memory Cube Consortium. Hybrid Memory Cube Specification 2.0. `www.hybridmemorycube.org/specification-v2-download-form/`, Nov. 2014. [pages 2 and 18]

[38] IBM. *Administering Platform LSF*, 2014. Version 9 Release 1.2. [page 67]

[39] IBM. *System x iDataPlex dx360 M4 Types 7912 and 7913: Problem Determination and Service Guide*, Apr 2014. [pages 30 and 87]

[40] IBM. IBM Integrated Management Module II Firmware 3.70. `https://www.kernel.org/doc/Documentation/hwmon/ibmaem`, 2015. [page 67]

[41] Intel. Intel VTune Amplifier 2016. https://software.intel.com/en-us/intel-vtune-amplifier-xe/, 2016. [page 59]

[42] Intel. The memkind library. http://memkind.github.io/memkind/, 2016. [page 59]

[43] Jacob, B., NG, S. W., and Wang, D. T. *Memory Systems: Cache, DRAM, Disk.* Morgan Kaufmann, 2008. [pages 3 and 84]

[44] JEDEC Solid State Technology Association. High Bandwidth Memory (HBM) DRAM. `www.jedec.org/standards-documents/docs/jesd235`, Oct. 2013. [pages 2 and 18]

[45] Jeffers, J., Reinders, J., and Sodani, A. *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition*, 2nd ed. Morgan Kaufmann, June 2016. [page 34]

[46] Kanter, D. Knights Landing Reshapes HPC. *Microprocessor Report* (Sept. 2015). [page 21]

[47] Kleen, A. MCELOG: Memory Error Handling in User Space. In *International Linux System Technology Conference (Linux Kongress)* (Sep 2010). [pages 30 and 87]

[48] Kleymenov, A., and Park, J. HPCG on Intel Xeon Phi 2nd Generation, Knights Landing. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (Nov 2016). HPCG BoF. [page 58]

[49] Kogge, P., Bergman, K., Borkar, S., Campbell, D., Carlson, W., Dally, W., Denneau, M., Franzon, P., Harrod, W., Hill, K., Hiller, J., Karp, S., Keckler, S., Klein, D., Lucas, R., Richards, M., Scarpelli, A., Scott, S., Snavely, A., Sterling, T., Williams, R. S., and Yelick, K. ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems, Sept. 2008. [pages 1, 4, and 32]

[50] Koop, M. J., Jones, T., and Panda, D. K. Reducing Connection Memory Requirements of MPI for InfiniBand Clusters: A Message Coalescing Approach. In *Proc. of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID)* (May 2007), pp. 495–504. [page 45]

[51] Kultursay, E., Kandemir, M., Sivasubramaniam, A., and Mutlu, O. Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)* (Apr 2013). [page 4]

[52] Laros, J. H., Pedretti, K. T., Kelly, S. M., Shu, W., and Vaughan, C. T. Energy Based Performance Tuning for Large Scale High Performance Computing Systems. In *Proc. of the Symp. on High Performance Computing (HPC)* (Mar. 2012), pp. 6:1–6:10. [page 76]

[53] Lee, B. C., Ipek, E., Mutlu, O., and Burger, D. Architecting Phase Change Memory As a Scalable Dram Alternative. In *Proc. of the International Symposium on Computer Architecture (ISCA)* (June 2009), pp. 2–13. [page 4]

[54] Lee, D., Ghose, S., Pekhimenko, G., Khan, S., and Mutlu, O. Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost. *ACM Transactions on Architecture and Code Optimization (TACO) 12*, 4 (Jan. 2016), 63:1–63:29. [page 4]

[55] Lenovo. System x iDataPlex dx360 M4 Product Guide. `https://lenovopress.com/tips0878`, Jan. 2015. [page 67]

[56] Li, X., Huang, M. C., Shen, K., and Chu, L. A Realistic Evaluation of Memory Hardware Errors and Software System Susceptibility. In *Proc. of the USENIX Conference on USENIX Annual Technical Conference (USENIXATC)* (Jun 2010), pp. 6–6. [pages 87, 91, and 106]

[57] Lim, M. Y., Freeh, V. W., and Lowenthal, D. K. Adaptive, Transparent Frequency and Voltage Scaling of Communication Phases in MPI Programs. In *Proc. of the Conference on Supercomputing (SC)* (Nov. 2006). [page 76]

[58] Locklear, D. Chipkil Correct Memory Architecture. *DELL Technology Brief* (Aug. 2000). [page 16]

[59] Loh, G. H. 3D-Stacked Memory Architectures for Multi-core Processors. In *Proc. of the International Symposium on Computer Architecture (ISCA)* (2008), pp. 453–464. [page 4]

[60] Luszczek, P., and Dongarra, J. J. Introduction to the HPC Challenge Benchmark Suite. ICL Technical Report ICL-UT-05-01, University of Tennessee, 2005. [page 48]

[61] Malladi, K. T., Lee, B. C., Nothaft, F. A., Kozyrakis, C., Periyathambi, K., and Horowitz, M. Towards Energy-proportional Datacenter Memory with Mobile DRAM. In *Proc. of the Int. Symp. on Computer Architecture (ISCA)* (June 2012), pp. 37–48. [page 77]

[62] Malladi, K. T., Shaeffer, I., Gopalakrishnan, L., Lo, D., Lee, B. C., and Horowitz, M. Rethinking DRAM Power Modes for Energy Proportionality. In *Proc. of the Int. Symp. on Microarchitecture (MICRO)* (Dec. 2012), pp. 131–142. [page 77]

[63] Marjanović, V., Garcia, J., and Glass, C. W. Performance Modeling of the HPCG Benchmark. In *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation* (Nov. 2014), Springer International Publishing, pp. 172–192. [pages 43, 44, 45, and 60]

[64] Martino, C. D., Kalbarczyk, Z., Iyer, R. K., Baccanico, F., Fullop, J., and Kramer, W. Lessons Learned from the Analysis of System Failures at Petascale: The Case of Blue Waters. In *Proc. of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (Jun 2014), pp. 610–621. [page 105]

[65] McCalpin, J. D. Memory Bandwidth and Machine Balance in Current High Performance Computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter* (Sept. 1995). [page 4]

[66] McCalpin, J. D. STREAM: Sustainable Memory Bandwidth in High Performance Computers. `https://www.cs.virginia.edu/stream/ref.html`, 1997. [page 24]

[67] McKee, S. A. Reflections on the Memory Wall. In *Proc. of the 1st Conference on Computing Frontiers* (Apr. 2004), pp. 162–. [page 1]

[68] McVoy, L., and Staelin, C. LMbench: Portable Tools for Performance Analysis. In *Proc. of the Annual Conference on USENIX Annual Technical Conference* (1996). [page 24]

[69] Meswani, M. R., Blagodurov, S., Roberts, D., Slice, J., Ignatowski, M., and Loh, G. H. Heterogeneous Memory Architectures: A HW/SW Approach for Mixing Die-stacked and Off-package Memories. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)* (Feb. 2015), pp. 126–136. [pages 4, 34, 59, and 61]

[70] Meza, J., Wu, Q., Kumar, S., and Mutlu, O. Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field. In *Proc. of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (Jun 2015), pp. 415–426. [pages 91 and 107]

[71] Murphy, R. On the Effects of Memory Latency and Bandwidth on Supercomputer Application Performance. In *Proc. of the IEEE International Symposium on Workload Characterization (IISWC)* (Nov. 2007), pp. 35–43. [page 1]

[72] Murphy, R., Berry, J., McLendon, W., Hendrickson, B., Gregor, D., and Lumsdaine, A. DFS: A Simple to Write Yet Difficult to Execute Benchmark. In *IEEE International Symposium on Workload Characterization (IISWC)* (Oct. 2006), pp. 175–177. [page 58]

[73] Murphy, R., Wheeler, K., Barrett, B., and Ang, J. Introducing the Graph 500. Cray User's Group (CUG), May 2010. [page 58]

[74] NERSC. Large Scale Computing and Storage Requirements for High Energy Physics: Target 2017. Report of the NERSC Requirements Review, Lawrence Berkeley National Laboratory, Nov. 2012. [page 32]

[75] NERSC. Large Scale Computing and Storage Requirements for Biological and Environmental Science: Target 2017. Report of the NERSC Requirements Review LBNL-6256E, Lawrence Berkeley National Laboratory, June 2013. [page 32]

[76] NERSC. High Performance Computing and Storage Requirements for Basic Energy Sciences: Target 2017. Report of the HPC Requirements Review LBNL-6978E, Lawrence Berkeley National Laboratory, Oct. 2014. [page 32]

[77] NERSC. Large Scale Computing and Storage Requirements for Fusion Energy Sciences: Target 2017. Report of the NERSC Requirements Review LBNL-6631E, Lawrence Berkeley National Laboratory, May 2014. [page 32]

[78] NERSC. High Performance Computing and Storage Requirements for Nuclear Physics: Target 2017. Report of the NERSC Requirements Review LBNL-6926E, Lawrence Berkeley National Laboratory, Jan. 2015. [page 32]

[79] NERSC. Large Scale Computing and Storage Requirements for Advanced Scientific Computing Research: Target 2017. Report of the NERSC Requirements Review LBNL-6978E, Lawrence Berkeley National Laboratory, Apr. 2015. [page 32]

[80] Newburn, C. J. Code for the future: Knights Landing and beyond. International Supercomputing Conference (ISC), July 2015. IXPUG Workshop. [page 34]

[81] P. Reviriego et al. An Initial Evaluation of Energy Efficient Ethernet. *IEEE Communications Letters 15*, 5 (May 2011), 578–580. [page 67]

[82] Park, J., Smelyanskiy, M., Vaidyanathan, K., Heinecke, A., Kalamkar, D. D., Liu, X., Patwary, M. M. A., Lu, Y., and Dubey, P. Efficient Shared-memory Implementation of High-performance Conjugate Gradient Benchmark and Its Application to Unstructured Matrices. In *Proc. of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (Nov. 2014), pp. 945–955. [page 44]

[83] Pavlovic, M., Etsion, Y., and Ramirez, A. On the Memory System Requirements of Future Scientific Applications: Four Case-studies. In *Proc. of the IEEE International Symposium on Workload Characterization (IISWC)* (Nov. 2011), pp. 159–170. [pages 60 and 61]

[84] Pavlovic, M., Puzovic, N., and Ramirez, A. Data Placement in HPC Architectures with Heterogeneous Off-chip Memory. In *IEEE International Conference on Computer Design (ICCD)* (Oct. 2013), pp. 193–200. [page 2]

[85] Pavlovic, M., Radulovic, M., Ramirez, A., and Radojkovic, P. Limpio - LIghtweight MPI instrumentatiOn. In *Proc. of the Int. Conference on Program Comprehension (ICPC)* (May 2015), https://www.bsc.es/computer-sciences/computer-architecture/memory-systems/limpio, pp. 303–306. [pages 30, 35, and 71]

[86] Perks, O., Hammond, S., Pennycook, S. J., and Jarvis, S. A. Should We Worry About Memory Loss? *SIGMETRICS Performance Evaluation Review 38*, 4 (Mar. 2011), 69–74. [pages 60 and 61]

[87] Petitet, A., Whaley, R. C., Dongarra, J., and Cleary, A. HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers. `http://www.netlib.org/benchmark/hpl/`, Sept. 2008. [pages 36, 39, 40, and 41]

[88] PinnacleMicro. `http://www.pinnaclemicro.com/`, nov 2015. [page 75]

[89] PRACE. Unified European Applications Benchmark Suite. www.prace-ri.eu/ueabs/, 2013. [pages 7, 25, 33, 35, 45, 52, 55, 66, and 68]

[90] PRACE Research Infrastructure. `http://www.prace-ri.eu`, 2015. [pages 23, 26, 35, and 66]

[91] Radulovic, M., Zivanovic, D., Ruiz, D., de Supinski, B. R., McKee, S. A., Radojković, P., and Ayguadé, E. Another Trip to the Wall: How Much Will Stacked DRAM Benefit HPC? In *Proc. of the International Symposium on Memory Systems (MEMSYS)* (2015), pp. 31–36. [page 59]

[92] Rajovic, N., Carpenter, P. M., Gelado, I., Puzovic, N., Ramirez, A., and Valero, M. Supercomputing with Commodity CPUs: Are Mobile SoCs Ready for HPC? In *Proc. of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)* (2013), pp. 40:1–40:12. [page 113]

[93] Ranganathan, P. From Microprocessors to Nanostores: Rethinking Data-Centric Systems. *Computer 44*, 1 (Jan. 2011), 39–48. [page 5]

[94] Saleh, A. M., Serrano, J. J., and Patel, J. H. Reliability of Scrubbing Recovery-Techniques for Memory Systems. *IEEE transactions on reliability 39*, 1 (1990), 114–122. [page 87]

[95] Saravanan, K. P., Carpenter, P. M., and Ramirez, A. A Performance Perspective on Energy Efficient HPC Links. In *Proc. of the Int. Conference on Supercomputing (ICS)* (June 2014), pp. 313–322. [page 77]

[96] Schroeder, B., and Gibson, G. A Large-Scale Study of Failures in High-Performance Computing Systems. *IEEE Transactions on Dependable and Secure Computing 7*, 4 (Oct 2010), 337–350. [page 105]

[97] Schroeder, B., and Gibson, G. A. Understanding Failures in Petascale Computers. *Journal of Physics: Conference Series 78* (2007). [page 6]

[98] Schroeder, B., Pinheiro, E., and Weber, W.-D. DRAM Errors in the Wild: A Large-scale Field Study. In *Proc. of the International Joint Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)* (Jun 2009), pp. 193–204. [pages 91, 106, and 107]

[99] Siddiqua, T., Papathanasiou, A., Biswas, A., and Gurumurthi, S. Analysis and Modelling of Memory Errors from Large-Scale Field Data Collection. In *IEEE Workshop on Silicon Errors in Logic - System Effects (SELSE)* (Mar 2013). [pages 91 and 107]

[100] Sim, J., Alameldeen, A. R., Chishti, Z., Wilkerson, C., and Kim, H. Transparent Hardware Management of Stacked DRAM As Part of Memory. In *Proc. of the International Symposium on Microarchitecture (MICRO)* (Dec. 2014), pp. 13–24. [pages 34 and 61]

[101] Sites, R. It's the Memory, Stupid! *Microprocessor Report 10*, 10 (Aug. 1996), 2–3. [page 1]

[102] Sodani, A. Race to Exascale: Opportunities and Challenges. International Symposium on Microarchitecture (MICRO), Dec. 2011. Keynote. [pages 1 and 32]

[103] Sodani, A., Gramunt, R., Corbal, J., Kim, H.-S., Vinod, K., Chinthamani, S., Hutsell, S., Agarwal, R., and Liu, Y.-C. Knights Landing: Second-Generation Intel Xeon Phi Product. *IEEE Micro 36*, 2 (Mar. 2016), 34–46. [pages 19, 20, and 34]

[104] Song, S. L., Barker, K., and Kerbyson, D. Unified Performance and Power Modeling of Scientific Workloads. In *Proc. of the Int. Workshop on Energy Efficient Supercomputing (E2SC)* (Nov. 2013), pp. 4:1–4:8. [page 67]

[105] Sridharan, V., DeBardeleben, N., Blanchard, S., Ferreira, K. B., Stearley, J., Shalf, J., and Gurumurthi, S. Memory Errors in Modern Systems: The Good, The Bad, and The Ugly. In *Proc. of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (Mar 2015), pp. 297–310. [pages 87, 91, 92, and 107]

[106] Sridharan, V., and Liberty, D. A Study of DRAM Failures in the Field. In *Proc. of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)* (Nov 2012), pp. 76:1–76:11. [pages 87, 91, 92, 106, and 107]

[107] Sridharan, V., Stearley, J., DeBardeleben, N., Blanchard, S., and Gurumurthi, S. Feng Shui of Supercomputer Memory: Positional Effects in DRAM and SRAM Faults. In *Proc. of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)* (Nov 2013), pp. 22:1–22:11. [pages 87, 91, 92, 106, and 107]

[108] Stevens, R., and White, A. Architectures and Technology for Extreme Scale Computing. Tech. rep., DOE, dec 2009. [page 65]

[109] Stevens, R., White, A., Beckman, P., Bair-ANL, R., Hack, J., Nichols, J., GeistORNL, A., Simon, H., Yelick, K., Shalf-LBNL, J., Ashby, S., Khaleel-PNNL, M., McCoy, M., Seager, M., Gorda-LLNL, B., Morrison, J., Wampler-LANL, C., Peery, J., Dosanjh, S., Ang-SNL, J., Davenport, J., Schlagel, T., BNL, Johnson, F., and Messina, P. 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. [pages 1, 4, and 32]

[110] Suresh, A., Cicotti, P., and Carrington, L. Evaluation of Emerging Memory Technologies for HPC, Data Intensive Applications. In *International Conference on Cluster Computing (CLUSTER)* (Sep 2014). [page 2]

[111] The HPCG Benchmark. `http://www.hpcg-benchmark.org/`, 2016. [pages 25, 42, and 43]

[112] Tolentino, M. E., Turner, J., and Cameron, K. W. Memory MISER: Improving Main Memory Energy Efficiency in Servers. *IEEE Transactions on Computers 58*, 03 (Sept. 2008), 336–350. [page 77]

[113] TOP500 List. `http://www.top500.org/`, Nov. 2017. [pages 25, 36, and 39]

[114] Torrellas, J. FlexRAM: Toward an Advanced Intelligent Memory System: A Retrospective Paper. In *Proc. of the International Conference on Computer Design (ICCD)* (Sept. 2012), pp. 3–4. [page 4]

[115] U.S. Energy Information Administration. Electric Power Monthly with Data for August 2015. Tech. rep., DOE, Oct. 2015. [page 75]

[116] Wong, F. C., Martin, R. P., Arpaci-Dusseau, R. H., and Culler, D. E. Architectural Requirements and Scalability of the NAS Parallel Benchmarks. In *Proc. of the ACM/IEEE Conference on Supercomputing (SC)* (Jan. 1999). [page 48]

[117] Wong, H.-S. P., Raoux, S., Kim, S., Liang, J., Reifenberg, J. P., Rajendran, B., Asheghi, M., and Goodson, K. E. Phase Change Memory. *Proceedings of the IEEE 98*, 12 (Dec 2010), 2201–2227. [pages 4 and 20]

[118] Woo, S. C., Ohara, M., and Torrie, E. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proc. of the International Symposium on Computer Architecture (ISCA)* (July 1995), pp. 24–36. [page 48]

[119] Wulf, W., and McKee, S. Hitting the Memory Wall: Implications of the Obvious. *ACM SIGARCH Computer Architecture News 23*, 1 (Mar. 1995), 20–24. [page 3]

[120] Xie, Yuan. Modeling, Architecture, and Applications for Emerging Memory Technologies. *IEEE Design & Test 28*, 1 (Jan. 2011), 44–51. [page 2]

[121] Zivanovic, D., Pavlovic, M., Radulovic, M., Shin, H., Son, J., Mckee, S. A., Carpenter, P. M., Radojković, P., and Ayguadé, E. Main Memory in HPC: Do We Need More or Could We Live with Less? *ACM Transactions on Architecture and Code Optimization 14*, 1 (Apr. 2017), 3:1–3:26. [page 7]

[122] Zivanovic, D., Radulovic, M., Llort, G., Zaragoza, D., Strassburg, J., Carpenter, P. M., Radojković, P., and Ayguadé, E. Large-Memory Nodes for Energy Efficient High-Performance Computing. In *International Symposium on Memory Systems (MEMSYS)* (Oct. 2016). [pages 7 and 52]

Glossary

BQCD	Berlin Quantum Chromo-Dynamics.
CE	Corrected Error.
CMP	Chip Multiprocessors.
CPU	Central Processing Unit.
DARPA	The Defense Advanced Research Projects Agency.
DDR	Double Data Rate Dynamic Random-Access Memory.
DIMM	Dual In-line Memory Modules.
DOE	US Department of Energy.
DRAM	Dynamic Random-Access Memory.
DVFS	Dynamic Voltage and Frequency Scaling.
ECC	Error-Correcting Code.
EDP	Energy-Delay Product.
FLOP	Floating-Point Operation.
FLOP/s	Floating-Point Operations per second.
GADGET	GAlaxies with Dark matter and GasintEracT.
GPU	Graphics Processing Unit.
GROMACS	GROningen MAchine for Chemical Simulations.
GUI	Graphical User Interface.
HBC	Hybrid Memory Cube.
HBM	High-Bandwidth Memory.
HPC	High-Performance Computing.
HPCG	High-Performance Conjugate Gradients.
HPL	High-Performance Linpack.
KNL	Knights Landing.

LSF	Load Sharing Facility.
MCA	Machine-Check Architecture.
MCDRAM	Multi-Channel Dynamic Random-Access Memory.
MPI	Message Passing Interface.
MTBF	Mean Time Between Failures.
NAMD	Nanoscale Molecular Dynamics.
NEMO	Nucleus for European Modelling of the Ocean.
NVM	Non-Volatile Memory.
PCM	Phase Change Memory.
PRACE	Partnership for Advanced Computing in Europe.
QE	Quantum Espresso.
RAM	Random-Access Memory.
RLDRAM	Reduced Latency Dynamic Random-Access Memory.
RRAM	Resistive Random-Access Memory.
SEC	Single-Error Correction.
SECDED	Single-Error Correction Double-Error Detection.
STT-MRAM	Spin-Transfer Torque Magnetoresistive Random-Access Memory.
TSV	Through-Silicon Via.
UE	Uncorrected Error.
UEABS	Unified European Application Benchmark Suite.