Main Memory in HPC: Do We Need More or Could We Live with Less?

DARKO ZIVANOVIC, MILAN PAVLOVIC, and MILAN RADULOVIC, Barcelona Supercomputing Center (BSC), Universitat Politècnica de Catalunya
HYUNSUNG SHIN and JONGPIL SON, Samsung Electronics Co., Ltd., Memory Division
SALLY A. MCKEE, Chalmers University of Technology
PAUL M. CARPENTER and PETAR RADOJKOVIĆ, Barcelona Supercomputing Center (BSC)
EDUARD AYGUADÉ, Barcelona Supercomputing Center (BSC), Universitat Politècnica de Catalunya

An important aspect of High-Performance Computing (HPC) system design is the choice of main memory capacity. This choice becomes increasingly important now that 3D-stacked memories are entering the market. Compared with conventional Dual In-line Memory Modules (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.

This study analyzes the memory capacity requirements of important HPC benchmarks and applications. We find that the High-Performance Conjugate Gradients (HPCG) benchmark could be an important success story for 3D-stacked memories in HPC, but High-Performance Linpack (HPL) 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 toward adoption of this novel technology in the HPC domain.

CCS Concepts: • **Computer systems organization → Distributed architectures**; • **Hardware → Analysis and design of emerging devices and systems**; **Memory and dense storage;**

Additional Key Words and Phrases: Memory capacity requirements, high-performance computing, production HPC applications, HPL, HPCG

This work was supported by the Collaboration Agreement between Samsung Electronics Co., Ltd. and BSC, 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). Darko Zivanovic holds the Severo Ochoa grant (SVP-2014-068501) of the Ministry of Economy and Competitiveness of Spain.

Authors' addresses: D. Zivanovic, M. Pavlovic, M. Radulovic, P. M. Carpenter, P. Radojković, and E. Ayguadé, Barcelona Supercomputing Center, Jordi Girona 1-3, K2M 102, 08034, Barcelona, Spain; emails: {darko.zivanovic, milan.pavlovic, milan.radulovic, paul.carpenter, petar.radojkovic, eduard.ayguade}@bsc.es; H. Shin and J. Son, Samsung Electronics Co., Ltd., Memory Division, 1-1, Samsungjeonja-ro, Hwaseong-si, Gyeonggi-do 445-701 Korea; emails: {hyunsung.shin, jp.son}@samsung.com; S. A. McKee, Chalmers University of Technology, Rännvägen 6, 4th floor, E/D/IT (D&IT), 41296 Göteborg, Sweden; email: mckee@chalmers.se; M. Pavlovic is currently at ASML, Eindhoven, Netherlands.
Correspondence email: darko.zivanovic@bsc.es.

DOI: http://dx.doi.org/10.1145/3023362

ACM Reference Format:
Darko Zivanovic, Milan Pavlovic, Milan Radulovic, Hyunsung Shin, Jongpil Son, Sally A. McKee, Paul M. Carpenter, Petar Radojković, and Eduard Ayguadé. 2017. Main memory in HPC: Do we need more or could we live with less? ACM Trans. Archit. Code Optim. 14, 1, Article 3 (March 2017), 26 pages.
DOI: http://dx.doi.org/10.1145/3023362

1. INTRODUCTION

Memory systems are important contributors to the deployment and operational costs of large-scale High-Performance Computing (HPC) clusters [Kogge et al. 2008; Stevens et al. 2010; Sodani 2011], 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 1GB per Blue Gene PowerPC core (see Figure 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 [Atkins et al. 2003; NERSC 2012; 2013, 2014a, 2014b, 2015a, 2015b], there are no or very few studies that thoroughly analyze and quantify the memory footprints of HPC workloads across multiple domains. In this article, we try to bridge this gap and to examine whether current memory design strategies meet the memory requirements of important 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, at least 2GB of main memory is required 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 7.6GB and 16.1GB in a million-core cluster). In contrast, HPCG memory requirements are fundamentally different. To converge to the optimal performance, the benchmark requires roughly 0.5GB 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 [PRACE 2013]. 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

[1] In our system, the MareNostrum supercomputer [Barcelona Supercomputing Center 2013], main memory accounts to more than 10% of server cost and 10–15% of server energy consumption.

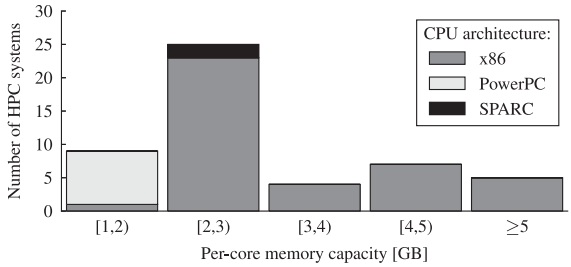


Fig. 1. Per-core memory capacity of HPC systems leading the TOP500 list (June 2015). Systems with exactly 2, 3, and 4GB of memory per core are included in bars [2,3) GB, [3,4) GB, and [4,5) GB, respectively. Today's HPC systems are dominated by x86 architectures coupled with 2–3GB of main memory per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

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 Dual In-line Memory Modules (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 [Sodani et al. 2016]. 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 Dynamic Random Access Memory (DRAM) with standard DIMMs [Dong et al. 2010; Chou et al. 2014; Sim et al. 2014; Meswani et al. 2015; Sodani et al. 2016]. 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 [Newburn 2015; Cantalupo et al. 2015; Jeffers et al. 2016].

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 toward adoption of this novel technology in the HPC domain.

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.

2.1. Hardware Platform

We execute all experiments on the MareNostrum supercomputer [Barcelona Supercomputing Center 2013], one of six Tier-0 HPC systems in the Partnership for Advanced Computing in Europe (PRACE) [PRACE 2016]. MareNostrum contains 3,056 compute nodes connected with InfiniBand. Each node contains two Intel Sandy Bridge-EP E5-2670 sockets, each comprising eight 2.6GHz cores. As in most HPC systems, hyperthreading is disabled. The interconnect is InfiniBand FDR-10 (40Gb/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.

The processors connect to main memory through four channels, each with a single DDR3-1600 DIMM. Regular MareNostrum compute nodes include 32GB of DRAM memory (8DIMMs $\times$ 4GB), i.e., 2GB per core. To study application memory footprints in systems with higher capacity, we execute some experiments on large-memory nodes containing 128GB of DRAM (8DIMMs $\times$ 16GB), i.e., 8GB per core.

2.2. Applications

We study memory capacity requirements for two widely used HPC benchmarks, HPL and HPCG. We also study the UEABS [PRACE 2013], the set of production applications, and datasets designed for benchmarking the European PRACE HPC systems for procurement and comparison purposes [PRACE 2016]. 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 article, *per-process* and *per-core* memory footprint have equivalent meanings.

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 [Barcelona Supercomputing Center 2014; Pavlovic et al. 2015] to read and log VmRSS values at equidistant time intervals chosen to provide at least 1,000 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.

3. HPL

For more than 20 years, the HPL benchmark is the most widely recognized and discussed metric for the ranking of HPC systems [Strohmaier et al. 2015]. 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

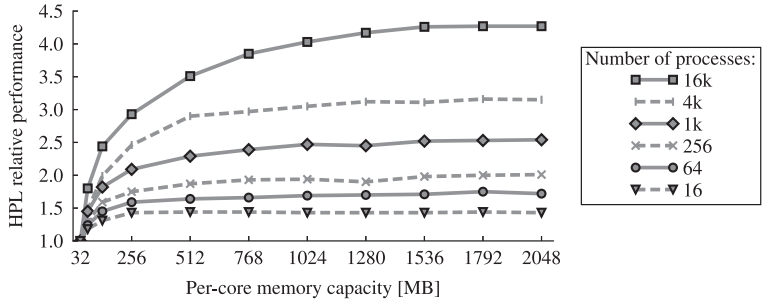


Fig. 2. HPL performance also depends on the available memory capacity. When increasing per-core memory, HPL performance first increases and then reaches the saturation point—the GFLOP/s of the system. As we increase the number of processes, the saturation point moves toward larger per-core memory capacities.

achieve the best performance on a given system. The documentation recommends setting a problem size that uses approximately 80% of the available memory [Petitet et al. 2012]. 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.

3.1. Measured Memory Requirements

Figure 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 32MB 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 1,024 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 toward larger per-core memory capacities; for instance, 16, 256, and 4,096 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 1.5GB of memory per core. 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 2GB of memory per core. This matches the main memory sizing trends of the large HPC systems that dominate the TOP500 list.

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

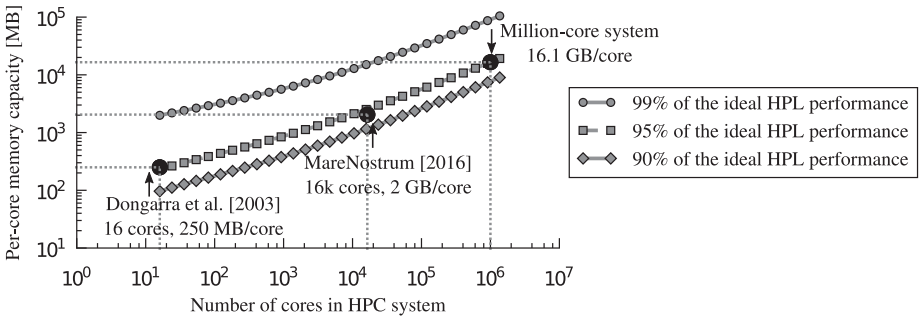


Fig. 3. Per-core memory needed to get 90%, 95%, and 99% of the ideal HPL performance for different sizes of HPC systems. Three systems, with 250MB/core (year 2003), 2GB/core (year 2016), and 16.1GB/core (future potential system) are compared with our estimation curves. Moving from 95% to 99% of ideal HPL performance requires a huge step in the amount of per-core memory (axes are plotted on logarithmic scales).

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 the Appendix.

Our analysis confirms the HPL performance trends presented in Figure 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 multiplication 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 Figure 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 3.

Figure 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

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

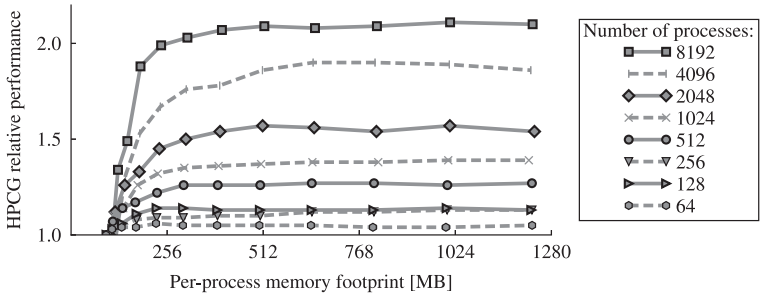


Fig. 4. HPCG performance also depends on the available memory capacity. Performance increases until it reaches the saturation point, where it is constrained by the sustained memory bandwidth. The saturation point remains constant across a wide range of HPCG processes, at around 512MB of main memory per process.

cores, which would *further increase* the per-core memory requirement for large systems. From Figure 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 3 have a logarithmic scale). For systems with 100,000 cores, 2.6GB and 5.5GB 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, 7.6GB and 16.1GB 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 31.4GB and a system with 1,000,000 cores would require 88.6GB per core, a huge amount.

To put our estimation into perspective on Figure 3, we plot three systems: the 16-core system used in the first study that analyzed HPL memory capacity requirements [Dongarra et al. 2003], our largest experiment with 16,384 cores on MareNostrum supercomputer (year 2016), and a future potential 1,000,000-core system. 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. HPCG

The HPCG benchmark [Dongarra et al. 2016] 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 [Dongarra and Heroux 2013]. 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.

4.1. Measured Memory Requirements

In Figure 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.

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 [Marjanović et al. 2014], and it is considered to be a non-representative use of HPCG [Dongarra et al. 2014, 2016]. 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.[4] Unlike HPL, we detect that the saturation point remains constant, roughly 512MB 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 1.5GB, 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 8GB of main memory per core. The HPCG performance drop because of memory swapping was not reported in the past, and we would suggest that HPCG developers take this into account when providing suggestions about dataset sizing.

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 [Marjanović et al. 2014; Park et al. 2014]. 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.

$$\#FLOPs_{\text{per\_process}} \approx \mathcal{O}(N). \tag{1}$$

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

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

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 256MB per process in Figure 4, the HPCG performance is affected by the interprocess communication, factors $\mathcal{O}(N^{\frac{2}{3}})$ and $\mathcal{O}(\log n)$ in Equation (2). 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 [Dongarra and Heroux 2013] 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

[4]As for HPL, this trend is not visible in Figure 4 because for each number of processes, the performance is normalized to the result with the smallest input dataset.

1,[5] 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 4.1.

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 toward the larger per-process memory capacity, i.e., toward the right in Figure 4. Marjanović et al. [2014] 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%.

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 UEABS [PRACE 2013], which has been designed to represent production applications running on large-scale Tier-1 and Tier-0 HPC systems in Europe.[6] Parallelized using the Message Passing Interface (MPI), these applications are regularly executed on hundreds to thousands of cores. Most of the applications come with two input datasets. Smaller datasets (Test Case A) are deemed suitable for Tier-1 systems up to about 1,000 cores, and larger datasets (Test Case B) target Tier-0 systems up to about 10,000 cores. For Berlin Quantum Chromo-Dynamics (BQCD), GADGET, and NEMO, a single dataset (Test Case A) is provided that is suitable for both system sizes. Table I summarizes the applications and input datasets. For each application, we show the area of science that it targets, 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 8,192, except for Quantum Espresso (QE), which reports errors when executing on 4,096 or 8,192 cores. Note that SPECFEM3D always runs with the specified numbers of cores: 864 in Test Case A and 11,616 in Test Case B.

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

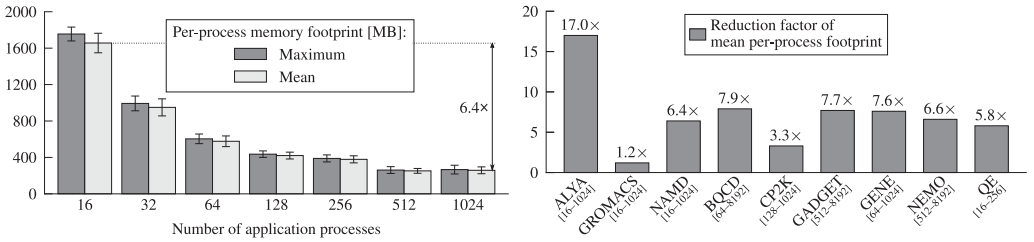
[5]Our node comprises two 8-core Sandy Bridge sockets and each core can execute up to 8 double-precision FLOPs per cycle. Each socket has four 64-bit wide, 1.6GHz memory channels. Therefore, byte/FLOP $= \frac{8\times(8\text{-bytes})\times(1.6\text{-GHz})}{16\times(8\text{-FLOP})\times(3\text{-GHz})}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

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

Table I. Scientific HPC Applications Used in the Study

		Test Case A: Smaller input dataset		Test Case B: Larger input dataset	
Application	Area of science	Problem size	Process range	Problem size	Process range
ALYA	Computational mechanics	27 million element mesh	16–1k	552.9 million element mesh	256–8k
BQCD[a]	Particle physics	$32^2 \times 64^2$ lattice	64–8k	N/A	N/A
CP2K	Computational chemistry	Energy calculation of 1,024 waters	128–1k	216 LiH system with Hartree-Fock exchange	∅[b]
GADGET	Astronomy and cosmology	135 million particles	512–8k	N/A	N/A
GENE	Plasma physics	Ion-scale turbulence in Asdex-Upgrade	64–1k	Ion-scale turbulence in Jet	2k–8k
GROMACS	Computational chemistry	150,000 atoms	16–1k	3.3 million atoms	16–8k
NAMD	Computational chemistry	$2 \times 2 \times 2$ replication of the STM Virus	16–1k	4×4×4 replication of the STM Virus	64–8k
NEMO	Ocean modeling	12° global configuration; 4322×3059 grid	512–8k	N/A	N/A
QE	Computational chemistry	112 atoms; 21 iterations	16–1k	1,532 atoms; two iterations	1k–2k
SPECFEM3D	Computational geophysics	$6 \times 12 \times 768$ 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.



(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.

Fig. 5. Per-process memory footprints shrink as the number of processes increases.

communication buffers [Koop et al. 2007]. Although it may seem obvious that memory footprints of HPC applications are tightly-coupled with the number of application processes, previous studies of memory footprints *ignore this relationship* (see Section 7).

Figure 5(a) 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 1656MB. As we increase the number of processes, the footprint drops significantly. When the application runs using 1,024 processes, the per-process memory footprint is only 258-MB, a difference of 6.4×. Maximum memory footprints exhibit the same trends (see Figure 5(a)), and thus, in the rest of the article, we discuss only mean footprint values.

Figure 5 summarizes memory footprint results for all UEABS workloads. Except GROMACS, which has a small overall memory footprint,[7] 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.

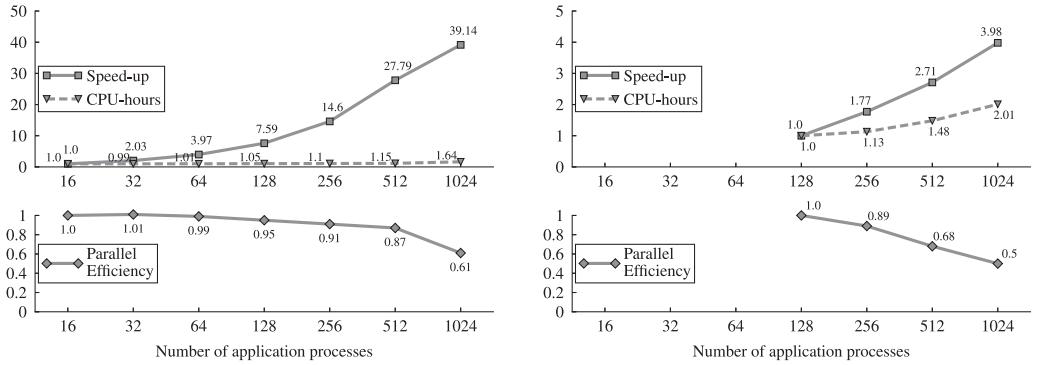
5.1.1. Discussion. Our analysis emphasizes that the memory footprints of HPC applications are tightly-coupled with the number of application processes. State-of-the-art parallel benchmark suites, however, do not strictly define the number of processes to use in experiments. UEABS recommends experiments 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 [SPEC 2015b], SPEC MPI2007 [SPEC 2015a], SPLASH-2 [Woo et al. 1995]) or do not discuss this issue at all (NAS [Wong et al. 1999], PARSEC [Bienia et al. 2008], HPC Challenge [Luszczek and Dongarra 2005], Berkeley dwarfs [Asanovic et al. 2006]). 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.

5.2. Selecting the Number of Processes

5.2.1. HPC Categories. High-performance computing is broadly divided into two categories [ETP4HPC 2013]. *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 article, 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: $cost \propto CPU\_hours = \#cores \times exe\_time$.

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 an unacceptable impact, e.g., on the productivity of engineers waiting for simulation results.

[7]The GROMACS developers explain that the application requires only around 100MB 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.



(a) NAMD (Test Case A), good scalability (b) CP2K (Test Case A), limited scalability

Fig. 6. Tradeoffs between normalized execution time and experiment cost (CPU-hours) for applications with good and limited scalability.

5.2.2. 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 tradeoffs between them. In Figure 6, we analyze this relationship for NAMD and CP2K, respectively. The upper graphs of Figure 6(a) and (b) 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.[8] 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 one 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 6(a)). 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 1,024, 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 1,024 processes in the experiments presented in Figure 6(a)). 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 (e.g., 42-fold) 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.

CP2K is an example of an application with limited scalability (Figure 6(b)). When we change the number of processes from 128 to 256, 512, and 1,024, 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 tradeoffs between cost and speed-up. When used in capability computing, CP2K should be executed with a large

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

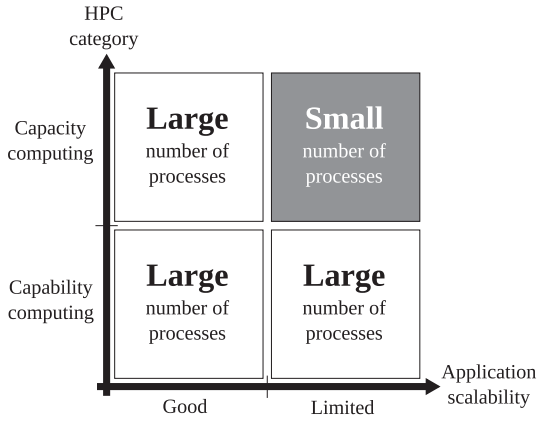


Fig. 7. The representative number of application processes is determined by application scalability and the targeted HPC category.

number of processes, 512 and 1,024 in the experiments presented in Figure 6(b). 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.

5.2.3. 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 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 tradeoff 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 [Zivanovic et al. 2016].

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 8,192 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 6 out of 10 applications, see Section 2.2). We analyze Test Case B in more detail in Section 5.4. Recall that our applications roughly scale up to 1,000–10,000 processes when running these input datasets.

The results are summarized in Figure 8. The left side of Figure 8 shows results for the applications with good scalability—ALYA, BQCD, GENE, NAMD, and 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

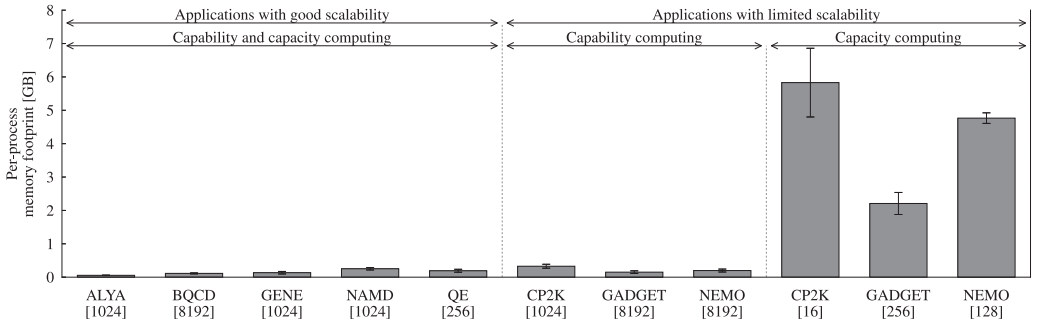


Fig. 8. Memory footprints of production HPC applications depend on application scalability and the targeted HPC category. Only the applications with limited scalability that target capacity computing require gigabytes of main memory per process.

applications range from 57MB for ALYA to 258MB for NAMD. The footprints for BQCD, GENE, and QE are 116MB, 137MB, and 197MB, respectively.[9]

The right side of Figure 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: 1,024 for CP2K, and 8,192 for GADGET and NEMO. The per-process memory footprints are again fairly small—336MB, 154MB, and 203MB 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 0.4GB (ALYA, 8 $\times$ 57MB) and 2.6GB (CP2K, 8 $\times$ 336MB). The first-generation 3D memory devices already provide such memory capacities (see Section 6.1).

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 128GB of main memory, i.e., 8GB per core. When we partition CP2K, GADGET, and NEMO to 16, 256, and 128 processes, their per-process footprints are 5.8GB, 2.2GB, and 4.8GB, respectively (rightmost part of Figure 8). On standard nodes with 2GB 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 1.1GB, 1.2GB, and 1.3GB, for CP2K, GADGET, and NEMO, respectively.

Figure 8 omits results for GROMACS and SPECFEM3D. The per-process footprint of GROMACS is very low, between 60MB and 70MB, and it decreases only slightly from 16 to 1,024 processes (see Section 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 2.53GB.

These results show that different production HPC applications—or even 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 57MB to 258MB of memory, which means that they heavily under-utilize the memory

[9]Although Quantum Espresso processing Test Case A should scale up to 1,024 processes [PRACE 2013], we observed very good scalability up to 256 processes but poor scalability (slowdowns) for 512 and 1,024 processes. We, therefore, report results for 256 processes for this application.

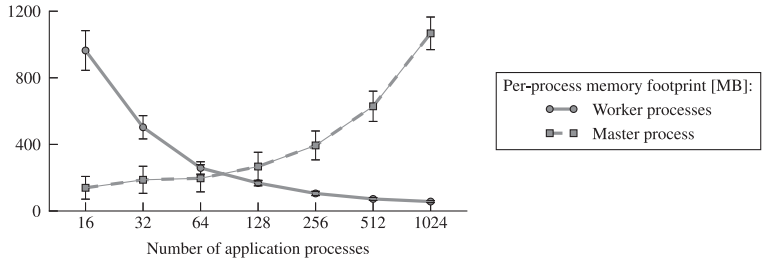


Fig. 9. As the number of processes increase, memory footprints of worker processes decrease as expected, but memory footprint of the master process increases: ALYA, Test Case A.

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.

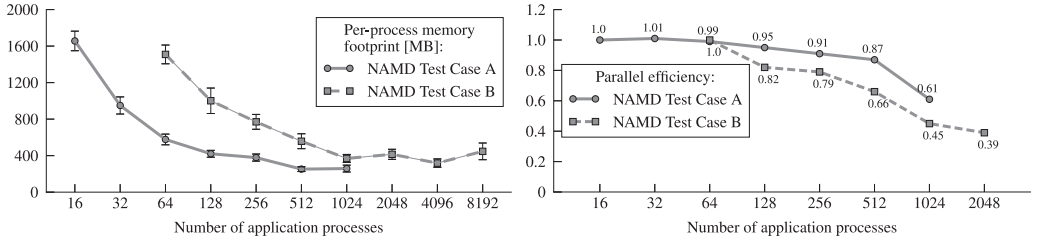
5.3.1. Master Process Memory Requirements. Many HPC applications are written using a master–worker process model. In these applications, the problem is decomposed into data segments that can be processed independently by different *worker processes*. The *master process* (usually the first process) assigns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

Figure 9 plots the memory footprints of the ALYA master and worker processes as the number of processes increases from 16 to 1,024. Memory footprints of worker processes drop as we increase the number of processes, following the trend described in Section 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 the master process may have significantly higher memory footprints than workers, up to 36.6× for NEMO (master 7.2GB, worker 0.2GB) and 62.5× for BQCD (master 7.1GB, worker 0.1GB). Both application developers and computer architects should pay attention to this phenomenon, still not well-highlighted or quantified by the community.

5.4. Toward 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 the case of ALYA, Test Case A is a model of the respiratory system, whereas Test Case B is a mesh of generic elements [Bull 2013]. 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 [Bull 2013], 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 10(a), we plot the NAMD memory footprint results for both the smaller and



(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.

Fig. 10. Increasing the input dataset changes memory footprint and scalability of the NAMD application. We detect the same trend for all UEABS applications.

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 1,024 processes, and for Test Case B, on up to 8,192 processes, as recommended by UEABS documentation [PRACE 2013]. For both input datasets, the NAMD footprint is around 1,600MB 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 toward 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 10(b) 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 1,024 (64×). With Test Case B, when increasing from 64 to 2,048 processes (32×), the parallel efficiency drops to 0.39.[10] 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 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 10(a)). Application scalability, however, reduces, in some cases significantly. Therefore, increasing the input dataset requires repeating the analysis of the tradeoffs between execution time and CPU-hours (Section 5.2) in order to determine the representative number of processes for a production run of the application.

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. In this section, we summarize the pros and the cons of 3D-stacked

[10]NAMD running Test Case B should scale up to 10,000 processes, but we detect very low or no speed-up after 2,048 processes. Therefore, we plot parallel efficiency results up to 2,048 processes for Test Case B.

DRAM, and briefly overview 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. 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.

6.1. Background: 3D-stacked DRAM

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. 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.

HMC [HMC Consortium 2014] is connected to the CPU with a high-speed serial interface that provides up to 480GB/s per device. Announced production runs of HMC components are limited to 2GB and 4GB devices, while the standards specify capacities of up to 8GB. 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. This enables chaining of devices to increase memory capacity, but multi-hop routing may increase access latency and variability. The impact of such variability on HPC workloads requires further analysis and justification.

HBM [JEDEC 2013] is connected to the host CPU or GPU with a wide 1024-bit parallel interface that delivers up to 256GB/s. Similarly to HMC, the standard specifies up to 8GB 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.

Intel Knights Landing (KNL) processors [Sodani et al. 2016] 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 16GB (0.2GB per core) with 400GB/s of peak theoretical bandwidth. In addition to MCDRAM, KNL can be connected to up to 384GB (5.4GB 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.

6.2. 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.

6.2.1. 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 1GB 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, especially on small clusters (see Figure 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 3.2, a million-core system would require 16.1GB per core to achieve 95% of the potential performance (Figure 3). Although the HPC community is questioning whether HPL is representative of modern production HPC applications [Murphy et al. 2006, 2010], 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.

6.2.2. Production HPC Applications. In Section 5.3 and Figure 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 MCDRAM 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 [Chou et al. 2014; Meswani et al. 2015]. 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 [Intel 2016a] and data management

libraries and APIs [Intel 2016b] that simplify efficient programming 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.

6.2.3. 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 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 2GB per high-end x86 core, with some large-memory nodes having 4GB to 8GB per core, i.e., more than 100GB 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 [Radulovic et al. 2015]. 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 centers. 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.

7. RELATED WORK

Dongarra et al. [2003] 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 500MHz CPUs and analyze the benchmark performance for various interconnects and input dataset sizes of up to around 250MB 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. [2014] 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 [Biswas et al. 2011; Perks et al. 2011; Pavlovic et al. 2011]. 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. [2011] and Perks et al. [2011] investigate different techniques to reduce memory footprints in order to improve the performance of HPC workloads. Biswas et al. [2011] 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. [2011] 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. [2011] 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 [Dong et al. 2010; Chou et al. 2014; Sim et al. 2014; Meswani et al. 2015]. 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 [Chou et al. 2014; Sim et al. 2014] or by an interaction between hardware and the operating system [Dong et al. 2010; Meswani et al. 2015]. 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.

8. FUTURE WORK

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. We hope that our study will trigger further research and discussion on this topic. The first topic for future work could be an analysis of 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 sizes.

Also, the community, or at least large research centers that purchase HPC clusters should question the current trends for memory system sizing. Maybe the first steps could be to study the histograms of memory system usage of existing HPC clusters and understand whether users are taking advantage of most of the installed memory. It seems that decisions for new machines have traditionally been based on experiences with previous HPC clusters and on undocumented knowledge of the principal system integrators. This is a conservative approach that will unlikely lead to a breakthrough in the memory system design.

Finally, the principal HPC system designers and integrators should publicly document and justify their memory sizing decisions. For example, based on our results, one could question whether the memory capacity of the largest HPC clusters are provisioned with too much emphasis on the TOP500 list. Since smaller HPC systems are influenced by the system architectures of the larger ones, this could mean that the HPL scores are responsible for most of the HPC memory system sizing decisions. It is about time that HPC system designers try to convince us that this is not true, and explain the real rationale behind the 2–3GB per-core rule.

9. CONCLUSIONS

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 toward 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.

APPENDIX: HPL ANALYTICAL ANALYSIS

The Appendix complements the HPL analysis in Section 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 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 [Petitet et al. 2012]. For a given problem size N, the benchmark performs the following number of double-precision floating-point operations (*#FLOPs*) [Dongarra et al. 2003]:

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

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

$$\#FLOPs_{\text{per\_process}} \simeq \frac{2N^3}{3n}. \tag{4}$$

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 $N_B \times N_B$, 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., $P = Q = \sqrt{n}$ [Petitet et al. 2012].

An approximation of the HPL execution time T that illustrates the cost of the dominant factors is [Petitet et al. 2012]:

$$T = \frac{2\gamma_3 N^3}{3PQ} + \frac{\beta N^2(3P + Q)}{2PQ} + \frac{\alpha N((N_B + 1)\log P + P)}{N_B}. \tag{5}$$

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 Equation (5) by Equation (4), we have:

$$T_{\text{per\_FLOP}} = \gamma_3 + \frac{3\beta\sqrt{n}}{N} + \frac{3\alpha n\left(\frac{1}{2}(N_B + 1)\log n + \sqrt{n}\right)}{2N_B N^2}. \tag{6}$$

Equation (6) 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: $m = \frac{10N^2}{n}$.[11] Therefore, the problem size that generates per-process memory capacity m when the HPL is executed on n processes can be computed as: $N = \sqrt{\frac{mn}{10}}$.

[11]Problem size should be set to 80% of available memory [Petitet et al. 2012]. The $N \times N$ coefficient matrix requires $8N^2$ bytes, so the per-core memory capacity is $\frac{100}{80} \times \frac{8N^2}{n} = \frac{10N^2}{n}$.

If we put this into Equation (6), we determine the dependency between the time per FLOP ($T_{\text{per\_FLOP}}$) and the per-core memory capacity (m):

$$T_{\text{per\_FLOP}} = \gamma_3 + \frac{3\beta\sqrt{10}}{\sqrt{m}} + \frac{15\alpha\left(\frac{1}{2}(N_B+1)\log n + \sqrt{n}\right)}{N_B m}. \tag{7}$$

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 (7) 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, the 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 Equation (7) by γ_3 to normalize relative to the lowest execution time with infinite memory gives the relative execution time per FLOP:

$$T_{\text{rel\_per\_FLOP}} = 1 + k_1\frac{1}{\sqrt{m}} + \left(k_2\sqrt{n} + k_3\log n\right)\frac{1}{m}. \tag{8}$$

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 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 results with infinite memory, i.e., to reach a relative execution time per FLOP of ε. This we get by solving a quadratic equation:

$$\left(k_2\sqrt{n} + k_3\log n\right)\left(\frac{1}{\sqrt{m}}\right)^2 + k_1\left(\frac{1}{\sqrt{m}}\right) + (1-\varepsilon) = 0. \tag{9}$$

The results for ε values that contribute to 90%, 95%, and 99% of ideal (infinite memory) HPL performance are explained in detail in Section 3.2. By taking the derivative of the solution to Equation (9), we find that, for 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 3.1 and historical systems in Figure 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 increases the memory per core for any fixed overhead $\epsilon > 0$.

ACKNOWLEDGMENT

The authors thank Harald Servat from BSC and Vladimir Marjanović from High Performance Computing Center Stuttgart for their technical support.

REFERENCES

Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A. Yelick. 2006. *The Landscape of Parallel Computing Research: A View from Berkeley*. Technical Report UCB/EECS-2006-183. EECS Department, University of California, Berkeley.

Daniel E. Atkins, Kelvin K. Droegemeier, Stuart I. Feldman, Stuart I. Feldman, Michael L. Klein, David G. Messerschmitt, Paul Messina, Jeremiah P. Ostriker, and Margaret H. Wright. 2003. *Revolutionizing Science and Engineering Through Cyberinfrastructure*. Report of the National Science Foundation Blue-Ribbon Advisory Panel on Cyberinfrastructure. National Science Foundation.

Barcelona Supercomputing Center. 2013. *MareNostrum III System Architecture*. Technical Report.

Barcelona Supercomputing Center. 2014. *Extrae User Guide Manual for Version 2.5.1*. Barcelona Supercomputing Center.

Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 72–81.

Susmit Biswas, Bronis R. de Supinski, Martin Schulz, Diana Franklin, Timothy Sherwood, and Frederic T. Chong. 2011. Exploiting data similarity to reduce memory footprints. In *Proceedings of the IEEE International Parallel & Distributed Processing Symposium (IPDPS)*. 152–163.

Mark Bull. 2013. PRACE-2IP: D7.4 unified european applications benchmark suite final. (2013).

Chris Cantalupo, Karthik Raman, and Ruchira Sasanka. 2015. MCDRAM on 2nd Generation Intel Xeon Phi Processor (code-named Knights Landing): Analysis Methods and Tools. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*.

Chiachen Chou, Aamer Jaleel, and Moinuddin K. Qureshi. 2014. CAMEO: A two-level memory organization with capacity of main memory and flexibility of hardware-managed cache. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. 1–12.

Xiangyu Dong, Yuan Xie, Naveen Muralimanohar, and Norman P. Jouppi. 2010. Simple but effective heterogeneous main memory with on-chip memory controller support. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. 1–11.

Jack Dongarra, Michael Heroux, and Piotr Luszczek. 2016. The HPCG Benchmark. Retrieved from http://www.hpcg-benchmark.org/.

Jack J. Dongarra and Michael A. Heroux. 2013. *Toward a New Metric for Ranking High Performance Computing Systems*. Sandia Report SAND2013-4744. Sandia National Laboratories.

Jack J. Dongarra, Piotr Luszczek, and Michael A. Heroux. 2014. HPCG: One year later. In *Proceedings of the International Supercomputing Conference (ISC)*.

Jack J. Dongarra, Piotr Luszczek, and Antoine Petitet. 2003. The LINPACK benchmark: Past, present and future. *Concurrency and Computation: Practice and Experience* 15, 9 (2003), 803–820.

ETP4HPC. 2013. ETP4HPC Strategic Research Agenda Achieving HPC Leadership in Europe. (June 2013).

Hybrid Memory Cube Consortium. 2014. Hybrid Memory Cube Specification 2.0. Retrieved from http://www.hybridmemorycube.org/specification-v2-download-form/.

Intel. 2016a. Intel VTune Amplifier 2016. Retrieved from https://software.intel.com/en-us/intel-vtune-amplifier-xe/.

Intel. 2016b. The memkind library. Retrieved from http://memkind.github.io/memkind/.

JEDEC Solid State Technology Association. 2013. High Bandwidth Memory (HBM) DRAM. http://www.jedec.org/standards-documents/docs/jesd235. (Oct. 2013).

James Jeffers, James Reinders, and Avinash Sodani. 2016. *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition* (2nd ed.). Morgan Kaufmann.

Peter Kogge, Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Kerry Hill, Jon Hiller, Sherman Karp, Stephen Keckler, Dean Klein, Robert Lucas, Mark Richards, Al Scarpelli, Steven Scott, Allan Snavely, Thomas Sterling, R. Stanley Williams, and Katherine Yelick. 2008. ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems. Technical Report.

Matthew J. Koop, Terry Jones, and Dhabaleswar K. Panda. 2007. Reducing connection memory requirements of MPI for infiniband clusters: A message coalescing approach. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*. 495–504.

Piotr Luszczek and Jack J. Dongarra. 2005. *Introduction to the HPC Challenge Benchmark Suite*. ICL Technical Report ICL-UT-05-01. University of Tennessee.

Vladimir Marjanović, José Garcia, and Colin W. Glass. 2014. Performance modeling of the HPCG benchmark. In *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*. Springer International Publishing, 172–192.

Mitesh R. Meswani, Sergey Blagodurov, David Roberts, John Slice, Mike Ignatowski, and Gabriel H. Loh. 2015. Heterogeneous memory architectures: A HW/SW approach for mixing die-stacked and off-package memories. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 126–136.

Richard Murphy, Jonathan Berry, William McLendon, Bruce Hendrickson, Douglas Gregor, and Andrew Lumsdaine. 2006. DFS: A simple to write yet difficult to execute benchmark. In *IEEE International Symposium on Workload Characterization (IISWC)*. 175–177.

Richard Murphy, Kyle Wheeler, Brian Barrett, and James Ang. 2010. Introducing the Graph 500. Cray User's Group (CUG). (May 2010).

NERSC. 2012. *Large Scale Computing and Storage Requirements for High Energy Physics: Target 2017*. Report of the NERSC Requirements Review. Lawrence Berkeley National Laboratory.

NERSC. 2013. *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.

NERSC. 2014a. *High Performance Computing and Storage Requirements for Basic Energy Sciences: Target 2017*. Report of the HPC Requirements Review LBNL-6978E. Lawrence Berkeley National Laboratory.

NERSC. 2014b. *Large Scale Computing and Storage Requirements for Fusion Energy Sciences: Target 2017*. Report of the NERSC Requirements Review LBNL-6631E. Lawrence Berkeley National Laboratory.

NERSC. 2015a. *High Performance Computing and Storage Requirements for Nuclear Physics: Target 2017*. Report of the NERSC Requirements Review LBNL-6926E. Lawrence Berkeley National Laboratory.

NERSC. 2015b. *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.

Chris J. Newburn. 2015. Code for the future: Knights Landing and beyond. In *Proceedings of the International Supercomputing Conference (ISC)*.

Jongsoo Park, Mikhail Smelyanskiy, Karthikeyan Vaidyanathan, Alexander Heinecke, Dhiraj D. Kalamkar, Xing Liu, Md. Mosotofa Ali Patwary, Yutong Lu, and Pradeep Dubey. 2014. Efficient shared-memory implementation of high-performance conjugate gradient benchmark and its application to unstructured matrices. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. 945–955.

Milan Pavlovic, Yoav Etsion, and Alex Ramirez. 2011. On the memory system requirements of future scientific applications: Four case-studies. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*. 159–170.

Milan Pavlovic, Milan Radulovic, Alex Ramirez, and Petar Radojkovic. 2015. Limpio - Lightweight MPI instrumentation. In *Proceedings of the International Conference on Program Comprehension (ICPC)*. Retrieved from https://www.bsc.es/computer-sciences/computer-architecture/memory-systems/limpio, 303–306.

O. Perks, S. D. Hammond, S. J. Pennycook, and S. A. Jarvis. 2011. Should we worry about memory loss? *SIGMETRICS Performance Evaluation Review* 38, 4 (March 2011), 69–74.

Antoine Petitet, Clint Whaley, Jack Dongarra, Andy Cleary, and Piotr Luszczek. 2012. HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers. Retrieved from http://www.netlib.org/benchmark/hpl/.

PRACE. 2013. Unified European Applications Benchmark Suite. www.prace-ri.eu/ueabs/. (2013).

PRACE. 2016. Prace Research Infrastructure. http://www.prace-ri.eu. (2016).

Milan Radulovic, Darko Zivanovic, Daniel Ruiz, Bronis R. de Supinski, Sally A. McKee, Petar Radojković, and Eduard Ayguadé. 2015. Another trip to the wall: How much will stacked DRAM benefit HPC? In *Proceedings of the International Symposium on Memory Systems (MEMSYS)*. 31–36.

Jaewoong Sim, Alaa R. Alameldeen, Zeshan Chishti, Chris Wilkerson, and Hyesoon Kim. 2014. Transparent hardware management of stacked DRAM as part of memory. In *Proc. of the International Symposium on Microarchitecture (MICRO)*. 13–24.

Avinash Sodani. 2011. Race to Exascale: Opportunities and Challenges. International Symposium on Microarchitecture (MICRO). (Dec. 2011). Keynote.

Avinash Sodani, Roger Gramunt, Jesus Corbal, Ho-Seop Kim, Krishna Vinod, Sundaram Chinthamani, Steven Hutsell, Rajat Agarwal, and Yen-Chen Liu. 2016. Knights landing: Second-generation Intel Xeon Phi product. *IEEE Micro* 36, 2 (March 2016), 34–46.

SPEC. 2015a. SPEC MPI2007. Retrieved from http://www.spec.org/mpi2007/.

SPEC. 2015b. SPEC OMP2012. https://www.spec.org/omp2012/.

Rick Stevens, Andy White, Pete Beckman, Ray Bair-ANL, Jim Hack, Jeff Nichols, Al GeistORNL, Horst Simon, Kathy Yelick, John Shalf-LBNL, Steve Ashby, Moe Khaleel-PNNL, Michel McCoy, Mark Seager, Brent Gorda-LLNL, John Morrison, Cheryl Wampler-LANL, James Peery, Sudip Dosanjh, Jim Ang-SNL, Jim Davenport, Tom Schlagel, BNL, Fred Johnson, and Paul Messina. 2010. A Decadal DOE Plan for Providing Exascale Applications and Technologies for DOE Mission Needs. Presentation at Advanced Simulation and Computing Principal Investigators Meeting.

Erich Strohmaier, Jack Dongarra, Horst Simon, Martin Meuer, and Hans Meuer. 2015. TOP500 List. Retrieved from http://www.top500.org/.

Frederick C. Wong, Richard P. Martin, Remzi H. Arpaci-Dusseau, and David E. Culler. 1999. Architectural requirements and scalability of the NAS parallel benchmarks. In *Proceedings of the of the ACM/IEEE Conference on Supercomputing (SC)*.

Steven Cameron Woo, Moriyoshi Ohara, and Evan Torrie. 1995. The SPLASH-2 programs: Characterization and methodological considerations. In *Proceedings of the of the International Symposium on Computer Architecture (ISCA)*. 24–36.

Darko Zivanovic, Milan Radulovic, Germán Llort, David Zaragoza, Janko Strassburg, Paul M. Carpenter, Petar Radojković, and Eduard Ayguadé. 2016. Large-memory nodes for energy efficient high-performance computing. In *Proceedings of the of the International Symposium on Memory Systems (MEMSYS)*.

Received May 2016; revised November 2016; accepted December 2016