HPGMG 1.0: A Benchmark for Ranking High Performance Computing Systems

Mark F. Adams[1], Jed Brown[2], John Shalf[1]
Brian Van Straalen[1], Erich Strohmaier[1], and Sam Williams[1]

[1] *Computational Research Division, Lawrence Berkeley National Laboratory*
[2] *Mathematics and Computer Science Division, Argonne National Laboratory, Department of Computer Science, University of Colorado Boulder*

Abstract

This document provides an overview of the benchmark – *HPGMG* – for ranking large scale general purpose computers for use on the Top500 list [8]. We provide a rationale for the need for a replacement for the current metric HPL, some background of the Top500 list and the challenges of developing such a metric; we discuss our design philosophy and methodology, and an overview of the specification of the benchmark. The primary documentation with maintained details on the specification can be found at hpgmg.org and the Wiki and benchmark code itself can be found in the repository https://bitbucket.org/hpgmg/hpgmg.

1 Summary

The High Performance Linpack (HPL) benchmark (and associated Top-500 List [8]) has been a successful metric for ranking high performance computing systems. HPL became a broadly accepted representative for application performance when it came to prominence in the 1990s, but over time has become less reflective of the system performance metrics that matter to contemporary science and engineering applications as lower complexity algorithms have been developed and required for extreme scale computing. We define a metric for ranking the worlds largest general purpose computers that maintains many of HPL's desirable qualities: a direct, non-iterative, solver (although only asymptotically exact) for systems of linear algebraic equations with a metric of equations solved per second (with a mapping to flops/s). We define a high performance geometric multigrid (HPGMG) benchmark that provides a more balanced exercise of machine capabilities, relative to application of interest in scientific computing, to provide a more accurate proxy for modern application requirements. HPGMG is composed of computations and data

access patterns more commonly found in contemporary applications. Using HPGMG, we aim to create a benchmark for ranking systems that will promote system design improvements that are better aligned to real scientific application performance.

2 Introduction

The High Performance Linpack () benchmark is 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.

HPL rankings of computer systems are no longer so strongly correlated to real application performance, especially for the broad set of HPC applications that analyze differential equations, which tend to have much stronger needs for high bandwidth and low latency, and tend to access data using irregular patterns. 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. Despite that, due to its long accumulated history, the Top500 list continues to offer a wealth of information about HPC industry trends and investments, but the mismatch between observed application performance and HPL ranking has become unsustainable.

We expect the gap between HPL measurements and the expectations it sets for real application performance to increase in the future. The lack of correspondence between observed application performance and the expectations set by HPL leads to skepticism by the application community and a tendency to prematurely declare success at each major increment of peak flop rate improvement when applications are do not see corresponding performance improvements. In fact, based on the example set by Tianhe-2, the first 1 Exaflop HPL will likely to be achieved via brute-force, which will result in design that is unattractive for real applications. Without intervention, such a development will undercut R&D investments to create more usable systems due to premature declaration of success on achieving the next 1000x milestone in HPL performance improvement. 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.

2.1 Why HPL has lost relevance

HPL factors and solves 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 can achieve very high computational intensity. With proper organization of the computation, data access is predominantly unit stride and is mostly hidden by concurrently performing computation on previously retrieved data. This kind of algorithm strongly favors computers with very high floating-point computation rates and adequate streaming memory systems, and was a reasonably good reflection of the dominant computational techniques of its day. Furthermore, HPL was a good proxy for a time when floating point performance and memory capacity were among the most costly aspects of the design, and represented the most desirable attribute for such systems. The ability to the benchmark to scale through many orders of magnitude improvements in hardware performance improvements, and to memory capacity has enabled it to remain a standard metric for documenting the historical improvements in the performance of top systems for three decades – outlasting any other known benchmark for assessing and ranking system performance.

However, over the lifetime of HPL benchmark and the Top500 list, the development of faster and more computationally efficient algorithms has reduced the use of these highly floating point intensive algorithms in most engineering and scientific communities. *Many of these algorithms are multilevel methods (e.g., mesh partitioners, multi-scale methods as well as equation solvers).* These algorithms generally have lower or equal complexity in nearly every conceivable complexity measure (e.g., flops, computational depth, memory usage, memory movement at all levels of the memory hierarchy) than their predecessors. For instance multigrid has improved time to solution by reducing the total number of steps to solution (including requirements for nearly all hardware performance metrics) in addition to reducing the total number of FLOPs. In fact, the SCALES report of [6] observed that the history of performance improvements in algorithms has achieved 9 orders of magnitude improvement in delivered application performance during the same period that hardware improvements delivered a mere 3 orders of magnitude improvement. Today, algorithms with arithmetic intensity (AI) of 2-4 flops/byte, and lower, are common. While research and development in algorithm and techniques to increase AI, with the aim to reduce the data movement, HP benchmark should reflect the higher memory bandwidth and lower latency demands of modern algorithms and future application.

Not only have the design of algorithms drifted away from the characteristics that dominated at the time that HPL was first formulated, so too have the fundamental engineering challenges of HPC hardware design. In the 1980's

FLOPS were among the most precious components of HPC systems and represented a first order engineering challenge to scale. Now, energy consumption has become a leading design challenge for future HPC systems. Whereas the energy cost of performing a floating point operation used to be dominant, the energy consumed by moving the operands for such an operation exceed the energy cost of the floating point operation because logic performance and cost has been improving faster than improvements in data movement since 2004. So HPL is neither reflective of current algorithm design nor is it strongly reflective of the most pressing challenges to the design of future computational hardware. It emphasizes aspects of the hardware that are comparatively easy to scale (FLOPs) and neglects the emerging challenges such as scaling up the interconnect and memory bandwidth.

2.2 Historic Challenges to Creating a New HPC Performance Metric

There have been many failed attempts to supplant HPL, partly due to its long history, but also due to its ability to keep weak-scaling through many orders of magnitude improvements in memory capacity and peak system performance. Erich Strohmaier has pointed out that we must choose *extensive* metrics such as HPL that are able to rank systems in a manner that tracks growth of capability even if the metric does not reflect the rate of growth in the metics that are of most value to the user community [7]. Attempts to supplant HPL rankings have tended to fail because most common formulations of extensive metrics tend to correlate (albeit imprecisely) to the overall size of the machine. Systems that offer more HPL FLOPs tend to be larger, and hence have larger quantities of the other extensive metrics such as memory bandwidth, bisection bandwidth, and other key metrics that might be exercised through scaling. However, the desired outcome is to make the absolute value of the rankings more closely track observed application performance benefits, and to offer algorithms challenging enough to reflect programmability concerns for real systems. This prevents premature declaration of success with each major performance improvement milestone, which undercut the support from applications (that are not realizing similar improvements) and undercut research funding to address the much more difficult hardware and software improvements.

3 Design Principles for a New HPC Performance Metric

A benchmark must reflect improvements to computer systems that benefit our applications and is essential for documenting future improvements to HPC systems. The metric should must be designed so that, as we optimize metric

results for a particular platform, the changes will also lead to performance improvements realized by real applications. Any new metric we introduce must satisfy a number of requirements:

- Accurately reflect the characteristics of contemporary high-efficiency algorithms.
- Accurately reflect the principle challenges of future hardware design – a balanced combination of memory bandwidth, interconnect performance (both for small and large messages), computational performance, and reliability. It should not be possible to "cheat" the benchmark by over-provisioning the hardware in any one of these areas. A machine designed for the sole purpose of performance on out metric should result in a "pretty good" machine for scientific and engineering applications.
- The absolute improvements in this benchmark should ultimately be reflective of performance improvements realizable in real applications, which are occurring at a much slower rate than improvements in peak FLOPs.
- It must be able to scale through many orders of magnitude improvement of hardware storage capacity, and performance – much as HPL has for the past three decades.

No one benchmark can provide an accurate proxy of any particular application but we believe that one comprehensive benchmark has two advantages over the alternative: a weighted set or bag of (simple) benchmarks. One comprehensive benchmark, like HPL, has the advantage that it is much easier to administer, define and adjudicate. Deciding on weights would be a contentious process (although we have desire parameters that are somewhat similar), and the current administrators of the top500 list believe that the one monolithic "solve" approach is highly desirable. Note, our choice of algebraic equation solvers is somewhat arbitrary as other applications could plausible provide a framework for a benchmark. The wighted bag of benchmarks and that parameters in our benchmark requires *explicitly* defining machine metrics, measuring these metrics in the benchmark and applications, and fitting internal parameters so as to provide the best proxy for the applications of interest. Posing a fundamentally hard problem (i.e., a problem with high mathematical complexity) to solve *implicitly* demands an effective machine. We desire a rational approach, with modeling and measurements, to benchmark design but models are not perfect measures of machine effectiveness and so a benchmark that both implicitly and explicitly demands an effective machine is desirable. The advantage of the weighted bag of benchmarks is that one has a great deal of freedom in fitting the benchmark to applications; we believe that HPGMG provides enough internal parameters (e.g., equations to solve, discretization, and various multigrid scheduling) to adequately match applications. It is incumbent on us to demonstrate this 3.3.

3.1 Modeling of Machine Metrics

Metrics to assess the effectiveness of a benchmark as a proxy for an application or a collection of applications is critical to the rational design and assessment of a benchmark. To this end we define machine metrics and methodology for collecting these metrics, collect these metrics for applications, and parameterize (§3.3) our benchmark(s) to best match "applications". We will be responsive to the community to define the application mix. One goal of benchmark design is to be flexible enough to adequately match applications in scientific commuting, however what this exact mix is requires community input. Definitions of the metrics and methodology can be found on the repository Wiki (https://bitbucket.org/hpgmg/hpgmg/wiki).

3.2 Design requirements, Goals, and Definition of the Benchmark

An HPC benchmark is a complex socio, economic, political, technical, and mathematical endeavor. The many failures to supplant HPL attest to this. We aim to emulate what we see as some of the important aspect of HPL. In addition to the HPL's strengths that we have already mentioned we see additional technical characteristics of HPL that have lead to its success:

(1) A mathematically global, fully coupled, application – solving random dense linear systems.
(2) An optimal order complexity algorithm for the application – $\mathcal{O}(N^3)$ work complexity.
(3) A high quality implementation – Linpack was and is a state-of-the-art implementation of the numerics of Gauss elimination.
(4) A clear, well defined, valid metric – equations/s with a mapping to flops/s.

We propose a new benchmark to be added to the Top500 list, that solves a system of algebraic equations with one application of full geometric multigrid. This proposal address our design requirements as follows:

Application: We propose to solve a discretization of the variable coefficient Laplacian problem with an analytic solution on a Cartesian grid in 3D. The solution of the Laplacian has $\mathcal{O}(\log(N))$ computational depth or parallel complexity inherent to the problem. That is, *it is a mathematically global problem and requires a full coupled computation to solve.*

Complexity: One full multigrid (FMG) application, [4, 5, 9] §A, has optimal work complexity of $\mathcal{O}(N)$, computational depth of $\mathcal{O}(\log^2 N)$, and is a *provably asymptotically exact solver for the constant coefficient Laplacian* with a

convergence rate of the same order as the discretization. We observe and can verify this asymptotic convergence for non-constant problems within FMG within our benchmark because FMG has a Cauchy sequence (norms of the errors and residuals at each level) that can be reported to verify quadratic convergence. Additionally, we can achieve what is known as "textbook multigrid efficiency" where the constant in the complexity is about 10 residual calculations. Near textbook efficiency has been demonstrated for many applications including nonlinear, eight field, resistive magneto-hydrodynamics [3]. The computational depth is the square of the theoretical lower bound. This is apparent non-optimality is actually a feature in that it puts more pressure on the communication fabric because more work is done on coarse grids where there is less work to amortize communication latency. There are other multigrid cycles, which have different rations of global to local work, and we may use this a parameter in the benchmark. We use Chebyshev smoothers, which only require the application of the operator and basic vector operations. While, Gauss-Seidel is a popular multigrid smoother it requires a separate kernel, is complex to implement in parallel for high order operators (potentially a good feature but we have elected not to pursue it) [1], and is not highly desirable for elliptic operators.

Implementation: We have experts in multigrid, have implemented multigrid many times between us, consult and are coauthors with the worlds experts on multigrid, and are thus well qualified to conduct this work. This application is simple enough that we can deploy implementations that are reasonably close to optimal, at least in the high level message passing layer. We deploy reference implementations of the kernels, which can be optimized by users, that are highly efficient on at least some architectures.

The metric: While flops/s is the published HPL metric, this is not a valid metric. Strohmaier points out: "... the basic ideas on how to construct a generalized utility metric for such a purpose. It multiplies all desired quantities (computing capability) and divides them by undesired quantities...". HPL circumvents this problem by, crucially, *defining* the flop count (and undesirable quantity) as a function of the the number of equations solved (a desirable quantity). *Our metric is equations solved per second.* We can map equations to flops with a linear function, that depends on many design factors that we define §3.3, if this is desired by the community. Because we only have an asymptotically exact method we face the challenge of defining "solved". One of the advantages of FMG, besides being efficient and non-iterative, is that it is equipped with a Cauchy sequence (norms of errors or residuals at each level) that can be use to verify same order convergence as the discretization.

3.3 Benchmark Parameters

Given the machine metrics for measuring the computational characteristics of a code (application and benchmark) we wish our benchmark to demand a machine that close to that of our applications of interest.Geometric multigrid with matrix free operators provides several free parameters that can be adjusted to improve the fit of benchmark to applications.

The equations and discretizations are a free parameter that can be clearly defined and understood, and easily verified. Matrix free methods offer a wide range of arithmetic intensity (a proxy for flop rates), demands for both streaming and cache memory, and complex process cooperation where natural units of computation like elements or flux surfaces update multiple variables.

Multigrid also allows for the the smoother scheduling, the number of pre and post smoothing steps at each level, to be specified. Large numbers of smoothing steps uses more flops in the high intensity operator or smoother kernels and allows for more loop fusion. More smoothing steps on coarse grids and be used to tune the amount of work done on coarse grids. Multigrid cycles (e.g., V, W, and F) can be used as a parameter to also define the proportion of work on on coarse versus fine grids.

3.4 HPGMG Code Design

We provide a high level harness, or framework code, for the benchmark that runs the FMG solver and reference implantations of kernels, which contain all numerics and communication. Our harness uses MPI and given the number of MPI processes, an optional process index space, and the size of a local rectangular 3D Cartesian grid on each process, defines a 3D problem domain. We run FMG with a manufactured solution, generating a right hand side and verifying the error at each level, calling kernel methods for the numerics (e.g., multigrid prolongation and restriction, applying the operator, norms, etc.). We provide reference implementations of the kernels and users are encouraged to write custom kernels and add them to the repository for verification and public release after the top500 list is published. We discourage writing new a new harness and reserve the right to reject any submission that does not use our harness. We encourage users to work with us in providing flexibility that would motivate writing a new harness. For instance, loop fusion techniques might require that more of the FMG algorithm is abstracted. We will work with users to provide desired flexibility. If users would like to not use MPI in the harness, for instance, thus requiring a rewrite of the harness, we will work with you in certifying new harnesses to add to the benchmark

repository, again to be publicly available after its publication on the top500 list [2]. Further details of the specification can be found on the repository Wiki (https://bitbucket.org/hpgmg/hpgmg/wiki).

Acknowledgments

This work was funded by the Office of Advanced Scientific Computing Research.

This manuscript has been authored by an author at Lawrence Berkeley National Laboratory under Contract No. DE-AC02-05CH11231 with the U.S. Department of Energy. The U.S. Government retains, and the publisher, by accepting the article for publication, acknowledges, that the U.S. Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for U.S. Government purposes.

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.

Appendices

A Multigrid Methods

Multigrid methods are motivated by the observation that a low resolution discretization of an operator can capture modes or components of the error that are expensive to compute directly on a highly resolved discretization. More

generally, any poorly locally-determined solution component has the potential to be resolved with coarser representation. This process can be applied recursively with a series of coarse "grids", thereby requiring that each grid resolve only the components of the error that it can solve efficiently. These coarse grids have fewer grid points or work complexity, typically about a factor of 2-4 in each dimension, such that the total amount of work in multigrid iterations can be expressed as a geometric sum that converges to a small factor of the work on the finest mesh. This is a general process and can be applied to problems with particles/atoms or pixels as well as the traditional grid or cell variables considered here. Multigrid provides a basic framework within which particular multigrid methods can be developed for particular problems.

Geometric multigrid not only provides a powerful basis on which to build a specific solution algorithm, but also allows for the straightforward use of nonlinear multigrid, or full approximation scheme (FAS) multigrid [4] and matrix-free implementations. Figure A.1 shows the standard multigrid FAS V-cycle and uses the smoother $u \leftarrow S(A, u, b)$, the restriction operator R_k^{k+1}, which maps residuals and current solutions from the fine grid space k to the coarse grid space $k+1$ (the rows of R_k^{k+1} are the discrete representation, on the fine grid, of the coarse grid functions), and the prolongation operator P_{k+1}^k, which maps the current solution from the coarse grid to the fine grid.

function $u \leftarrow MGV(A_k, u_k, f_k)$
  if $k > 0$
    $u_k \leftarrow S(A_k, u_k, f_k)$ /* pre-smoothing */
    $w_{k-1} \leftarrow R_k^{k-1}(u_k)$ /* restriction of current solution to coarse grid */
    $p_k \leftarrow A_k u_k$
    $\tau_{k-1} \leftarrow A_{k-1} w_{k-1} - R_k^{k-1}(p_k)$ /* τ correction */
    $p_{k-1} \leftarrow MGV(A_{k-1}, w_{k-1}, f_{k-1} + \tau_{k-1})$
    $w_{k-1} \leftarrow p_{k-1} - w_{k-1}$ /* convert to an increment */
    $u_k \leftarrow u_k + P_{k-1}^k(w_{k-1})$ /* prolongate coarse grid correction */
    $u_k \leftarrow S(A_k, u_k, f_k)$ /* post-smoothing */
  else
    $u_0 \leftarrow A_0^{-1} f_0$ /* accurate solve of coarsest grid */
  return u_k

Fig. A.1. FAS multigrid *V-cycle* algorithm

Common notation for this multigrid V-cycle is V(μ_1,μ_2), where μ_1 and μ_2 are the number of pre- and post-smoothing steps, respectively. Figure A.2 shows the standard full multigrid algorithm.

The complexity of an F-cycle is asymptotically similar to a V-cycle, and it can be proven to produce a solution with algebraic error that is less than the incremental error on some problems [9], that is and *asymptotically exact solver*,

$f_0 \leftarrow F(0)$ /* compute RHS */
$u_0 \leftarrow A_0^{-1} f_0$ /* coarse grid solve */
$k \leftarrow 1$
while(!*done*)
    $u_k \leftarrow \bar{P}_k^{k-1}(u_{k-1})$ /* FMG prolongation */
    $f_k \leftarrow F(k)$ /* compute RHS */
    $u_k \leftarrow MGV(A_k, u_k, f_k)$
    $e_k \leftarrow error(u_k)$ /* compute error for convergance test */
    $k \leftarrow k + 1$

Fig. A.2. *Full* multigrid algorithm

and has been demonstrated to be asymptotically exact on many problems, such as resistive magneto-hydrodynamics [3].

References

[1] M. F. Adams. A distributed memory unstructured Gauss–Seidel algorithm for multigrid smoothers. In *ACM/IEEE Proceedings of SC2001: High Performance Networking and Computing*, Denver, Colorado, November 2001.

[2] Mark Adams, Jed Brown, John Shalf, Brian Van Straalen, Erich Strohmaier, and Sam Williams. HPGMG. https://bitbucket.org/hpgmg/hpgmg, 2014.

[3] Mark F. Adams, Ravi Samtaney, and Achi Brandt. Toward textbook multigrid efficiency for fully implicit resistive magnetohydrodynamics. *Journal of Computational Physics*, 229(18):6208 – 6219, 2010.

[4] A. Brandt. Multi-level adaptive solutions to boundary value problems. *Math. Comput.*, 31:333–390, 1977.

[5] Achi Brandt. Multigrid techniques: 1984 guide with applications for fluid dynamics. Technical Report GMD-Studien Nr. 85, Gesellschaft fur Mathematik und Dataenverarbeitung, 1984.

[6] Phillip Colella, Thom H. Dunning, William D. Gropp Jr., and David E. Keyes. A science-based case for large-scale simulation. Technical report, DOE, 2003.

[7] Erich Strohmaier. Generalized utility metrics for supercomputers. *Computer Science-Research and Development*, 23(3-4):185–193, 2009.

[8] Top500. Top 500 supercomputer sites. http://www.top500.org/, 2013.

[9] U. Trottenberg, C. W. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, London, 2001.