HPCG Benchmark: a New Metric for Ranking High Performance Computing Systems[*]

Jack Dongarra[†] Michael A. Heroux[‡] Piotr Luszczek[§]

January 14, 2015

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 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 to aid the scientific discovery and engineering optimization. The HPCG benchmark is used to test an 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 3D grid discretized with a 27-point stencil.

The High Performance Conjugate Gradient (HPCG) benchmark [1] 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 High Performance LINPACK (HPL) [2] currently used to rank systems as part of the TOP500 project [3], but HPCG is intended to better represent how today's applications perform.

HPCG generates a regular sparse linear system

[*]This research was supported by the U.S. Department of Energy through award number 14-1589.

[†]Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, USA; Computer Science and Mathematics Division, Oak Ridge National Laboratory, USA; School of Mathematics and School of Computer Science, University of Manchester, UK

[‡]Scalable Algorithms Department, Sandia National Laboratories, USA

[§]Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, USA 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 [4] with a conjugate gradient method that uses an additive Schwarz preconditioner. Each subdomain is preconditioned using a symmetric Gauss-Seidel sweep.

The High Performance LINPACK (HPL) benchmark [2] is one of the most widely recognized and discussed metric 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 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. Presently, HPL remains tremendously valuable as a measure of historical trends, and as a stress test, especially for leadership class systems that 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 for computing is impressed by the tremendous increases in performance that HPC systems have attained over the past several 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[1] is a design that may be very unattractive for our real applications. Without some intervention, future architectures targeted toward 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) [5, 6, 7] includes a CG benchmark. It shares many attributes with HPCG. 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 solve is not represented and is not easily introduced, again because of the choice of a non-physical sparsity pattern. Although NPB CG has been extensively used for HPC analysis, it does meet the criteria for our target application mix and, consequently, we do not consider as an appropriate as a broad metric for our effort.

A lesser-known but nonetheless relevant benchmark, the Iterative Solver Benchmark [8] specifies the execution of a preconditioned CG and GMRES (Generalized Minimal RESidual) 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

[1] 10^{18} floating-point calculations per second

memory parallelism or nested parallelism.

High Performance LINPACK benchmark [2] has been a yardstick of supercomputing performance for over 4 decades and a basis for biannual TOP500 [3] list of the 500 world's fastest supercomputer for over 3 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 (HPCC) benchmark suite [9, 10, 1] 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 adds a much richer view of the 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, 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 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 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 accesses to memory and fine-grain recursive computations.

In contrast to the new HPCG metric, 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 the Type 2 patterns and this may lead the hardware designers not to include the Type 1 patterns in the design decisions for the next generations 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 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 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 [11]. 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 were 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 [12, 13].

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

- **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 solves) 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 on future systems.
- **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 Validation and Verification efforts).
- **Rewards investment in high-performance of collectives:** neighborhood and all-reduce collectives represent essential performance bottlenecks for our applications that can benefit from high-quality system design. Improving the performance of HPCG will improve the performance of real applications.
- **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, compilation of HPCG code, and the performance of the underlying system will improve HPCG benchmark results and real application performance, and will inform application developers on new approaches to optimizing their own implementations.

Any new metric we introduce must satisfy a number of requirements. Two overarching goals are:

1. Accurately predict system rankings for 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. 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 partial differential equation model problem [14], 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 3D 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 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

Algorithm 1: Preconditioned Conjugate Gradient algorithm used by HPCG.

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

$P_x \times P_y \times P_z$, 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 3D domain such that the equation at point (i,j,k) depends on the values at its location and its 26 surrounding neighbors. The matrix is constructed to be weakly diagonally dominant for interior points of the global domain, and strongly diagonally dominant for 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:

- 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 1 from the diagonal.
- A generated known exact solution vector with all values equal to 1.0.
- A matching right-hand-side vector.
- 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 CG 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 CG iteration. But 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 [15, p.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 CG 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 a 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 Gauss-Seidel (locally) preconditioned CG solver. The mesh coarsening and refinement (restriction and prolongation) is done based halving the number of points in every dimension and thus each coarse grid level has 8 times as few points as the neighboring fine grid level.

Just was the case with the preconditioned CG, our goal is only to provide basic components rather than a complete a multigrid solver. Consequently, we do not include neither the full V nor W cycles (named after their shape in the grid mesh hierarchy) and neither we perform an accurate solve at the coarsest grid level. Instead we limit the number of grid levels to 3, which results in 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 our 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 a 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 performance on a specific system. To validate the user-provided kernels, HPCG includes a symmetry test for the sparse matrix multiply with discretization matrix A: $|x^t Ay - y^t Ax|$, and for the symmetric Gauss-Seidel preconditioner M: $|x^t My - y^t Mx|$.

A spectral test is also included in HPCG to test for fast convergence of the CG algorithm on a modified matrix A that is close to being diagonal. The theoretical framework underlying the CG solver [16] 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 an 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 the hardware progress and include the optimizations required on the contemporary supercomputing platforms. Instead, we aim at 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 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 the Message Passing Interface (MPI) functions that are very likely to portable across a wide range of distributed memory systems and, if optimized, will deliver a good portion of the optimal performance. But the custom implementations of HPCG are likely to contain a bigger variety of communication options. In MPI, there is a possibility to improve 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 [17], which has by now become much more prevalent and thus are 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 APIs such as the Common Communication Interface (CCI). 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 of using 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 the matrix-vector multiplication [18, 19, 20, 21]. As with other optimizations, we opted for not including hardware-specific code in the reference implementation and instead we provide the user with a set of interfaces and data structures that allow the user to easily include many of the existing implementation of this computational kernels.

In order to maintain 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, nor 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 that could be used to artificially accelerate the CG iterations or provide nearly optimal preconditioner. This might strike as an artificial constrain 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 CG 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 CG method that completely bypass the challenging aspects of the classic rendition of the algorithm. One example of this would be the reordered variants of CG [22, 23, 24, 25, 26] or pipelined CG [27].

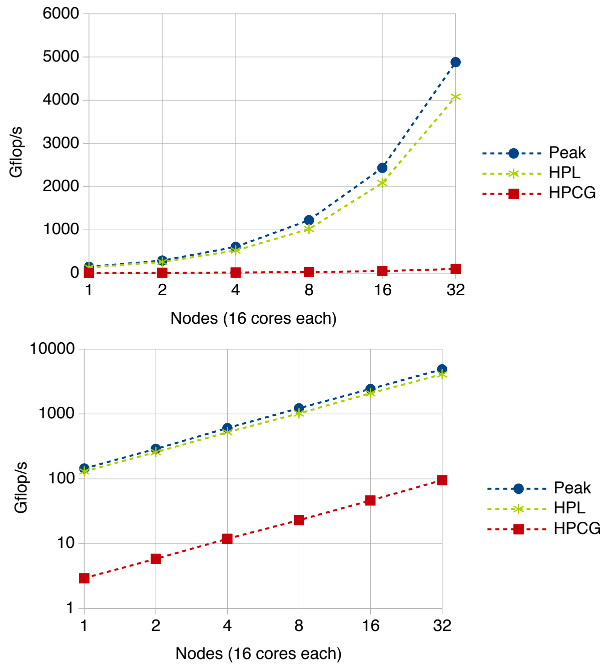

Figure 1: One of the early performance results from running HPCG and HPL on up to 32 nodes or 512 MPI processes. Two version are provided: with the semi-logarithmic scale (top) and logarithmic scale (bottom).

8 Performance Results

HPCG has already been run on a number of large-scale supercomputing installations in Europe, Japan, and the US. It is not feasible to list them all here for lack of space but also due to the early nature of the results as the community is gaining experience in running the code. Instead, we are presenting very preliminary results from a fairly small-scale deployment. This is presented in Figure 1 and should not be interpreted as an official 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 know 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 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 interconnect of the tested system.

9 Future Work

The future work includes 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 [28]. 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.

Acknowledgments

The authors thank the Department of Energy National Nuclear Security Agency for 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 give valuable feedback.

References

[1] Jack Dongarra and Michael Heroux. Toward a new metric for ranking high performance computing systems. Technical Report SAND2013-4744, Sandia National Laboratories, 2013.

[2] Jack J. Dongarra, Piotr Luszczek, and Antoine Petitet. The linpack benchmark: Past, present, and future. *Concurrency and Computation: Practice and Experience*, 15(9):803–820, August 10 2003. DOI: 10.1002/cpe.728. ISSN 1532-0634.

[3] Hans W. Meuer, Erich Strohmaier, Jack J. Dongarra, and Horst D. Simon. TOP500 supercomputer sites, 42nd edition, November 2013. The report can be downloaded from http://www.netlib.org/benchmark/top500.html (accessed 10 August 2015).

[4] Barry F. Smith, Petter E. Bjørstad, and William D. Gropp. *Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, Cambridge, MA, USA, 1996.

[5] David Bailey, Eric Barszcz, J. Barton, D. Browning. R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, Horst Simon, V. Venkatakrishnan, and S. Weeratunga. The nas parallel benchmarks. Technical Report NAS Technical Report RNR-94-007, NASA Ames Research Center, Moffett Field, CA, 1994.

[6] D.H. Bailey, T. Harris, W.C. Saphir, R. F. Van der Wijngaart, A.C. Woo, and M. Yarrow. The nas parallel benchmarks 2.0. Technical Report NAS-95-020, NASA Ames Research Center, Moffett Field, CA, 1995.

[7] Rob F. Van der Wijngaart. Nas parallel benchmarks version 2.4. NAS Technical Report NAS-02-007, Computer Sciences Corporation, NASA Advanced Supercomputing (NAS) Division, NASA Ames Research Center, Moffett Field, CA 94035-1000, October 2002.

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

[9] Piotr Luszczek, Jack Dongarra, and Jeremy Kepner. Design and implementation of the HPCC benchmark suite. *CT Watch Quarterly*, 2(4A):18–23, November 2006.

[10] Piotr Luszczek and Jack Dongarra. Analysis of various scalar, vector, and parallel implementations of RandomAccess. Technical Report Technical Report, ICL-UT-10-03, Innovative Computing Laboratory (ICL), June 2010.

[11] ORNL Leadership Computing Facility. Introducing Titan — the world's #1 open science supercomputer, 2013. Cited 2013 May 29, 2013. Available from: http://www.olcf.ornl.gov/titan.

[12] Wayne Joubert, Douglas Kothe, and Hai Ah Nam. Preparing for exascale: ORNL leadership computing facility application requirements and strategy. Technical Report ORNL/TM-2009/308, Oak Ridge National Laboratory, December 2009.

[13] ORNL Leadership Computing Facility. Annual report 2012-2013, December 2013. https://www.olcf.ornl.gov/wp-content/uploads/2014/03/2013_ARv2M.pdf (accessed 10 August 2015).

[14] R. M. M. Mattheij, S. W. Rienstra, and J. H. M. ten Thije Boonkkamp. *Partial Differential Equations, Modeling, Analysis, Computation*. SIAM, Philadelphia, 2005.

[15] Ulrich Trottenberg, Cornelis W. Oosterlee, and Anton Schüller. *Multigrid*. Academic Press, London NW1 7BY, UK, 2001.

[16] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2nd edition, 2003.

[17] T. Hoefler, P. Gottschling, A. Lumsdaine, and W. Rehm. Optimizing a Conjugate Gradient Solver with Non-Blocking Collective Operations. *Elsevier Journal of Parallel Computing (PARCO)*, 33(9):624–633, Sep. 2007.

[18] Richard Vuduc, James Demmel, and Katherine Yelick. Oski: A library of automatically tuned sparse matrix kernels. In *Proceedings of SciDAC 2005, 26-30 June 2005, San Francisco, CA, USA, Journal of Physics: Conference Series*, volume 16, pages 521–535. IOPscience, Bristol, UK, June 2005.

[19] Jong-Ho Byun, Richard Lin, Katherine A. Yelick, and James Demmel. Autotuning sparse matrix-vector multiplication for multicore. Technical Report UCB/EECS-2012-215, Electrical Engineering and Computer Sciences University of California at Berkeley, November 28 2012. http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-215.html.

[20] Eun-Jin Im, Katherine Yelick, and Richard Vuduc. Sparsity: Optimization framework for sparse matrix kernels. *Int. J. High Perform. Comput. Appl.*, 18(1):135–158, February 2004.

[21] Xing Liu, Mikhail Smelyanskiy, Edmond Chow, and Pradeep Dubey. Efficient sparse matrix-vector multiplication on x86-based many-core processors. In *ICS'13*, Eugene, Oregon, USA, June 10-14 2013. ACM.

[22] Jack Dongarra and Victor Eijkhout. Finite-choice algorithm optimization inconjugate gradients. Technical Report 159, LAPACK Working Note (LAWN), January 2003.

[23] A. Chronopoulos and C. Gear. s-step iterative methods for symmetric linear systems. *Journal of Computational and Applied Mathematics*, 25:153–168, 1989.

[24] E. D'Azevedo, V. Eijkhout, and C. Romine. Lapack working note 56: Reducing communication costs in the conjugate gradient algorithm on distributed memory multiprocessor. Technical Report CS-93-185, Computer Science Department, University of Tennessee, Knoxville, 1993.

[25] V. Eijkhout. Lapack working note 51: Qualitative properties of the conjugate gradient and Lanczos methods in a matrix framework. Technical Report CS 92-170, Computer Science Department, University of Tennessee, 1992.

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

[27] Pieter Ghysels and W. Vanroose. Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm. Technical Report 12.2012.1, Intel Labs Europe, December 2012. Presented at PRECON13, June 19-21, 2013, Oxford, UK.

[28] Michael A. Heroux, Douglas W. Doerfler, Paul S. Crozier, James M. Willenbring, H. Carter Edwards, Alan Williams, Mahesh Rajan, Eric R. Keiter, Heidi K. Thornquist, and Robert W. Numrich. Improving performance via mini-applications. Technical Report SAND2009-5574, Sandia National Laboratories, September 2009.