Original Article

International Journal of HIGH PERFORMANCE COMPUTING APPLICATIONS

The International Journal of High Performance Computing Applications
1–8
© The Author(s) 2015
Reprints and permissions:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/1094342015593158
hpc.sagepub.com
SAGE

High-performance conjugate-gradient benchmark: A new metric for ranking high-performance computing systems

Jack Dongarra[1], Michael A Heroux[2] and Piotr Luszczek[3]

Abstract
We describe a new high-performance conjugate-gradient (HPCG) benchmark. HPCG is composed of computations and data-access patterns commonly found in scientific applications. HPCG strives for a better correlation to existing codes from the computational science domain and to be representative of their performance. HPCG is meant to help drive the computer system design and implementation in directions that will better impact future performance improvement.

Keywords
Preconditioned conjugate gradient, multigrid smoothing, additive Schwarz, HPC benchmarking, validation and verification

1 Introduction

Many aspects of the physical world may be modeled with partial differential equations (PDEs) and lend a hand to predictive capability so as to aid scientific discovery and engineering optimization. The high-performance conjugate-gradient (HPCG) benchmark is used to test a high-performance conjugate (HPC) machine's ability to solve these important scientific problems. To that end, the primary scope of the project is to measure the execution rate of Krylov subspace solvers on distributed-memory hardware. In doing so, HPCG aims to increase the prominence of sparse matrix methods and put them on an equal footing with other benchmarks of high-end machines.

Over the years, the field of iterative methods has grown in significance, and today it offers a wide range of algorithms that form the backbone of non-linear and differential equation solvers. HPCG aims to tackle the complexity of the field by offering a simple test that represents the performance characteristics of these algorithms. In particular, the conjugate-gradient algorithm and a symmetric Gauss–Seidel preconditioner were chosen for measurement and they are used to solve the Poisson differential equation on a regular three-dimensional grid discretized with a 27-point stencil.

The HPCG benchmark (Dongarra and Heroux, 2013) is a tool for ranking computer systems based on a simple additive Schwarz, symmetric Gauss–Seidel preconditioned conjugate-gradient solver. HPCG is similar in its purpose to HPL (Dongarra et al., 2003) which is currently used to rank systems as part of the TOP500 project (Meuer et al., 2013), but HPCG is intended to better represent how today's applications perform.

HPCG generates a regular sparse linear system that is mathematically similar to a finite element, finite volume or finite difference discretization of a three-dimensional heat diffusion equation on a semi-regular grid. The problem is solved using domain decomposition (Smith et al., 1996) with a conjugate-gradient method that uses an additive Schwarz preconditioner. Each subdomain is preconditioned using a symmetric Gauss–Seidel sweep.

The HPL benchmark (Dongarra et al., 2003) is one of the most widely recognized and discussed metrics for ranking high-performance computing systems. When HPL gained prominence as a performance metric in the early 1990s there was a strong correlation between its

[1]Department of Electrical Engineering and Computer Science, University of Tennessee, USA; Computer Science and Mathematics Division, Oak Ridge National Laboratory, ORNL School of Mathematics and School of Computer Science, University of Manchester, UK
[2]Scalable Algorithm Department, Sandia National Laboratories, Albuquerque, New Mexico
[3]Department of Electrical Engineering and Computer Science, University of Tennessee, USA

Corresponding author:
Piotr Luszczek, University of Tennessee, 1122 Volunteer Blvd Street 203, Knoxville, TN 37996-3450, USA.
Email: luszczek@eecs.utk.edu

predictions of system rankings and the ranking that full-scale applications would realize. Computer-system vendors pursued designs that would increase HPL performance, which would in turn improve overall application performance. Currently, HPL remains tremendously valuable as a measure of historical trends, and as a stress test, especially for the leadership class systems which are pushing the boundaries of current technology. Furthermore, HPL provides the HPC community with a valuable outreach tool, understandable to the outside world. Anyone with an appreciation of computing is impressed by the tremendous increases in performance that HPC systems have attained over the past few decades in terms of HPL. At the same time, HPL rankings of computer systems are no longer so strongly correlated to real application performance, especially for the broad set of HPC applications governed by differential equations, which tend to have much stronger needs for high bandwidth and low latency. This is tied to the irregular access patterns to data that these codes tend to exhibit. In fact, we have reached a point where designing a system for good HPL performance can actually lead to design choices that are wrong for the real application mix, or add unnecessary components or complexity to the system. Worse yet, we expect the gap between HPL predictions and real application performance to increase in the future. Potentially, the fast track to a computer system with the potential to run HPL at 1 Eflop/s (10^{18} floating-point calculations per second) is a design that may be very unattractive for our real applications. Without some intervention, future architectures targeted towards good HPL performance will not be a good match for our applications. As a result, we seek a new metric that will have a stronger correlation to our application base and will therefore drive system designers in directions that will enhance application performance for a broader set of HPC applications.

2 Related work

Similar benchmarks have been proposed and used before. In particular, the NAS Parallel Benchmarks (NPB) (Bailey et al., 1994; 1995, der Wijngaart, 2002) include a conjugate-gradient benchmark and it shares many attributes with the HPCG benchmark. Despite the wide use of this benchmark, it has the critical design decision that the matrix is chosen to have a random sparsity pattern with a uniform distribution of entries per row. This choice has led to the known side effect that a two-dimensional distribution of the matrix achieves optimal performance. Therefore, the computational and communication patterns are non-physical. Furthermore, no preconditioning is present, so the important features of a local sparse triangular solver are not represented and are not easily introduced, again because of the choice of using a non-physical sparsity pattern. Although the NPB conjugate gradient has been extensively used for HPC analysis, it does meet the criteria for our target application mix and, consequently, we do not consider it to be appropriate as a broad metric for our effort.

A lesser known but nonetheless relevant benchmark, the iterative solver benchmark (Dongarra et al., 2001), specifies the execution of a preconditioned conjugate gradient and generalized minimal residual (GMRES) iteration using physically meaningful sparsity patterns and several preconditioners. As such, its scope is broader than what we propose here, but this benchmark does not address scalable distributed-memory parallelism or nested parallelism.

The HPL benchmark (Dongarra et al., 2003) has been a yardstick of supercomputing performance for over four decades and a basis for the biannual TOP500 (Meuer et al., 2013) list of the 500 world's fastest supercomputer for over three decades. HPCG has a similar aim by measuring the computation and communication patterns currently prevalent in a vast number of applications of computational science at multiple scales of deployment. HPCG measures the performance of the sparse iterative solver in order to reward balanced system design as opposed to stressing a specific hardware components exercised by HPL. This has been elaborated in detail above.

The HPC Challenge Benchmark Suite (Dongarra and Heroux, 2013, Luszczek et al., 2006, Luszczek and Dongarra, 2010) has established itself as a performance measurement framework with a comprehensive set of computational and, more importantly, memory-access patterns that build on the popularity and relevance of HPL but add a much richer view of benchmarked hardware. In comparison to HPCG, the most differentiating factor tends to be the focus on a multidimensional view of the tested system that does not focus on a single figure of merit. Instead, the HPC Challenge delivers a suite of performance metrics that may be filtered out, combined or singled out according to the end user needs and application profiles. Also of importance is the fact that the HPC Challenge does not include a component that measures sparse-solver performance directly but instead it would have to be derived out of various bandwidth and latency measurements and performed across the memory hierarchy and the communication interconnect.

3 Background and goals

HPCG is designed to measure performance that is representative of many important scientific calculations, with low computation to data-access ratios, which we call Type 1 data access patterns. To simulate these patterns that are commonly found in real applications, HPCG exhibits the same irregular access to memory and fine-grain recursive computations.

In contrast to the new HPCG metric, the HPL is a program that factors and solves a large dense system of linear equations using Gaussian elimination with partial pivoting. The dominant calculations in this algorithm are dense matrix–matrix multiplication and related kernels, which we call Type 2 patterns. With proper organization of the computation, data access is predominantly unit stride and its cost is mostly hidden by concurrently performing computations on previously retrieved data. This kind of algorithm strongly favors computers with very high floating-point computation rates and adequate streaming memory systems. The performance issues related to the Type 1 patterns may be fully eliminated when the code only exhibits Type 2 patterns, and this may lead hardware designers to not include Type 1 patterns in the design decisions for next-generation systems.

In general, we advocate that a well-rounded computer system should be designed to execute both Type 1 and Type 2 patterns efficiently, as this combination allows the system to run a broad mix of applications and run them well. Consequently, for a meaningful metric to test the true capabilities of a general-purpose computer, it should stress both Type 1 and Type 2 patterns; however, HPL only stresses Type 2 patterns, and, as a metric, is incapable of measuring Type 1 patterns.

Another issue with the existing performance metrics stems from the emergence of accelerators, which are extremely effective (relative to CPUs) with Type 2 patterns, but much less so with Type 1 patterns. This is related to the divide that exists between massively parallel throughput workloads and the latency sensitive ones. For many users, HPL results show a skewed picture relative to the Type 1 application performance, especially on machines that are heavily Type 2 biased, like a machine that features accelerators for the majority of the computational power.

For example, the Titan system at Oak Ridge National Laboratory has 18,688 nodes, each with a 16-core, 32 GiB AMD Opteron processor and a 6 GiB NVIDIA K20 GPU (Facility, 2013b). Titan was the top-ranked system on TOP500 in November 2012 using HPL; however, in obtaining the HPL result on Titan, the Opteron processors played only a supporting role in the result. All floating-point computation and all data was resident on the GPUs. In contrast, real applications, when initially ported to Titan, will typically run solely on the CPUs and selectively off-load computations to the GPU for acceleration (Facility, 2013a, Joubert et al., 2009).

The HPCG Benchmark can help alleviate many of the problems described above by using the following principles.

1. **Provides coverage of the major communication and computational patterns**: the major communication patterns (both global and neighborhood collectives) and computational patterns (vector updates, dot products, sparse matrix–vector multiplications, and local triangular solver) from our production differential equation codes, both implicit and explicit, are present in this benchmark. Emerging asynchronous collectives and other latency-hiding techniques can be explored in the context of HPCG and aid in their adoption and optimization in future systems.
2. **Represents a minimal collection of the major patterns**: HPCG is the smallest benchmark code containing these major patterns while at the same time representing a real mathematical computation (which aids in the validation and verification efforts).
3. **Rewards investment in high performance of collectives**: neighborhood and all-reduce collectives represent essential performance bottlenecks for our applications and can benefit from high-quality system design, improving the performance of HPCG will improve the performance of real applications.
4. **Rewards investment in local-memory system performance**: the local-processor performance of HPCG is largely determined by the effective use of the local-memory system. Improvements in the implementation of HPCG data structures, the compilation of HPCG code and the performance of the underlying system, will improve the HPCG benchmark results and real application performance, along with informing application developers of new approaches to optimizing their own implementations.

Any new metric we introduce must satisfy a number of requirements. Two overarching goals for the metric are as follows.

1. To accurately predict the system rankings for a target suite of applications: the ranking of computer systems using the new metric must correlate strongly to how our real applications would rank these same systems.
2. To drive improvements to computer systems to benefit relevant applications: the metric should be designed so that, as we try to optimize metric results for a particular platform, the changes will also lead to better performance in the identified real applications; furthermore, computation of the metric should drive system reliability in ways that help the applications.

4 CG iteration setup and execution

The HPCG benchmark generates a synthetic discretized three-dimensional PDE model problem (Mattheij et al., 2005), and computes preconditioned conjugate-gradient iterations for the resulting sparse linear system. The model problem can be interpreted as a single degree of freedom heat diffusion equation with zero Dirichlet boundary conditions. The PDE is discretized with a finite-difference scheme on a three-dimensional rectangular grid domain with fixed spacing of the nodes.

The global domain dimensions are $(n_x \times P_x) \times (n_y \times P_y) \times (n_z \times P_z)$ where $n_x \times n_y \times n_z$ are the local subgrid dimensions in the x, y and z dimensions, respectively, assigned to each Message Passing Interface (MPI) process. The local grid dimensions are read from the data file hpcg.dat, or could also be passed in as command line arguments. The dimensions $P_x \times P_y \times P_z$, constitute a factoring of the MPI process space that is computed automatically in the HPCG setup phase. We impose ratio restrictions on both the local and global x, y and z dimensions, which are enforced in the setup phase of HPCG.

HPCG then performs m sets of n iterations, using the same initial guess each time, where m and n are sufficiently large to test the system resilience and ability to remain operational. By doing this, we can compare the numerical results for 'correctness' at the end of each of the m sets. A single-set computation is shown in Algorithm 1.

The setup phase constructs a logically global, physically distributed sparse linear system using a 27-point stencil at each grid point in the three-dimensional domain, such that the equation at point (i, j, k) depends on the values of its location and 26 surrounding neighbors. The matrix is constructed to be weakly diagonally dominant for the interior points of the global domain, and strongly diagonally dominant for the boundary points, reflecting a synthetic conservation principle for the interior points and the impact of zero Dirichlet boundary values on the boundary equations. The resulting sparse linear system has the following properties:

Algorithm 1: Preconditioned conjugate-gradient algorithm used by HPCG.

```
$\vec{p}_0 \leftarrow \vec{x}_0, \vec{r}_0 \leftarrow \vec{b} - A\vec{p}_0$
for $i = 1, 2,$ to max_iterations do
    $\vec{z}_i \leftarrow M^{-1}\vec{r}_{i-1}$
    if $i = 1$ then
        $\vec{p}_i \leftarrow \vec{z}_i$
        $\alpha_i \leftarrow \mathrm{dot\_prod}(\vec{r}_{i-1}, \vec{z}_i)$
    else
        $\alpha_i \leftarrow \mathrm{dot\_prod}(\vec{r}_{i-1}, \vec{z}_i)$
        $\beta_i \leftarrow \alpha_i / \alpha_{i-1}$
        $\vec{p}_i \leftarrow \beta_i \vec{p}_{i-1} + \vec{z}_i$
    $\alpha_i \leftarrow \mathrm{dot\_prod}(\vec{r}_{i-1}, \vec{z}_i) / \mathrm{dot\_prod}(\vec{p}_i, A\vec{p}_i)$
    $\vec{x}_{i+1} \leftarrow \vec{x}_i + \alpha_i \vec{p}_i$;
    $\vec{r}_i \leftarrow \vec{r}_{i-1} - \alpha_i A\vec{p}_i$
    if $\|\vec{r}_i\|_2 <$ tolerance then
        STOP
```

- a sparse matrix with 27 nonzero entries per row for interior equations and 7 to 18 nonzero terms for boundary equations;
- a symmetric, positive definite, nonsingular linear operator;
- the boundary condition is reflected by subtracting one from the diagonal.
- a generated known exact solution vector with all values equal to one;
- a matching right-hand-side vector; and
- an initial guess of all zeros.

The central purpose of defining this sparse linear system is to provide a rich vehicle for executing a collection of important computational kernels; however, the benchmark is not about computing a high-fidelity solution to this problem. In fact iteration counts are fixed in the benchmark code and we do not expect convergence to the solution, regardless of problem size. We do use the spectral properties of both the problem and the preconditioned conjugate-gradient algorithm as part of software verification.

The conjugate-gradient method allows the code to maintain the orthogonality relationship with a short three-term recurrence formula. This, in turn, allows the linear system data to be scaled arbitrarily without worrying about the excessive growth of storage requirements for the orthogonal basis.

The regularity of the discretization grid of the model PDE gives plenty of opportunity to optimize the sparse data structure for efficient computation. There are known results of how to optimally partition and reorder the mesh points to achieve good load balance, small communication volume and good local performance. We feel that allowing such optimizations would violate the spirit of the benchmark and trivialize its results. Instead, we insist that the knowledge of the regularity of the problem should not be taken into consideration when porting and optimizing the code for the user machine. The discretization should be treated as a generic mesh without any properties known a priori. In exchange, the users may take advantage of the simplicity of the mesh to find problems with their optimizations since many aspects of the optimal solution are known in closed form and can serve as a useful debugging tool.

In a similar fashion, we prohibit the use of knowledge of the problem when performing the conjugate-gradient iteration. However, we recognize that the users may wish to use the knowledge of the spectrum of the discretization matrix to estimate the accuracy of their optimized solver.

5 Elements of multigrid and coarse grid solve

The multigrid method is considered by many as being ideally suited for elliptic PDEs but by varying the discretization it is possible to apply successfully to a much larger class of linear and non-linear PDEs (Trottenberg et al., 2001: 2). As described so far, HPCG does directly characterize all of the computational and communication patterns exhibited by multigrid solvers. Specifically, the dominant performance bottleneck at coarse-grid levels is latency rather than bandwidth that dominates the message exchanges at the fine grid levels and dot products of the preconditioned conjugate-gradient method. For that reason, version 2.0 of HPCG introduced a multigrid component in the reference code to model the behavior of multi-level methods. The problem that we faced when introducing this new functionality was the potential of a substantial increase in the code complexity. To minimize the impact of the change, we reused the existing components and recast them in terms of commonly used parts of a typical multigrid solver. The smoother/solver for all of the levels of our simulated geometric multigrid is the Gauss–Seidel (locally) preconditioned conjugate-gradient solver. The mesh coarsening and refinement (restriction and prolongation) is done based on halving the number of points in every dimension and thus each coarse-grid level has eight times as few points as the neighboring fine-grid level.

Just as in the case of the preconditioned conjugate gradient, our goal is only to provide basic components, rather than a complete multigrid solver. Consequently, we do not include either the full V nor W cycles and neither do we perform an accurate solve at the coarsest grid level. Instead we limit the number of grid levels to three, resulting in a 256-fold reduction in the number of grid points, which is sufficient to address most of the bandwidth-latency bottlenecks and expose the performance of common algorithmic trade-offs. We also captured in this limited implementation the prevalent recursive patterns of code execution and the integer arithmetic required to capture some of the mesh manipulation.

6 Validation and verification components

HPCG detects and measures variances from bitwise identical computations because it is widely believed that future computer systems will not be able to provide deterministic execution paths for floating-point computations. Because floating-point addition is not associative, thus we may not have bitwise reproducible results, even when running the same exact computation twice on the same number of processors of the same system. This is in contrast with many of the MPI-only applications today, and presents a big challenge to applications that must certify their computational results and conduct debugging in the presence of bitwise variability. HPCG makes the deviation from bitwise reproducibility apparent.

To detect anomalies during the iteration phases, HPCG computes preconditions, post conditions, and invariants. These are likely to eliminate the majority of errors that might creep in when implementing an optimized version of the benchmark.

The computational kernels in HPCG may be optimized by the end user to fully take advantage of the tested hardware. A reference code that we provide is focused on portability, which may often have negative effects on the performance of a specific system. To validate the user-provided kernels, HPCG includes a symmetry test for the sparse matrix multiply with discretization matrix A, $|x^tAy - y^tAx|$, and for the symmetric Gauss–Seidel preconditioner M, $|x^tMy - y^tMx|$.

A spectral test is also included in HPCG to test for the fast convergence of the conjugate-gradient algorithm on a modified matrix A that is close to being diagonal. The theoretical framework underlying the conjugate-gradient solver (Saad, 2003) guarantees a short and fixed number of iterations for such matrices and the invalid optimizations attempted by the user should violate this property. The spectral test is meant to detect potential anomalies in the optimized implementation related to inaccurate calculations and convergence rate changes due to user-defined matrix ordering.

7 Allowed and disallowed optimizations

Good performance results from a HPCG run may only be achieved after hardware specific optimizations. Unfortunately, in the reference implementation of the benchmark, it is nearly impossible to keep up with hardware progress and include the optimizations required on the contemporary supercomputing platforms. Instead, we aim for simplicity of the reference implementation and offer here a number of ideas for improving performance for user runs.

One of the performance-critical aspects of efficient sparse computations is partitioning and ordering of the mesh points. By default, HPCG uses the lexicographical ordering; however, the user can change this in order to achieve more optimal results. For example, using red–black ordering is especially beneficial in the Gauss–Seidel preconditioner that is inherently sequential without the appropriate renumbering of elements. The numbering scheme established by the user before the iterations begin is carried throughout the timed computations in user-defined data structures and is passed to the computational kernels that may then take advantage of the user ordering by providing a specialized kernel.

Another likely source of improved performance could be the use of system-specific communication infrastructure: both the hardware and the software that takes full advantage of the communication network.

The reference implementation uses a small set of MPI functions that are very likely to be portable across a wide range of distributed-memory systems and, if optimized, will deliver a good portion of the optimal performance; however, the custom implementations of HPCG are likely to contain a bigger variety of communication options. In MPI, there is a possibility of improving performance with various communication modes, such as one-sided, buffered, ready or persistent. It is also possible to use newer additions to the MPI standard such as the neighborhood collectives (Hoefler et al., 2007), which has by now become much more prevalent and thus is likely to be optimized for the interconnect hardware by the system vendors or integrators. There of course also exists the possibility to go beyond the MPI standard and use lower-level application program interfaces such as the Common Communication Interface. This might not be an option for large code bases but it would be feasible within the context of HPCG where only a handful of communication scenarios are used. We do not envision at this point the need to use vendor-specific interfaces and the reference implementation is restricted to the widely implemented subset of the MPI standard.

The commonly used optimization in sparse iterative methods is aimed at matrix–vector multiplication (Byun et al., 2012, Im et al., 2004, Liu et al., 2013, Vuduc et al., 2005). As with other optimizations, we opt not to include hardware-specific code in the reference implementation and instead we provide the user with a set of interfaces and data structures that allow them to easily include many of the existing implementations of these computational kernels.

In order to maintain the wide applicability of the HPCG results and optimizations, we explicitly prohibit the use of knowledge of either the sparsity pattern of the discretization of the matrix (this includes the symmetry of the discretization), its structure and connectivity pattern, or the dimensionality of the domain. In our view, this invites the use of generic methods for matrix partitioning and hardware-specific optimization of the computational kernels.

At a higher level of abstraction, the knowledge of the spectral properties of the matrix could be used to artificially accelerate the conjugate-gradient iterations or provide a nearly optimal preconditioner. This might strike as an artificial constraint, because in practice it is always beneficial to take advantage of any numerical properties of the matrix; however, for an unknown problem structure and spectrum it is usually more costly to obtain this kind of information rather than to perform conjugate-gradient iteration barring any knowledge that can come from the domain that originated the PDE. In a similar vein, we do not allow the use of variants of the conjugate-gradient method that completely bypass the challenging aspects of the classic rendition of the algorithm. Some examples of this would be the reordered variants of the conjugate-gradient (Chronopoulos and Gear, 1989, D'Azevedo et al., 1993, Dongarra and Eijkhout, 2003, Eijkhout, 1992, Meurant, 1987) or pipelined conjugate-gradient (Ghysels and Vanroose, 2012).

8 Performance results

HPCG has already been run on a number of large-scale supercomputing installations in Europe, Japan and the USA. It is not feasible to list them all here for lack of space and also due to the early nature of the results, as the community is gaining experience in running the code. Instead, we present very preliminary results from a fairly small-scale deployment. This is presented in Figure 1 and should not be interpreted as official results for the tested system but rather as a preliminary comparison between the results that can be expected from HPCG and what is commonly reported as a result for HPL. The figure clearly shows a number of trends. Firstly, HPL follows relatively closely the peak performance of the machine – a fact well known to the benchmarking practitioners and most HPC experts. Secondly, HPCG exhibits performance levels that are far below the levels seen by HPL; again, this hardly comes as a surprise to anybody in the high-end and supercomputing fields and may be attributed to many factors with the most commonly cited one being the so-

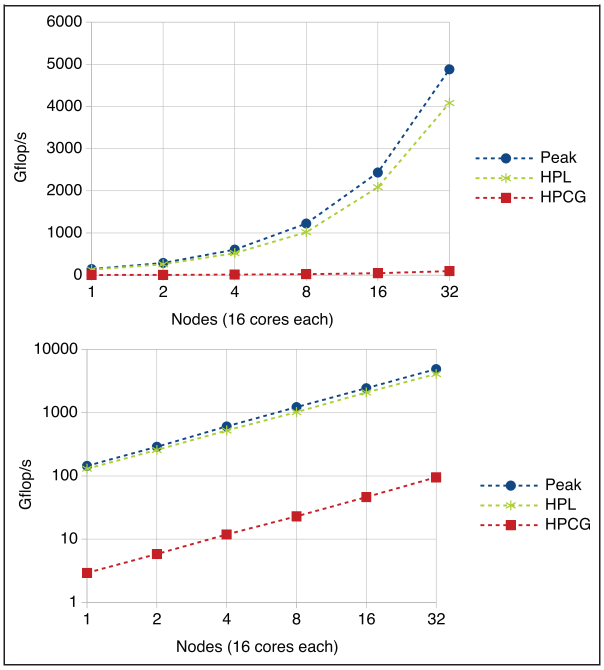

Figure 1. One of the early performance results from running HPCG and HPL on up to 32 nodes or 512 MPI processes. Two versions are provided, with the semi-logarithmic scale (top) and logarithmic scale (bottom).

called ‘memory wall.’ Finally, it is worth noting that, despite low absolute values, HPCG scales equally well when compared with HPL, which might be attributed to the custom interconnection of the tested system.

9 Future work

Future work will include a thorough validation testing of the HPCG benchmark against a suite of applications on current high-end systems using techniques similar to those identified in the Mantevo project (Heroux et al., 2009). Furthermore, we plan to fully specify opportunities and restrictions on changes to the reference version of the code, to ensure that only changes that have relevance to our application base are permitted.

Acknowledgements

The authors thank the Department of Energy National Nuclear Security Agency for the funding provided for this work.

We also thank Simon Hammond, Mahesh Rajan, Doug Doerfler and Christian Trott for their efforts to test early versions of HPCG and for giving valuable feedback.

Funding

This work was supported by the US Department of Energy (grant number 14-1589).

References

Bailey D, Barscz E, Barton J, et al. (1994) The NAS parallel benchmarks. Technical Report no. RNR-94-007, NASA Ames Research Center, USA.

Bailey D, Harris T, Saphir W, et al. (1995) The NAS parallel benchmarks 2.0. Techinical Report no. NAS-95-020, NASA Ames Research Center, USA.

Byun JH, Lin R, Yelick KA, et al. (2012) Autotuning sparse matrix-vector multiplication for multicore. Technical Report no. UCB/EECS-2012-215, University of California, USA.

Chronopoulos A and Gear C (1989) s-Step iterative methods for symmetric linear systems. *Journal of Computational and Applied Mathematics* 25: 153–168.

D’Azevedo E, Eijkhout V and Romine C (1993) LAPACK working note 56: Reducing communication costs in the conjugate gradient algorithm on distributed memory multiprocessor. Technical Report no. CS-93-185, University of Tennessee, Knoxville, USA.

der Wijngaart RFV (2002) NAS parallel benchmarks version 2.4. Technical Report no. NAS-02-007, Computer Sciences Corporation, NASA Advanced Supercomputing (NAS) Division, USA, October.

Dongarra J and Eijkhout V (2003) Finite-choice algorithm optimization inconjugate gradients. Technical Report no. 159, LAPACK Working Note, University of Tennessee, USA.

Dongarra J, Eijkhout V and van der Vorst H (2001) Iterative solver benchmark. *Scientific Programming* 9(4): 223–231.

Dongarra J and Heroux M (2013) Toward a new metric for ranking high performance computing systems. Technical Report no. SAND2013-4744, Sandia National Laboratories, USA.

Dongarra JJ, Luszczek P and Petitet A (2003) The LINPACK benchmark: Past, present, and future. *Concurrency and Computation: Practice and Experience* 15(9): 803–820.

Eijkhout V (1992) LAPACK working note 51: Qualitative properties of the conjugate gradient and Lanczos methods in a matrix framework. Technical Report no. CS 92-170, University of Tennessee, USA.

ORNL Leadership Computing Facility (2013a) Annual Report 2012–2013, Available at: https://www.olcf.ornl.gov/wp-content/uploads/2014/03/2013_ARv2M.pdf (accessed 10 August 2015).

ORNL Leadership Computing Facility (2013b) Introducing Titan — the world’s #1 open science supercomputer, Available at: http://www.olcf.ornl.gov/titan (accessed 29 May 2013).

Ghysels P and Vanroose W (2012) Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm. Technical Report no. 12.2012.1, Intel Labs Europe. Presented at *PRECON13*, June 19–21, 2013, Oxford, UK.

Heroux MA, Doerfler DW, Crozier PS, et al. (2009) Improving performance via mini-applications. Technical Report no. SAND2009-5574, Sandia National Laboratories.

Hoefler T, Gottschling P, Lumsdaine A, et al. (2007) Optimizing a Conjugate Gradient Solver with Non-Blocking Collective Operations. *Elsevier Journal of Parallel Computing* 33(9): 624–633.

Im EJ, Yelick K and Vuduc R (2004) Sparsity: Optimization framework for sparse matrix kernels. *International Journal of High Performance Computing Applications* 18(1): 135–158.

Joubert W, Kothe D and Nam HA (2009) Preparing for exascale: ORNL leadership computing facility application requirements and strategy. Technical Report no. ORNL/TM-2009/308, Oak Ridge National Laboratory, USA, December.

Liu X, Smelyanskiy M, Chow E, et al. (2013) Efficient sparse matrix-vector multiplication on x86-based many-core processors. In: *ICS’13*, Eugene, OR, 10–14 June 2013.

Luszczek P and Dongarra J (2010) Analysis of various scalar, vector, and parallel implementations of RandomAccess. Technical Report no. ICL-UT-10-03, Innovative Computing Laboratory, USA.

Luszczek P, Dongarra J and Kepner J (2006) Design and implementation of the HPCC benchmark suite. *CT Watch Quarterly* 2(4): 18–23.

Mattheij RMM, Rienstra SW and ten Thije Boonkkamp JHM (2005) *Partial Differential Equations, Modeling, Analysis, Computation*. Philadelphia: SIAM.

Meuer HW, Strohmaier E, Dongarra JJ, et al. (2013) TOP500 supercomputer sites, 42nd ed. Avaliable from: http://www.netlib.org/benchmark/top500.html (accessed 10 August 2015).

Meurant G (1987) Multitasking the conjugate gradient method on the CRAY X-MP/48. *Parallel Computing* 5: 267–280.

Saad Y (2003) *Iterative Methods for Sparse Linear Systems*. 2nd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics.

Smith BF, Bjørstad PE and Gropp WD (1996) *Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge, MA: Cambridge University Press.

Trottenberg U, Oosterlee CW and Schüller A (2001) *Multigrid*. London: Academic Press.

Vuduc R, Demmel J and Yelick K (2005) OSKI: A library of automatically tuned sparse matrix kernels. In: Proceedings of SciDAC 2005, Journal of Physics: Conference Series, San Francisco, CA, 2005, pp. 51–530. Bristol, UK: IOPscience.

Authors' Bibliographies

Jack Dongarra holds appointments at the University of Tennessee, Oak Ridge National Laboratory, and the University of Manchester. He specializes in numerical algorithms in linear algebra, parallel computing, use of advanced computer architectures, programming methodology and tools for parallel computers. His contributions to the HPC field have received numerous recognitions including the IEEE Sid Fernbach Award (2004), the first IEEE Medal of Excellence in Scalable Computing (2008), the first SIAM Special Interest Group on Supercomputing's award for Career Achievement (2010) and the IEEE IPDPS 2011 Charles Babbage Award. He is a fellow of the AAAS, ACM, IEEE and SIAM and a member of the National Academy of Engineering.

Michael A Heroux is a Distinguished Member of the Technical Staff at Sandia National Laboratories, working on new algorithm development, and the robust parallel implementation of solver components for problems of interest to Sandia and the broader scientific and engineering community. He leads development of the Trilinos Project, an effort to provide state of the art solution methods in a state of the art software framework. Dr Heroux also works on the development of scalable parallel scientific and engineering applications and maintains his interest in the interaction of scientific/engineering applications and high-performance computer architectures. He leads the Mantevo project, which is focused on the development of open source, portable mini-applications and mini-drivers for scientific and engineering applications. Dr Heroux is also the lead developer and architect of the HPCG benchmark, intended as an alternative ranking for the TOP 500 computer systems.

Piotr Luszczek is a Research Director at the University of Tennessee. His research interests are in large-scale parallel algorithms, numerical analysis, and high-performance computing. He has been involved in the development and maintenance of widely used software libraries for numerical linear algebra. In addition, he specializes in computer benchmarking of supercomputers using codes based on linear algebra, signal processing and PDE solvers.