Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 340 (2018) 55–65

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Understanding Communication Patterns in HPCG

Dean G. Chester,[1] Steven A. Wright, Stephen A. Jarvis

Department of Computer Science
University of Warwick
Coventry, United Kingdom

Abstract

Conjugate Gradient (CG) algorithms form a large part of many HPC applications, examples include bioinformatics and weather applications. These algorithms allow numerical solutions to complex linear systems. Understanding how distributed implementations of these algorithms use a network interconnect will allow system designers to gain a deeper insight into their exacting requirements for existing and future applications.

This short paper documents our initial investigation into the communication patterns present in the High Performance Conjugate Gradient (HPCG) benchmark. Through our analysis, we identify patterns and features which may warrant further investigation to improve the performance of CG algorithms and applications which make extensive use of them. In this paper, we capture communication traces from runs of the HPCG benchmark at a variety of different processor counts and then examine this data to identify potential performance bottlenecks. Initial results show that there is a fall in the throughput of the network when more processes are communicating with each other, due to network contention.

Keywords: HPCG, MPI, Communication Pattern, Performance

1 Introduction

Understanding communication in High Performance Computing (HPC) is an ongoing area of research as the network design can have a drastic effect on the performance of a system. Rajovic et al. show that the network interconnet can affect the overall performance of a system [19]. During the design phase of an HPC system, the network interconnect should not be overlooked; current and future requirements need to be understood to have a greater understanding of how future HPC applications will need this integral component. Bergman et al. have proposed that the key metrics for the interconnect to be maximised are: current flow, signalling rate, and peak and sustainable data bandwidth [3].

[1] Email:d.chester.1@warwick.ac.uk

https://doi.org/10.1016/j.entcs.2018.09.005

Linear solver algorithms are used in a wide variety of HPC applications from bioinformatics to environment/weather applications [4,16]. Conjugate Gradient (CG) algorithms, a type of linear solver algorithm that allow numerical solutions to linear systems, form a large part of many HPC applications. Gaining a deeper understanding of how these algorithms use the network interconnect will provide system designers an insight into their exacting requirements for existing and future applications. Notable examples include a CG solver for finding solutions to the Pressure-Poisson equation within an incompressible flow solver [1].

Benchmarking systems allows comparisons to be viewed so that system designers can design and build the most suited system for their applications. Various benchmarks are available to ascertain the performance of a full system and its subsystems, and all of these tools have advantages and disadvantages. Benchmarking subsystems can be beneficial when system designers want to gain a greater understanding of said subsystem; for example MPI benchmarks include SKaMPI and Intel MPI benchmarks. SKaMPI is a benchmark for testing an inconnect using the Message Passing Interface (MPI) developed by the University of Karlsruhe [20]. The Intel MPI benchmarks provide a set of tests for point-to-point and global communication operations for variable message sizes [11].

In this paper we seek to discover if there are any patterns that could be used to further investigate any performance issues relating to how CG solvers use the network interconnect. Specifically, this paper makes the following contributions:

- We execute the High Performance Conjugate Gradient (HPCG) mini-application in a number of configurations and use the Intel Trace Analyzer and Collector to collect data relating specifically to the performance of the MPI communication routines;
- We provide an analysis of the collected data, highlighting how message size varies across different scaled runs; how the messages affect network throughput; and how much time the HPCG mini-application spends performing different operations.

The remainder of this paper is organised as follows: Section 2 provides an overview of related work; Section 3 explains the HPCG benchmark; Section 4 describes the experimental setup; Section 5 outlines the results and discussion; finally, Section 6 summaries the conclusions and discusses future work.

2 Related Work

Benchmarking is critical when deciding what hardware to procure or deciding which algorithm to employ over another. Through benchmarking, the performance of a system or subsystem can be ascertained, and potential bottlenecks can be identified. This in turn helps to define current and future requirements [22].

To measure the *compute* performance of a parallel machine LINPACK is often used. LINPACK reports the number of floating point operations that can be performed each second (FLOP/s) for a *compute element* [9]. This is achieved by solving systems of simultaneous linear algebraic equations. High Performance LINPACK

(HPL) is considered the gold standard for benchmarking and is used for calculating the Top500 supercomputing list [9].

STREAM is a *memory* benchmark, that allows the memory in a system to be benchmarked by moving and storing data in RAM [15]. This is done by performing four long vector operations with data that is too large to fit in the cache and structuring the code so that data re-use is not possible [14].

SKaMPI works by stress testing the collective and point-to-point operations (ping-pong) of the *network interconnect* [20]. This provides a good approximation of the maximum throughput for MPI but fails to take in to account issues like contention in the network when more than two processes are communicating with each other. Intel overcome this issue in "Multimode PingPong" which allows more than two processes to run ping-pong simultaneously providing a better indication of the throughput of the network interconnect [11].

2.1 Mini-applications

To provide a more accurate indication of a systems performance, mini-applications are often used that are broadly representative of of a HPC centres scientific workload. One such example, examined in this paper, is the HPCG mini-application. This benchmark solves linear equations and was designed as a replacement for HPL which only benchmarks the compute elements in the system and does not take into account the irregular and recursive computations [6]. HPL does not take in to account communication patterns when calculating the results which means that system designers would be unaware of their bottleneck in the network interconnect. The performance of a system does not just come from the compute elements, it needs to be tightly coupled with the use of the local memory of the system. The use of a CG algorithm is therefore a closer reflection of the modern applications currently being used [7].

The NASA Advanced Supercomputing (NAS) Parallel Benchmarks (NPB) also contain a conjugate gradient solver, named CG. The CG benchmark evaluates irregular long distance data communications with matrix vector multiplication; this is similar to the HPCG benchmark [2].

Dongarra et al. present a further iterative solver benchmarking tool focusing on a wide variety of iterative methods for solving common problems in the HPC application space [5]. This was chosen as the variety of changeable variables (problem size, computer arithmetic, data structures, system architecture) does not make it easily benchmarkable. This allows users to choose representative solvers for their problems which allows for a better understanding of how the machine will perform their specific requirements.

2.2 Interconnect Performance Analysis

The MPI standard provides a mechanism for incorporating tracing in to an application [10]. This involves intercepting the MPI function calls and providing wrappers that can capture the required data. Vendor specific implementations provide easier

to use tracing capabilities; for example, the Intel Parallel Studio contains Intel Trace Analyser and Collector which provides an interface for capturing the MPI activity allowing communication patterns to be obtained [12]. Other implementations of these instrumentation libraries exist for OpenMPI such as VampirTrace [17].

Analysing the performance of MPI communications is an important issue when looking at underlying performance issues within an application. Pješivac-Grbović et al. presented performance analysis of Point-to-Point and collective MPI operations by verifying existing Point-to-Point and collective MPI models, such as Hockney and PLogP; then they compare the results to real-world data that they have collected from their research machines [18]. The authors demonstrate that for Point-to-Point MPI operations the models are not a reliable method for predicting performance while for collective operations the models were accurate.

3 HPCG

In this paper we focus on HPCG. This has been chosen as it is broadly representative of a number of real-world applications that are using linear solvers. It is also an easily accessible reference implementation of a conjugate gradient algorithm.

HPCG uses a preconditioned CG method with a local symmetric Gauss-Seidel preconditioner.

During a typical execution, HPCG has the following phases: (i) problem setup; (ii) verification and validation testing; (iii) reference sparse matrix-vector (MV) and Gauss-Seidel timing; (iv) reference CG timing and residual reduction; and, (v) optimise CG setup and results reporting [8].

(i) The problem setup constructs the geometry based upon the input parameters, sets up halos for process interaction, and performs optimisations. These include optimisations for better memory access patterns;

(ii) The verification and validation testing verifies the correct optimisations have been applied to the problem. This involves performing spectral and symmetry tests, that ensure that the problem is solvable;

(iii) The reference sparse MV and Gauss-Seidel phase times a reference implementation. These timings are then output in the final report;

(iv) The algorithm is then executed for 50 iterations which are timed and the reduction residual recorded as a baseline during the reference CG timing and residual reduction phase;

(v) Optimisations of the CG setup then take place and an optimised CG solver is then run until it reaches the same residual reduction as the baseline. Finally, the results are output to a file which can be uploaded to the HPCG website.

Algorithm 1 describes a preconditioned CG algorithm. This preconditioned algorithm solves $b - Ap_0 = r_i$. The first residual reduction (r_0) is calculated, and then updated each iteration until it converges. The solution is then x_i. The preconditioner is the matrix M^{-1} which is applied to the residual reduction [21].

Algorithm 1. Preconditioned Conjugate Gradient Algorithm

$r_0 \coloneqq b - Ax_0$
$z_0 \coloneqq M^{-1}r_0$
$p_0 \coloneqq z_o$
for $i = 1, 2, \ldots$ until convergence **do**
  $\alpha_i \coloneqq \frac{r_i \bullet z_i}{Ap_i \bullet p_i}$
  $x_{i+1} \coloneqq x_i + \alpha_i p_i$
  $r_{i+1} \coloneqq r_i - \alpha_i A p_i$
  $z_{i+1} \coloneqq M^{-1} r_{i+1}$
  $\beta_i \coloneqq \frac{r_{i+1} \bullet z_{i+1}}{r_i \bullet z_i}$
  $p_{i+1} \coloneqq z_{i+1} + \beta_i p_i$
end for

Lenovo NeXtScale nx360 M5	
Node Architecture	2x 14-core Intel Xeon E5-2680 v4 (Broadwell) 2.4 GHz
Memory per node	128 GB
Interconnect	Intel Omni-Path X16 100 Gb/s
OS	CentOS 7
Compilers and flags	Intel Compiler icc version 17.0.2 `-O3 -qopenmp`

Table 1
Orac Specifications

The problem setup of the HPCG benchmark is vital to ensure that the problem converges, as seen in Algorithm 1 it can be seen that if a problem does not converge then it could potential run forever as a solution may never be found.

4 Experimental Setup

To understand the communication patterns in the HPCG benchmark application, traces have been captured using the Intel Trace Analyzer and Collector using Intel's own MPI implementation [13]. The results of this are collated and analysed. The traces collected can be used to obtain performance data relating to the point-to-point communications between the processes for each of the runs.

HPCG version 3 was built using the 2017 Intel compiler taking advantage of OpenMP to parallelise some operations inside of the benchmarking tool. The experiments were conducted on Orac, an 84 node research cluster. Each node contains two Intel Xeon E5-2680 v4, 128GB RAM and the nodes are connected by an Intel Omni-Path X16 100 Gb/s interconnect. Other specifications can be seen in Table 1. In this paper each run of HPCG has been given the same problem size ($104 \times 104 \times 104$).

5 Results and Discussions

HPCG contains a mixture of point-to-point and collective operations. The point-to-point operations are due to halo exchanges; the process in which nearby processes

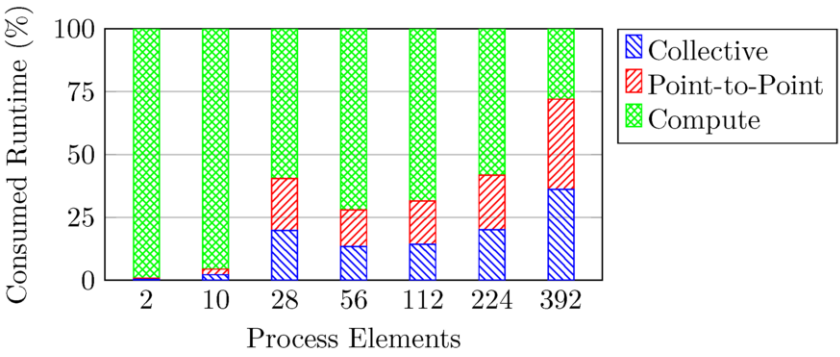

Fig. 1. The total percentage of time for collective, point-to-point and compute operations.

exchange their updated boundary data. The collective communications observed are in reduction operations, such as calculating the residual after each time step.

Figure 1 shows the percentage of time spent performing point-to-point, collective MPI operations and computation. For smaller numbers of processes most of the execution time is spent doing compute elements and not communicating. As the process count increases, more time is spent communicating than computing. Figure 1 also shows that the time for performing collective operations is approximately the same as the point-to-point operations; in the case of 28 processes the collective operations account for 29.7% of the runtime while the point-to-point operations take 20.7%. The total time communicating for two processes is only 0.4% of the runtime while for 392 processes the total proportion is 72.0%. This equates to 35.9% of the runtime performing point-to-point communications, compared with 36.1% of the runtime performing collective operations.

Figure 2 shows the total point-to-point communication time for each process in the 28, 56, 112 and 392 process runs. In all cases, the only point-to-point communications are between near neighbours, indicating that this is the halo exchange. For larger executions, each process has between 7 and 26 near neighbours that it will communicate with.

Figure 3 shows the total message count for the number of processes. The observed message sizes for the HPCG mini-application are 8, 104, 208, 416, 832, 1352, 5408, 21632 and 86528. These are grouped in to the following groups: less than 100 bytes ( ); 100-1000 bytes ( ); 1000-10000 bytes ( ); and, greater than 10000 bytes ( ).

It can be seen in Figure 3 that at smaller process counts (2 and 28 processes) large messages are dominant. Network interconnects generally perform much better when messages are larger. As the application is strongly scaled, at large scale each process operates on a smaller local problem, reducing the size of the communications required, thus reducing the observed message sizes.

Figure 3 also shows that there are a greater number of messages even though the problem size remains constant. This combination of more, yet smaller, messages leads to a significant reduction in overall network performance. Further, as the problem is strong scaled, each process is performing less computational work leading

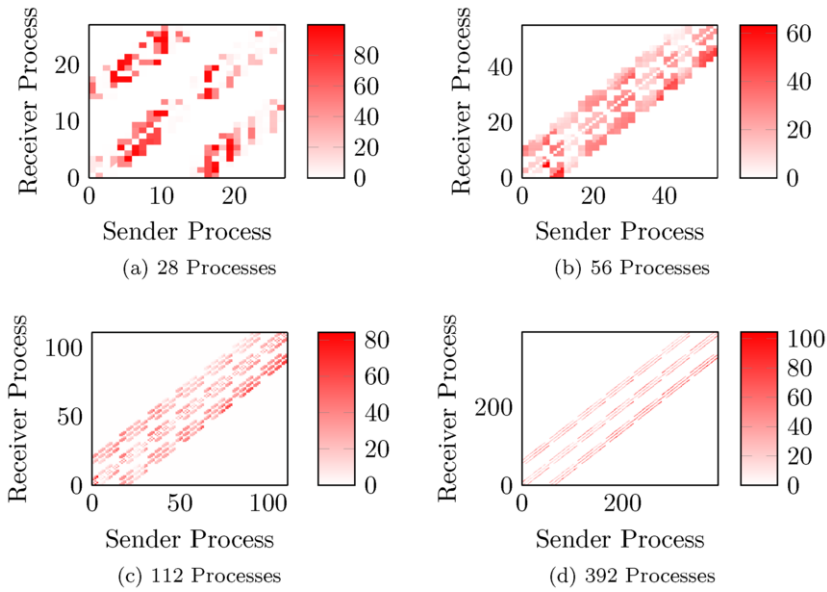

Fig. 2. Point-to-Point Communication Times (seconds)

Message Size	Transmission Speed (kbps)
Less than 100 bytes	416491.42
100-1000 bytes	5865413.33
1000-10000 bytes	28215160.00
Greater than 10000 bytes	70963500.00

Table 2
Benchmark transmission speeds for Orac

the communications time to become a bottleneck.

Figure 4 shows how the network transmission speed varies depending on the number of processing elements, for a messages of a given size. It is clear from Figure 4 that the smaller messages used by larger runs perform significantly worse than the larger messages used in smaller scale runs.

To better understand the achieved performance, the network throughput has additionally been benchmarked using the Intel MPI Benchmark suite across two nodes for point-to-point messages. These results can be seen in Table 2. This provides us with an achievable maximum performance we can expect and allows us to highlight any potential deviation from this.

A two process run of HPCG achieves the average transmission speed of only 0.26 Gbps; significantly under the 70.96 Gbps that can be achieved over the interconnect. As Orac is a shared system, network traffic from other applications running on the system can cause external contention. However, this does not explain the large difference in performance we observe.

Our final experiment is focussed on how the communication/computation balance changes with problem size. Figure 5 shows that for 392 cores the time for

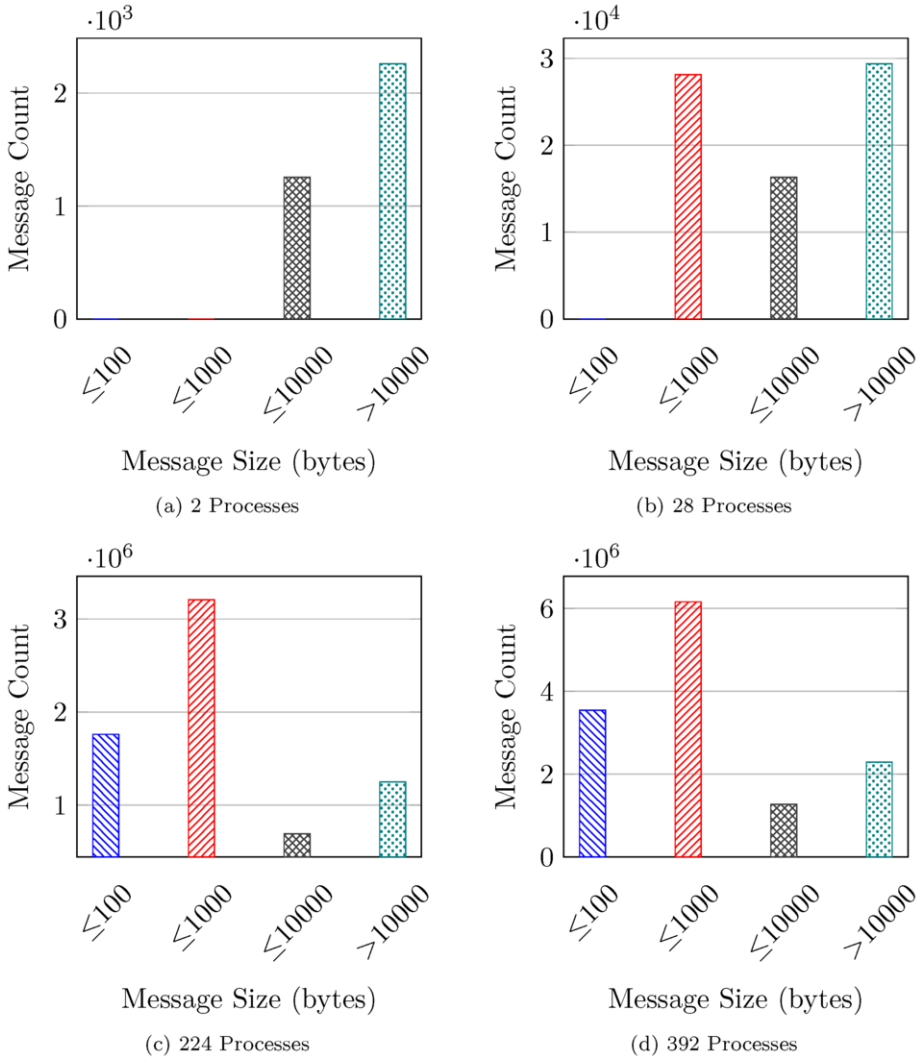

Fig. 3. Message Size against Message Count

collective operations remains close to 25% of the runtime of a application across a given number of cores. This is consistent despite changing problem sizes, since all collective operations use a fixed message size of 4.44 Kb. In line with our previous results, we also see that the time taken performing point-to-point operations is approximately equal to the time spend performing collective operations.

6 Conclusion

This paper documents our early investigation into the communication present in the HPCG application. Specifically, we show how the proportion of the program execution taken performing communications grows as the problem is scaled out, ultimately dominating performance at 392 cores. We also show that in the case of

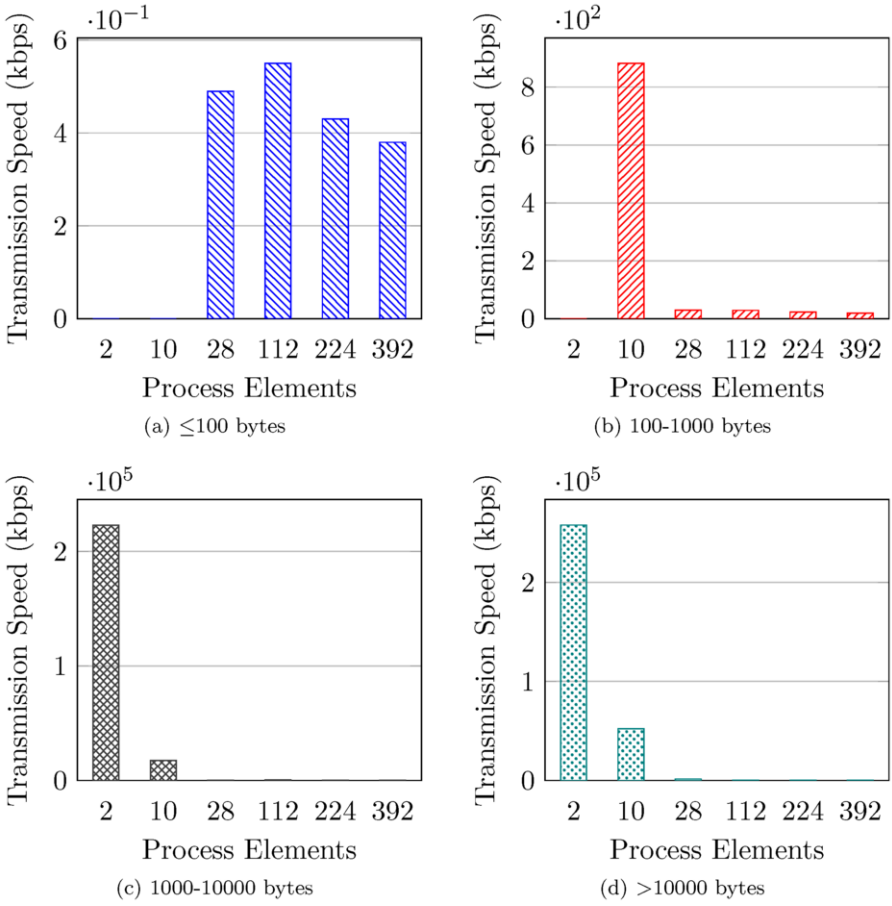

Fig. 4. Message size against average transmission speed

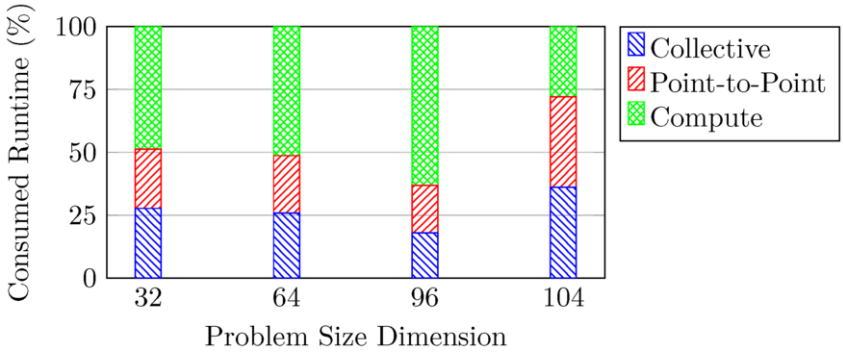

Fig. 5. The total portion of time for collective, point-to-point and compute operations across 392 cores

a large number of processes it spends nearly equal time performing point-to-point and collective MPI operations.

Further, it has been shown that when strong scaling a problem the message size decreases as the number of processes increases, this has an effect on the net-

work throughput because there are more processes trying to communicate with each other. The network throughput decrease has an overall negative effect on the performance of the system.

When the problem size is changed across a given machine size it has been shown that the proportion of time spent performing collective and point-to-point MPI operations remains approximately the same relative to the runtime as the same number of nodes need to communicate with each other.

6.1 Further Work

Building on the work in this paper, we aim to use the Structural Simulation Toolkit (SST) from Sandia National Laboratories to simulate linear solvers on varying systems. Using simulation, we can rapidly test different topologies, network fabrics, routing algorithms and traffic scheduling algorithms to find optimal configurations for CG-like applications. We can also investigate communication-avoidance techniques to determine how best to optimise the performance of linear solves on large-scale systems.

References

[1] Aubry, R., F. Mut, R. Löhner and J. R. Cebral, *Deflated Preconditioned Conjugate Gradient Solvers For the Pressure–Poisson Equation*, Journal of Computational Physics **227** (2008), pp. 10196–10208.

[2] Bailey, D. H., E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber et al., *The NAS Parallel Benchmarks*, The International Journal of Supercomputing Applications **5** (1991), pp. 63–73.

[3] Bergman, K., S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller et al., *Exascale Computing Study: Technology Challenges in Achieving Exascale Systems*, Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep **15** (2008).

[4] Department of Biophysical Chemistry of Groningen University, *GROMACS*, http://www.gromacs.org (accessed October 18, 2017) (2017).

[5] Dongarra, J. J., V. Eijkhout and H. v. d. Vorst, *An Iterative Solver Benchmark*, Scientific Programming **9** (2001), pp. 223–231.

[6] Dongarra, J. J. and M. A. Heroux, *Toward a New Metric for Ranking High Performance Computing Systems*, Sandia Report, SAND2013-4744 **312** (2013), p. 150.

[7] Dongarra, J. J., M. A. Heroux and P. Luszczek, *HPCG Benchmark: a New Metric for Ranking High Performance Computing Systems*, Electrical Engineering and Computer Sciente Department, Technical Report UT-EECS-15-736 (2015).

[8] Dongarra, J. J. and P. Luszczek, *HPCG Technical Specification*, Sandia Report SAND2013-8752 (2013).

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

[10] Forum, M., *MPI: A Message-Passing Interface Standard Version 3.1*, Technical report, MPI Forum (2015).

[11] Intel, *Getting Started with Intel® MPI Benchmarks 2018*, https://software.intel.com/en-us/articles/intel-mpi-benchmarks (accessed October 18, 2017) (2017).

[12] Intel, *Intel Parllel Studio XE 2017*, https://software.intel.com/parallel-studio-xe (accessed October 18, 2017) (2017).

[13] Intel, *Intel Trace Analyzer and Collector*, https://software.intel.com/en-us/intel-trace-analyzer (accessed October 18, 2017) (2017).

[14] John D. McCalpin, *STREAM: Sustainable Memory Bandwidth in High Performance Computers*, Technical report, University of Virginia, Charlottesville, Virginia (1991-2007).

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

[16] Mesoscale and Microscale Meteorology Division, NCAR, *MM5*, http://www2.mmm.ucar.edu/mm5/mm5-home.html (accessed October 18, 2017) (2017).

[17] Müller, M. S., A. Knüpfer, M. Jurenz, M. Lieber, H. Brunst, H. Mix and W. E. Nagel, *Developing Scalable Applications with Vampir, VampirServer and VampirTrace*, in: *Proceedings of the International Conference in Parallel Computing: Architectures, Algorithms and Applications (ParCo 2007)*, 1 **15**, RWTH Aachen University, Aachen, Germany, 2007, pp. 637–644.

[18] Pješivac-Grbović, J., T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel and J. J. Dongarra, *Performance Analysis of MPI Collective Operations*, Cluster Computing **10** (2007), pp. 127–143.

[19] Rajovic, N., P. M. Carpenter, I. Gelado, N. Puzovic, A. Ramirez and M. Valero, *Supercomputing with commodity CPUs: Are mobile SoCs ready for HPC?*, in: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ACM, 2013, p. 40.

[20] Reussner, R., P. Sanders, L. Prechelt and M. Müller, *SKaMPI: A Detailed, Accurate MPI Benchmark*, Lecture Notes in Computer Science (LNCS) **1497** (1998), pp. 52–59.

[21] Saad, Y., "Iterative Methods for Sparse Linear Systems," SIAM, 2003, second edition.

[22] Xie, G. and Y.-H. Xiao, *How to Benchmark Supercomputers*, in: *2015 14th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)*, IEEE, 2015, pp. 364–367.