HPC Benchmarking: Scaling Right and Looking Beyond the Average

Milan Radulovic[1,2(✉)], Kazi Asifuzzaman[1,2], Paul Carpenter[1], Petar Radojković[1], and Eduard Ayguadé[1,2]

[1] Barcelona Supercomputing Center (BSC), Barcelona, Spain
{milan.radulovic,kazi.asifuzzaman,paul.carpenter,
petar.radojkovic}@bsc.es
[2] Universitat Politècnica de Catalunya (UPC), Barcelona, Spain
eduard@ac.upc.edu

Abstract. Designing a balanced HPC system requires an understanding of the dominant performance bottlenecks. There is as yet no well established methodology for a unified evaluation of HPC systems and workloads that quantifies the main performance bottlenecks. In this paper, we execute seven production HPC applications on a production HPC platform, and analyse the key performance bottlenecks: FLOPS performance and memory bandwidth congestion, and the implications on scaling out. We show that the results depend significantly on the number of execution processes and granularity of measurements. We therefore advocate for guidance in the application suites, on selecting the representative scale of the experiments. Also, we propose that the FLOPS performance and memory bandwidth should be represented in terms of the proportions of time with low, moderate and severe utilization. We show that this gives much more precise and actionable evidence than the average. AQ1

Keywords: HPC applications · Bottlenecks · FLOPS
Memory bandwidth · Scaling-out

1 Introduction

Deploying an HPC infrastructure is a substantial investment in time and money, so it is extremely important to make the right procurement decision. Unfortunately, evaluating HPC systems and workloads and quantifying their bottlenecks is hard. There are currently three main approaches. The approach taken by TOP500 and Green500 is to evaluate systems using a prominent HPC benchmark, such as High-Performance Linpack (HPL) [20] or High Performance Conjugate Gradients (HPCG) [5]. Another approach is to measure the sustained performance of the various components in the system using specialized kernel benchmarks, such as HPC Challenge [13]. By design, kernel benchmarks quantify only the sustainable performance of individual system components, so they lack the capability to determine how a real-world production HPC application will behave on the same platform.

M. Aldinucci et al. (Eds.): Euro-Par 2018, LNCS 11014, pp. 1–12, 2018.
https://doi.org/10.1007/978-3-319-96983-1_10

The final approach, which is the one taken in this paper, is to mimic production use by running a set of real HPC applications from diverse scientific fields [23]. We execute seven production HPC applications, together with HPL and HPCG, on a production x86 platform, and we reach two main conclusions. Firstly, we find that HPC application performance and CPU/memory system bottlenecks are strongly dependent on the **number of application processes**. This is typically overlooked in benchmark suites, which seldom define how many processes should be used. We argue that it is essential that HPC application suites specify narrow ranges on the number of processes, so that the results are representative of real world application use, or that they at least provide some guidelines. Secondly, we find that **average values of bytes/FLOP, bytes/s and FLOPs/s can be highly misleading**. Our results show that the applications under study have low average FLOPs/s utilization and moderate pressure on the memory bandwidth. However, we identified several applications, such as ALYA and GENE, with a moderate *average* memory bandwidth that spend more than 50% of their computation time in phases where the memory bandwidth bottleneck is severe. We therefore recommend that rather than thinking in terms of average figures, one measures the percentage of time that the utilization of memory bandwidth or FLOPs/s is low (below 40% of sustainable maximum), moderate (40% to 80%) and severe (above 80%). These three figures give a much more precise picture of the application behavior than the average.

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

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

2 Experimental Environment

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

2.1 Experimental Platform

The experiments are executed on the MareNostrum 3 supercomputer [3], the third version of one of the six Tier-0 (largest) HPC systems in Europe [21]. It comprises dual-socket Intel Sandy Bridge-EP E5-2670 nodes. Each socket comprises eight cores operating at 3.0 GHz. As in most HPC systems, hyperthreading is disabled. The processors connect to main memory through four channels, each with a single DDR3-1600 DIMM. Regular MareNostrum compute nodes include 32 GB of DRAM memory, i.e., 2 GB per core. The nodes are connected with an InfiniBand FDR-10 (40 Gb/s) interconnect, as a non-blocking two-level fat-tree topology.

2.2 Workloads

High-Performance Linpack
For a long time, the High-Performance Linpack (HPL) [20] benchmark has been the de facto metric for ranking HPC systems. It measures the sustained floating-point rate (GFLOPs/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 the user can scale the problem to achieve the best performance on a given system. HPL stresses only the system's floating point performance, without stressing other important contributors to overall performance, such as the memory subsystem. The most prominent evaluation of HPC systems constitutes the TOP500 list [24], which has been criticized for assessing system performance using only HPL [12]. The community has pointed out the weaknesses of HPL and advocated for a way to evaluate HPC systems that is better correlated with the needs of production HPC applications [6].

High-Performance Conjugate Gradients
High Performance Conjugate Gradients (HPCG) [5], has been released as a complement to the FLOPs-bound HPL. It is based on an iterative sparse-matrix conjugate gradient kernel with double-precision floating-point values. While HPL can exploit data locality and thus cope with relatively low memory bandwidth, HPCG performance is largely proportional to the available memory bandwidth. HPCG is a good representative of HPC applications governed by differential equations, which tend to have much stronger needs for high memory bandwidth and low latency, and tend to access data using irregular patterns.

HPC Applications
Evaluating HPC systems using benchmarks that target specific performance metrics is not enough to determine the performance of a real-world application. It is therefore essential to execute production applications on an HPC system to better understand the bottlenecks and constraints experienced by a production HPC application. There are efforts in making suites of HPC applications that could be used in benchmarking purposes, such as NSF [17], NCAR [15] and NERSC Trinity benchmarks [16] in USA, and EuroBen [8], ARCHER [25] and Unified European Application Benchmark Suite (UEABS) [18] in Europe.

Table 1. Scientific HPC applications used in the study

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

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

In our evaluation, we used a set of UEABS applications. UEABS represents a set of production applications and datasets, from various scientific domains, designed for benchmarking the European HPC systems, included in the Partnership for Advanced Computing in Europe (PRACE) [21], for procurement and comparison purposes. Parallelized using the Message Passing Interface (MPI), these applications are regularly executed on hundreds to thousands of cores. We study 7 of 12 applications from UEABS [18], listed in Table 1.[1]

Tools and Methodology

The applications come with input datasets and a recommended range of CPU cores for the experiments. We use the *Test Case A* datasets, which are deemed suitable for Tier-1 systems up to about 1,000 cores [18]. In all experiments, we execute one application process per CPU core. The number of processes starts from 16 (a single MareNostrum node) and it increases by powers of two until 1,024 processes. Some of the applications have memory capacity requirements that exceed the available node memory, which limits the lowest number of processes in the experiments, e.g., BQCD cannot be executed with fewer than 64 processes (four nodes). The presented analysis keeps constant the input dataset and varies the number of application processes, which refers to a strong scaling case.[2]

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

[2] The alternative would be a weak scaling analysis, in which the problem size scales with the number of nodes. Unlike HPL and HPCG, for which the problem size is defined by the user and the input data is generated algorithmically, application benchmark suites include specific input problem data. We are not aware of a production application benchmark suite that has problems suitable for weak scaling analysis. Although some of the UEABS benchmarks are distributed with two input datasets, small and large, they are not comparable so are insufficient for weak scaling analysis [26].

The application's computation bursts were instrumented with Limpio [19] and Extrae [4]. We used core performance counters [10] to measure FLOPS performance (scalar and vector FLOPS counters) and uncore performance counters [9] to measure memory bandwidth (read and write CAS commands counters).

We analyze the application behavior at two levels of granularity. First, we plot mean FLOPs and memory bandwidth utilization using end-to-end measurements and averaging the values of all application processes. Second, we analyze the fine-granularity measurements done at the computational burst level. For each computational burst, we measure the FLOPs, bandwidth utilization and the burst execution time. Afterwards, we analyze the cumulative distribution function of the measurements.[3] As we show in this paper, these two levels of the analysis can, and often do, actually lead to different conclusions.

3 Results

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

3.1 Floating-Point Performance Analysis

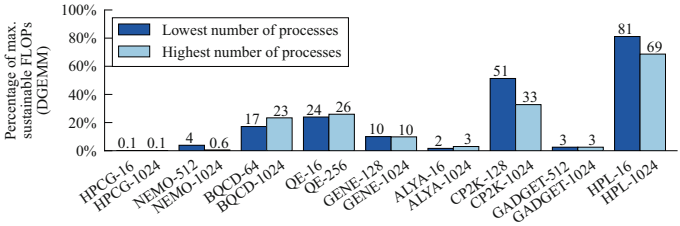
Figure 1a plots the average FLOPs/s utilization for different numbers of application processes. The results show that the average FLOPs/s utilization of production HPC applications is fairly low: for most applications it is below 30%, and in the best case it reaches only 51% (CP2K-128 experiment). Figure 1b summarizes the distribution of measurements done at computational burst level. We divide the computational burst measurements into five clusters: 0–20%, 20–40%, 40–60%, 60–80% and 80–100% of sustained FLOPs/s, and then plot the portion of execution time represented by each cluster. For example, in the BQCD-64 experiment, 72% of the time the FLOPs/s utilization is between 0 and 20%, while for the remaining 28% of the time it is between 20% and 40%.

Our results show that detailed measurements are indeed needed, and that plotting only average values may hide important information. The most obvious case would be the QE-16 experiment. Although the average FLOP utilization is only 24% (Fig. 1a), the application actually puts extremely high pressure on CPU FLOPs for around 18% of its computation time (Fig. 1b).

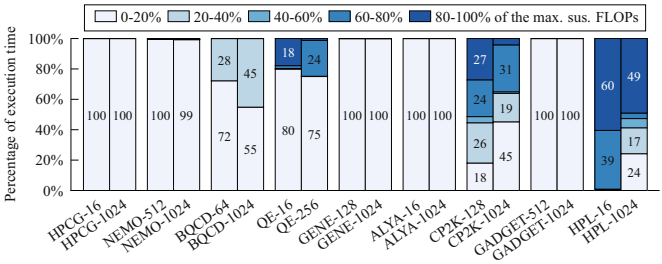
We also analyze changes in the application behavior when executing them using different numbers of processes. Both, average and per-burst measurements indicate significant changes in the application behavior as the applications scale-out[4].

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

[4] We remind the reader that we used the official input datasets, and followed the recommendations about the range of CPU processes that should be used in the experiments (see Sect. 2.2).



(a) Average FLOPS utilization



(b) FLOPS utilization on burst granularity

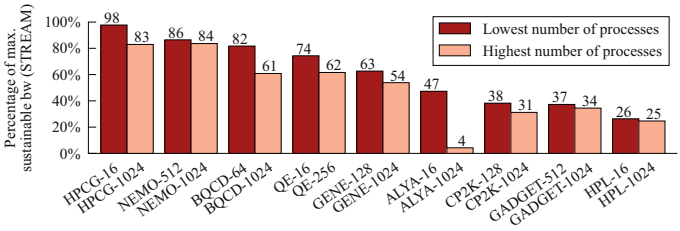
Fig. 1. Production HPC applications show fairly low FLOPS utilization, both on lowest and highest number of processes.

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

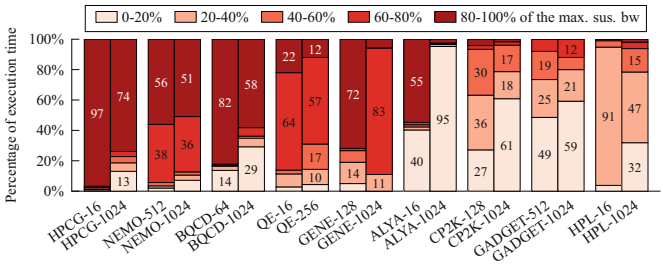
3.2 Memory Bandwidth Analysis

Memory bandwidth has become increasingly important in recent years. Keeping the memory bandwidth balanced with the CPU's compute capabilities, within affordable costs and power constraints, has become a key technological challenge. The increasing awareness of this challenge also resulted in the introduction of the HPCG benchmark, as an alternative to HPL. The industry also responded to the growing need for more memory bandwidth, and high-bandwidth 3D-stacked DRAM products are hitting the market. Their manufacturers promise significant performance boosts over standard DDRx DIMMs, although some independent studies doubt whether and to what extent high-bandwidth memory will benefit HPC applications [22].

Memory bandwidth collision can indeed have the strong negative performance impact. When a workload uses more than 40% of maximum sustainable bandwidth, concurrent memory accesses start to collide, which increases memory latency causing performance penalties. Using more than 80% of maximum sustainable bandwidth causes severe collisions among concurrent memory requests; thus memory latency increases exponentially and memory bandwidth becomes a serious performance bottleneck [11].



(a) Average memory bandwidth utilization



(b) Memory bandwidth utilization on burst granularity

Fig. 2. Contrary to FLOPS, memory bandwidth utilization of production HPC applications is substantial.

Figure 2 plots the memory bandwidth usage of UEABS applications. The memory bandwidth values are plotted relative to the maximum sustained memory bandwidth measured by the STREAM benchmark. Again, we plot the results at two levels of granularity: Fig. 2a plots average utilization over computation time and for different numbers of application processes, while Fig. 2b shows fine-granularity measurements at the computational burst level. The applications under study show higher utilization of memory bandwidth, than FLOPs performance, even for the average values.

Next we analyze the computational bursts measurements, presented in Fig. 2b. The chart shows moderate to high memory bandwidth utilization. All the applications under study have segments in which memory bandwidth utilization exceeds 40%, and all but two of them, CP2K and GADGET, spend a significant portion of time with bandwidth utilization above 60% or even 80%.

The computational burst measurements reveal some interesting scenarios, which are more apparent in Fig. 3. In this figure, the x-axis is the *average* memory bandwidth utilization, as in Fig. 2a. The y-axis is the proportion of time for which the memory bandwidth utilization is severe; i.e. more than 80% of the sustainable maximum, which corresponds to the darkest shade parts of the bars in Fig. 2b. Figure 3 shows that considering the average memory bandwidth on the x-axis, ALYA-16 and CP2K-128 may seem to be bandwidth insensitive, as their average bandwidths are around 50% and 40% of the sustained bandwidth. However, detailed in-time measurements show that they spent significantly different proportions of the time with severe memory bandwidth utilization: CP2K-128

spends only about 4% of its computation time, but ALYA-16 spends 55% of its computation time, which presents a serious performance penalty. We find a similar situation with BQCD-1024, GENE-128 and QE-1024. These applications all have average memory bandwidth of around 60% of the sustained maximum. Even so, QE-256 spends only 12% of its computation time with severe memory bandwidth utilization (more than 80% of maximum sustained). In contrast, BQCD-1024 and GENE-128 spend 58% and 72% of their computation time, respectively, with severe memory bandwidth utilization.

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

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

3.3 Discussion

Our analysis emphasizes that HPC application behavior is tightly coupled with the number of application processes. There are two main reasons for this. First, application scaling-out increases the inter-process communication time. To illustrate this, in Fig. 4 we plot the portion of overall execution time that applications under study spend in inter-process communication.

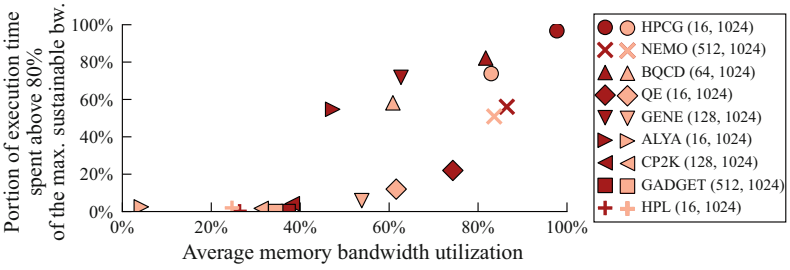


Fig. 3. Average memory bandwidth can mislead and hide potential bottlenecks. BQCD-1024, GENE-128 and QE-256 have similar average memory bandwidths, however BQCD-1024 and GENE-128 spend significantly more time utilizing more than 80% of max. sustainable bandwidth, which is a serious bottleneck.

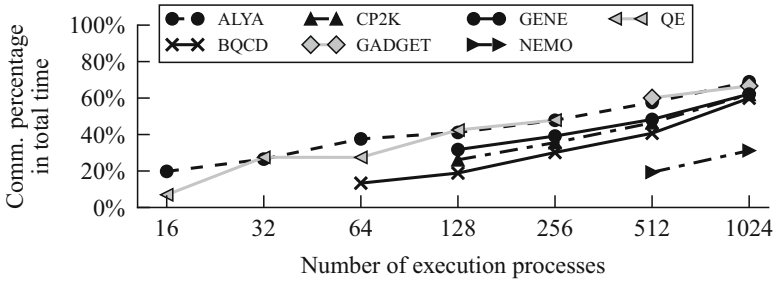


Fig. 4. Portion of total execution time spent in the inter-process communication for UEABS applications, strong scaling.

Even for the low number of application processes, the communication is non-negligible, and as the number of processes increases, it becomes the dominant part of the overall execution time. The higher the portion of time that is spent in communication, the lower the average utilization of FLOPs and memory bandwidth (as detected in Figs. 1a and 2a). Also, in general, the higher the number of processes, the smaller the portion of the input data handled by each process, which changes the effectiveness of cache memory and the overall process behavior (as detected in Figs. 1b and 2b).

HPC application behavior may be known by the application developers, but it is often overlooked in all HPC application suites for benchmarking purposes. State-of-the-art HPC application suites do not strictly define the number of processes to use in experiments. For example, UEABS recommends running the applications with corresponding input datasets on up to approximately 1,000 processes, but the minimum number of processes is not specified. Similarly, other HPC application suites either provide loose recommendations about the number of processes [15–17,25] or do not discuss this issue at all [8]. However, it is not surprising that HPC application suites overlooked the problem that application behavior is tightly-coupled with number of application processes. After all, this problem does not exist for single-threaded benchmarks, and it was not detected for HPC benchmarks that put high stress to a single resource, such as HPCG, HPL or HPCC suite.

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

4 Related Work

There are not many studies that analyse benchmarking methodologies and how to represent evaluation results of HPC systems and applications. Bailey [1] provides common guidelines for reporting benchmark results in technical computing, following his similar summary of misleading claims for reporting results in system evaluation [2]. He points out the possibilities of misleading conclusions and potential biases from using projections and extrapolations, tuning levels, evaluating non-representative segments of the workloads, etc. Nevertheless, he presents several rules and advocates the community to pay attention and avoid the biased results.

Hoefler and Belli [7] attempt to define ground rules and guidelines for the interpretation of HPC benchmarking. The authors propose statistical analysis and reporting techniques in order to improve the quality of reporting research results in HPC and ensure interpretability and reproducibility. In their study, they identify several frequent problems and propose rules to avoid them. Their analysis covers methods for reporting the results of speed-up, usage of various means, summarizing ratios, confidence intervals, normalization, usage of various chart techniques, and others.

Sayeed et al. [23] advocate the use of real applications for benchmarking in HPC, and that small benchmarks cannot predict the behavior of the real HPC applications. They discuss important questions, challenges, tools and metrics in evaluating performance using HPC applications. Afterwards, they evaluate the performance of four application benchmarks on three different parallel architectures, and measure the runtime, inter-process communication overhead, I/O characteristics and memory footprint. This way, they show the importance of reporting various metrics, in order to have a better representation of application and system performance. Since they measure these metrics on several numbers of execution processes, the results differ from one execution to another. It is clear from their results that on different numbers of execution processes, different platforms perform better or worse, which can significantly bias the analysis on certain scale of the experiments.

Marjanović et al. [14] explore the impact of input data-set for three representative benchmarks: HPL, HPCG and High-performance Geometric Multigrid (HPGMG). They perform an analysis on six distinct HPC platforms at the node level, and perform scale-out analysis on one of the platforms. Their results show that exploring multiple problem sizes gives a more complete picture of the underlying system performance, than a single number representing the best performance, which is the usual way of reporting the results. They advocate for the community to discuss and propose a method for aggregating these values into a representative result of the system performance.

In our study, we focus on two important aspects of benchmarking with HPC applications: the importance of defining the representative scale of the experiments and measurement granularity in quantifying performance bottlenecks, which are often overlooked by the community. To our knowledge, this is the first study that analyses the importance of a deterministic range for the number of

execution processes. We also suggest a simple way to show several values for portions of time spent in different utilizations of certain metric. It does not require additional executions or special evaluation infrastructure, yet it gives much better representation of application behavior and clearer focus on its bottlenecks.

5 Conclusions

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

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

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

Acknowledgements. This work was supported by the Spanish Ministry of Science and Technology (project TIN2015-65316-P), Generalitat de Catalunya (contracts 2014-SGR-1051 and 2014-SGR-1272), Severo Ochoa Programme (SEV-2015-0493) of the Spanish Government; and the European Union's Horizon 2020 research and innovation programme under ExaNoDe project (grant agreement No 671578).

References

1. Bailey, D.H.: Misleading performance claims in parallel computations. In: 2009 46th ACM/IEEE Design Automation Conference, pp. 528–533, July 2009. https://doi.org/10.1145/1629911.1630049
2. Bailey, D.H.: Twelve ways to fool the masses when giving performance results on parallel computers. In: Supercomputing Review, pp. 54–55, August 1991
3. Barcelona Supercomputing Center: MareNostrum III System Architecture (2013). http://www.bsc.es/marenostrum-support-services/mn3
4. Barcelona Supercomputing Center: Extrae User guide manual for version 3.1.0, May 2015
5. Dongarra, J., Heroux, M., Luszczek, P.: The HPCG Benchmark (2016). http://www.hpcg-benchmark.org
6. Heroux, M., Dongarra, J.: Toward a New Metric for Ranking High Performance Computing Systems. Technical report SAND2013-4744, UTK EECS and Sandia National Labs, June 2013
7. Hoefler, T., Belli, R.: Scientific benchmarking of parallel computing systems: twelve ways to tell the masses when reporting performance results. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 73:1–73:12, November 2015

8. Home page of the EuroBen Benchmark. http://www.euroben.nl
9. Intel Corporation: Intel® Xeon® Processor E5–2600 Product Family Uncore Performance Monitoring Guide. Technical report, March 2012
10. Intel Corporation: Intel® 64 and IA-32 Architectures Software Developer's Manual. Technical report, July 2017
11. Jacob, B.L.: The memory system: you can't avoid it, you can't ignore it, you can't fake it. Synth. Lect. Comput. Archit. **4**(1), 1–77 (2009)
12. Kramer, W.T.: Top500 versus sustained performance: the top problems with the Top500 list - and what to do about them. In: Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques, pp. 223–230, September 2012
13. Luszczek, P.R., et al.: The HPC Challenge (HPCC) Benchmark Suite. In: Proceedings of the ACM/IEEE Conference on Supercomputing (2006)
14. Marjanović, V., Gracia, J., Glass, C.W.: HPC benchmarking: problem size matters. In: Proceedings of the 7th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computing Systems, pp. 1–10, November 2016
15. National Center for Atmospheric Research: CISL High Performance Computing Benchmarks. http://www2.cisl.ucar.edu/resources/computational-systems/cisl-high-performance-computing-benchmarks
16. National Energy Research Scientific Computing Center: NERSC-8 / Trinity Benchmarks. http://www.nersc.gov/users/computational-systems/cori/nersc-8-procurement/trinity-nersc-8-rfp/nersc-8-trinity-benchmarks
17. National Science Foundation: Benchmarking Information Referenced in the NSF 11–511 High Performance Computing System Acquisition: Towards a Petascale Computing Environment for Science and Engineering. https://www.nsf.gov/pubs/2006/nsf0605/nsf0605.pdf
18. Partnership for Advanced Computing in Europe (PRACE): Unified european applications benchmark suite (2013). www.prace-ri.eu/ueabs/
19. Pavlovic, M., Radulovic, M., Ramirez, A., Radojković, P.: Limpio: LIghtweight MPI instrumentatiOn. In: Proceedings of the 23rd IEEE International Conference on Program Comprehension, pp. 303–306 (2015)
20. Petitet, A., Whaley, R.C., Dongarra, J., Cleary, A.: HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers, September 2008. http://www.netlib.org/benchmark/hpl/
21. PRACE Research Infrastructure. www.prace-ri.eu
22. Radulovic, M., et al.: Another trip to the wall: how much will stacked DRAM benefit HPC? In: Proceedings of the International Symposium on Memory Systems, pp. 31–36 (2015)
23. Sayeed, M., Bae, H., Zheng, Y., Armstrong, B., Eigenmann, R., Saied, F.: Measuring high-performance computing with real applications. Comput. Sci. Eng. **10**(4), 60–70 (2008). https://doi.org/10.1109/MCSE.2008.98
24. TOP500 List, November 2014. http://www.top500.org/
25. Turner, A.: UK National HPC Benchmarks. Technical report, UK National Supercomputing Service ARCHER (2016). http://www.archer.ac.uk/documentation/white-papers/benchmarks/UK_National_HPC_Benchmarks.pdf
26. Zivanovic, D., et al.: Main memory in HPC: do we need more or could we live with less? ACM Trans. Archit. Code Optim. **14**(1), 3:1–3:26 (2017)