Benchmarking Heterogeneous HPC Systems Including Reconfigurable Fabrics: Community Aspirations for Ideal Comparisons

Peter Jamieson
Department of Electrical and Computer Engineering
Miami University
Oxford, Ohio 45056
Email: jamiespa@miamioh.edu

Ahmed Sanaullah and Martin Herbordt
Department of Electrical and Computer Engineering
Boston University
Boston, Massachusetts 02215
Email: isanaulah@bu.edu and herbordt@bu.edu

***Abstract*—We describe a progressive philosophy to help benchmark systems and designs in the High Performance Computing (HPC) domain. These systems now include heterogeneous multi-node systems where nodes can include combinations of CPUs, GPUs, FPGAs, and other emerging computing devices. Because of the heterogeneous nature of these systems, benchmarking and comparison of systems is an ever increasingly complex process. Currently, there is no benchmark or other process that results good comparisons. We introduce a set of tenets that will allow us to compare one system to another and to include tool innovations. These tenets rely on our research community providing what we call a system *context* and an *instance* of a specific design (benchmark). A better benchmarking process will make our future research stronger scientifically, and will allow us to improve future systems, their accompanying compilation/synthesis tools, and the designs that execute on these systems. To achieve this, we survey existing benchmarks and their accompanying philosophies, and we describe our conceived ideal benchmarking scenario with a set of tenets, some definitions, a methodology, and a discussion that we hope will guide our research community forward.**

I. Introduction

One of the fundamental question in computer engineering is how does an application design and the platform on which it executes compare with other designs and systems? When focusing on just the comparison of computing systems, we use a process called benchmarking where, typically, a benchmark suite is implemented on the systems under comparison and metrics generated to compare them. This approach works well when comparing similar computing architectures, especially when the benchmarks can be written in a common language and there are obvious problem sizes. But the challenge becomes significantly harder when architectures with substantial variation, e.g., CPUs vs GPUs vs DSP chips, are compared. The problem is much worse when FPGAs or custom chips are included; this is because the hardware fabric itself is open for variation of architecture coupled with design.

Currently, there are a number of HPC systems emerging that contain nodes with a heterogeneous mix of CPUs, GPUs, FPGAs, and other devices in various architectures including clusters with mixes of these nodes. In other words, the benchmarking problem continues to become more complex since these systems are accompanied by multiple compilation/synthesis tools that map designs to the target system (including various computing chips), include different communication technologies and interconnects, and include a range of other differences. How can we compare these systems?

With such a rapidly changing field and with such flexible target architectures, other questions arise. How can we show improvement a particular application design? That improvement is likely to contain innovation in algorithm, partitioning, load balancing, and many other optimizations. Is it possible to separate these optimizations from changes in technology? In many cases, the new and reference (old) implementations of a designs are implemented on very different systems. Recreating the reference implementation on the new system is likely to be extremely time-consuming, if it is possible at all.

In this paper, we provide some initial insight, discussion, and ideas on how our research community can help solve these comparison problems. To do this, we survey a wide range of existing benchmarks, including their innovative ideas and philosophies. From this survey, we derive an ideal benchmarking situation, which includes a methodology, some key definitions, and an accompanying philosophy. We then discuss where current benchmarking and comparisons are still lacking in bringing stronger scientific rigor to our field, which if solved will help us build better systems, accompanying tools, and more efficient applications in the future.

II. Benchmarking

Benchmarking is the process by which a System Under Test (SUT) executes a defined input, known as a benchmark ($Benchmark$), and outputs measurements ($Measurements$ + $Results$). This comparison can be done for a number of reasons such as comparing SUTs in terms of performance, power, efficiency, scalability, or robustness. A benchmark can include input data ($Data$) and an application design ($Design$) where the input can take a variety of forms depending on the targeted

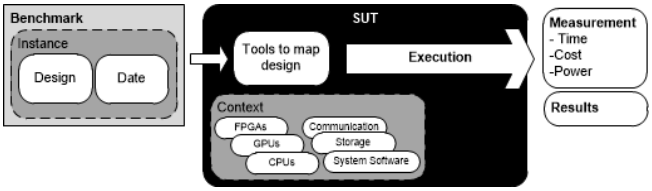

Fig. 1. Basic benchmark executed and measured on SUT

SUTs. We refer to the benchmark input as the ($Instance$) of the benchmark for future use. Figure 1 shows the typical benchmarking setup. Hennessy and Patterson [1] provide a good introduction to benchmarking computer systems.

A benchmark suite includes a collection of benchmarks with various properties to exercise the SUT in various ways. For example, to compare the execution time of System A (SUT_A) with System B (SUT_B), where the SUTs are CPUs, with a benchmark suite consisting of ($Benchmark_X$ and $Benchmark_Y$), which are programs written in the C language, then we execute the benchmarks on each system, record the execution times, and use the measurements (and statistical methods) to analyze the results. If this process is formalized in the benchmark suite, then we say that the suite has a defined methodology ($Methodology$), which includes details such as whether the $Instance$ is allowed to be modified and how results should be reported. In this example, since the benchmark suite includes executable benchmarks on both SUTs, the benchmarking process is easy. The benchmark suite only needs minor modifications to be instrumented for capturing time measurements and compilation to the respective SUT before executing.

A SUT can include various components including a variety of computation chips (CPUs, GPUs, FPGAs, etc.), a variety of meta-architectural technologies (communication, storage, input and output devices, system level software), and a variety tools to map aspects of the design to these technologies (compilers, synthesis tools). As the diversity of SUTs increase and new technologies emerge, so do the effect on application performance of tuning, algorithm, other optimizations, and varying capabilities of the computing SUT. Therefore, understanding the capabilities of a SUT can be as important to making sense of the results as understanding the benchmarks. We call this the $Context$ of the SUT. Additionally, $Context$ of a SUT captures a historical record of the capabilities of that system to help future comparisons that do not have access to older systems.

In computer systems, the measurements ($MetricOutputs$) of interest from a benchmark include, ideally, time (T) [1], energy (E), and cost (C). The metrics can be described in a variety of related terminology. For example, for FPGAs a benchmark can include the maximum clock frequency, a power profile given test input vectors, and fabric utilization. These metrics, however, are not a direct representation of the ideal metrics for time, energy, and cost, and the differences are usually due to the difficulty of measuring ideal metrics (and adopted practices).

A different use of SUTs and benchmarks is when the comparison involves both benchmark and SUT. This is common when, for example, researchers want to compare an application, such as the fast Fourier transform (FFT), which is a family of algorithms, but can be implemented on the SUT in different ways and potentially with different inputs. For example, on an FPGA we implement the FFT using some code that executes on a processor [2] but also a parallel implementation [3]. Since both of these are implemented on different FPGAs, both the benchmark $Instance$ and SUTs $Context$ differ making comparison even more difficult.

III. Ideas from Existing Benchmarks

There are a large number of benchmarks for computing; while we look at a large and representative number, this set is not exhaustive. Also there are many ways to characterize a benchmark. These are some important features as proposed by Becker, *et. al.* [4]:

- Target: the elements and types SUT that will be benchmarked and measured. This can include the measurement of different aspects of the SUT.
- Composition: the types of benchmarks in the suite. Micro-benchmarks - intended to test a small aspect of the SUT. Kernels - small common components of applications. Applications - representative of common full designs to problems. The benchmarks can be real or synthetic and may have sub-characteristics that describe details of the benchmarks. For example, in the Rodinia benchmark [5] they use Kiviat plots to show an 8-axis measure of their internal composition.
- Measurement Criteria: the type of measurement that the benchmark is meant to test.
- Workload execution: how the benchmarks executes on the SUT. This includes throughput, transaction based, intermittent, or a mixture.
- Languages: the form of the design description as part of the $Instance$.
- Data: the form of the input data as part of the $Instance$.
- Tools or Environment: specific restrictions that are imposed on the $Context$ of the SUT.
- Methodology: whether the benchmark suite specifies an exact $Methodology$ that must be followed when benchmarking a system.
- Run Report: whether the benchmark suite specifies a specific template and way to report measurements (as part of the $Methodology$).

Table I shows a subset of benchmarks. We focus on those that are more relevant to HPC and reconfigurable applications. We organize them into categories with the latest appearing at the top of a category. We omit categories if their ideas are subsumed elsewhere, e.g., HPC-GPU within other heterogeneous focused benchmarks such as HPC-Heterogeneous (CPUs, GPUs). A significant portion of this table is based on Becker, *et. al.* [41], but we have added more to the

TABLE I
THE SURVEYED BENCHMARK SUITES ORGANIZED INTO CATEGORIES

Benchmark	Category	Measurement	Languages
Valar [6]	HPC - Heterogeneous (CPUs, GPUs)	performance	OpenCL
OpenCL 13 Dwarfs [7]	HPC - Heterogeneous (CPUs, GPUs)	time	OpenCL
Parboil [8]	HPC - Heterogeneous (CPUs, GPUs)	time	C, OpenCL, CUDA
PBBS [9]	HPC - Heterogeneous (CPUs, GPUs)	time	C++ with Intel Cilk Plus
SHOC [10]	HPC - Heterogeneous (CPUs, GPUs)	time	OpenCL, CUDA, MPI
Rodiani [5]	HPC - Heterogeneous (CPUs, GPUs)	performance, power	OpenMP, OpenCL, CUDA
Spector [11]	HPC - FPGA	time	OpenCL
CHO [12]	HPC - FPGA	time	OpenCL
IMSuite [13]	HPC - Parallel CPUs	performance	X10, HJ
HPGMG [14]	HPC - Parallel CPUs	performance	MPI
HPCG [15]	HPC - Parallel CPUs	performance	OpenMP, MPI
Graph500 [16]	HPC - Parallel CPUs	time	C, OpenMP, MPI
MPAC [17]	HPC - Parallel CPUs	performance	-
HPCC [18]	HPC - Parallel CPUs	time	C, BLAS, MPI
HPL [19]	HPC - Parallel CPUs	performance, accuracy	MPI, BLAS
SPEC MPI2007 [20]	HPC - Parallel CPUs	time	MPI, C, C++, Fortran
SPEC OMP [21]	HPC - Parallel CPUs	time	OpenMP, C, Fortran
NAS Parallel [22]	HPC - Parallel CPUs	performance	MPI, OpenMP, Java, High Performance Fortran
Genesis [23]	HPC - Parallel CPUs	performance	Fortran
HINT [24]	HPC - Parallel CPUs	performance	-
SLALOM [25]	HPC - Parallel CPUs	performance	-
LAPACK [26]	HPC - Parallel CPUs	time	Fortran
Lawrence Livermore Loops [27]	HPC - Parallel CPUs	performance	-
EEMBC [28]	Processors	performance, energy	-
SPEC CPU2006 [29],[30]	General Purpose	performance	C, C++, Fortran
SPEC JVM98 [31]	General Purpose	performance	Java
Fhourstones [32]	General Purpose	performance	C, Java, Haskell
BDTi Video Kernel [33]	Semiconductor Devices	performance, cost, energy	C
IWLS05 [34]	Semiconductor Devices	speed, area	RTL HDL
ITC02 [35]	Semiconductor Devices	-	module netlist
Placement01 [36]	Semiconductor Devices	speed, area, power	RTL HDL
SCU-RTL [37]	Semiconductor Devices	speed, area, power	RTL HDL
ITC99 [38]	Semiconductor Devices	speed, area	RTL HDL
picoJava [39]	Semiconductor Devices	speed, area	RTL HDL
ERCBench [40]	Embedded Systems, Reconfigurable	performance, power	RTL HDL
GroundHog [41]	Embedded Systems, Reconfigurable	power	Algorithm, C, RTL HDL
BDTi DSP Kernel [42]	DSP	performance, cost, area, energy	C
BDTi Communications [43]	DSP, FPGAs	number of channels	C, matlab
RAW [44]	Reconfigurable	performance	C, RTL HDL

HPC domain and removed some other categories and older benchmarks.

These data illustrate the range of benchmarks that have been created for just a subset of potential computational research areas. Also, for systems including reconfigurable components, one should note the wide variety of design languages, which means SUTs could be impacted by the tools that map designs, as much as the hardware to which they are mapped. Finally, some important metrics are rarely included. Most benchmarks focus on performance with substantial emphasis also on power and energy. But engineering practice always includes cost as first order concern; this tends to be very difficult to quantify and so is rarely included in benchmarks outside chip-level research.

IV. IDEAL BENCHMARKING AND COMPARISON FOR HETEROGENEOUS HPC SYSTEMS

In this section, we describe our characterization of the ideal benchmark for targeting various architectures and application designs with a fair comparison methodology. While this ideal is not always possible, we believe that research in computation should strive towards the ideal as it will help all the community in improving, comparing, and understanding our current state-of-the-art solutions. Using ideas from the surveyed benchmarks (Table I) and drawing insights from our work and others' [45], [46], we believe the tenets of ideal benchmarking are as follows:

1) Openness - Benchmark $Instance$ should be released open source so that the results can be replicated and analyzed.
2) Fairness - SUT comparisons should be fair in terms of reporting optimizations to the benchmark $Instance$ and avoiding cheating in terms of SUT exploitation of benchmarks for benchmarking sake [47].
3) Complexity - The benchmark represents the computing challenges of interest and are large enough to exercise the capabilities of the SUTs. This needs to address coverage, which needs to be community defined as the view on Parallel Dwarfs has started [48].
4) Explicit Methodology - The details should be explicit on how the benchmark is to be measured and what needs to be reported. This should include what aspects of the SUT should be included in the measurements. For example,

can initialization of the problem be ignored or is it part of the benchmark, and therefore, part of an execution measurement?

5) Meaningful Measurement Metrics - The measurements should reflect the physical and real world as much as possible.
6) SUTs $Context$ - The details about the system are released, such that the SUT can be analyzed in terms of its capabilities. Additionally, this should describe what aspects of the SUT are measured. For example, in an energy measurement, does energy measured include the fans used for cooling the SUT?

Other researchers in benchmarking have suggested other important characteristics. For example, in the survey of HPC benchmarks for the Army by Infantolino*, et. al.* [49], they want benchmarks that have sufficient coverage of behaviors, industry acceptance, and applicability to different architectures. The Parboil [8], PBBS [9], and Rodinia [5] benchmarks attempt to solve a subset of this challenge with GPUs by supplying both CUDA (NVidia) and OpenCL (Intel) GPU source files. This is a valid approach for GPUs where there is a small set of target languages and where the SUTs have a well-understood core set of characteristics. The IMSuite [13], HPCG [15], Graph500 [16], HPCC [18], and NAS Parallel [22] benchmarks use multiple parallel programming languages in a similar fashion. However, the number of SUTs and the variety of design languages used to map designs to them, means that multi-language benchmarking is of less use.

A. High-level benchmarks

As heterogeneous HPC systems continue to emerge, including large clusters of multichip cores (of different chip types) the possible range of systems to benchmark seems intractable. The GroundHog [50] approach to high-level descriptions for benchmarks, where researchers release their own versions of their specific designs, seems to be the only way forward in an open target SUT world. GroundHog focuses on mobile applications with chips such as FPGAs, microprocessors, and DSP processors, and provides high-level algorithmic descriptions and example implementations; they also provide a means to stimulate and test the SUT. This approach, however, results in significant work for researchers, but as the community adopts this approach a codebase of solutions emerges that helps the next generation in implementation. In the end, high-level benchmarks $Instance$ for the variety of architectures requires implementation of algorithmic descriptions.

In parallel with unique application implementations for high-level benchmarks (benchmark $Instance$), the SUT $Context$ should also be released openly to show the nature of the SUTs respective capabilities, and this $Context$ captures a historical snapshot of the capability of the SUT. Earlier, we described the SUT $Context$ as the details about the architecture. In an ideal benchmark, we take this a step further by requiring researchers to release two additional sets of results for applicable micro-benchmarks and design patterns.

B. Micro-benchmarks for SUT $Context$

Micro-benchmarks are simple designs that show measurements that exercise various pieces of the SUT (SHOC [10] calls these "level 0" benchmarks). For example, a micro-benchmark for multiplying 32-bit integers and fixed-point gives us a base understanding of an FPGAs speed, power, and area costs (resource usage) of the reconfigurable architecture. Similarly, communication costs for transferring data to and from the host (for example, the host PC) and between nodes (for example, within a cluster of FPGAs) on various communication protocols, accounting for vendor or benchmark created IP, allows an understanding of bottlenecks on the target platform. Micro-benchmarking for SUT $Context$, therefore, would include a large list of tests, and researchers would supply measurements from a subset of this list that apply to the high-level benchmarks used in their comparisons.

C. Parallel Design Patterns for more SUT $Context$

Design patterns [51], and more importantly, parallel design patterns [52] are benchmarks that are not applications, but rather are common, reoccurring elements of a design. In SHOC [10] benchmark these are called "level 1" benchmarks; we might also compare patterns to kernels (part of benchmarks such as Valar [6] call these "algorithmic patterns"). The list of design patterns will need to be created by the community and, similar to the micro-benchmarks, researchers will release measurements on those design patterns relevant to the $Context$ of the SUT and the high-level benchmarks used in the comparison.

A side-benefit of the identification of parallel design patterns is that it will help compiler and synthesis tool designers since they can test their tools on these patterns to verify and communicate how their tools identify and map these design structures efficiently. This may also lead to different $Designs$ for different SUTs, which, in turn, will help researchers understand how to structure their implementations to be most efficiently implemented for a particular SUT.

D. Using the benchmark $Instance$ to allow comparison

The SUT $Context$ describes the system architecture and includes measurements of both micro-benchmarks and parallel design patterns that relate to the high-level benchmarks that will be used to compare the systems. Still, if we want to compare $Benchmark_X$ on SUT_A and SUT_B where the instances ($Instance_A$ and $Instance_B$) are different, then how can we do this? This scenario typically happens when researchers build their own version for a SUT and want to compare with earlier versions by other researchers on a SUT they don't have access to. Also, open-source release of benchmarks is rare (for certain classes of SUTs) and past implementations tend to disappear into the ether.

Benchmarks that supply specific code written in a common language and compiled for SUTs avoid this problem since the $Instance$ in each case is consistent. Therefore a simple comparison of measurements shows the differences in the SUTs. In a more ambitious research agenda, we are interested

in our systems scaling and in employing all optimizations to algorithmic design, improvement in the quality of tools for the SUT, and improvements in the SUT. This means the $Instance$, as described above, is different and the SUTs are different.

To allow for ideal comparison in this case we need to capture each $Instance$ numerically as $|Instance|$, which we will call the $quanta$ of the $Instance$. For example, an m-by-n matrix multiplied by an n-by-o matrix would have a $quanta$ of $m * n * o$. This assumes that we are comparing two different SUTs with different $Instance$, where the type of multiplication and bit-width used are the same. If they are different, then these qualities need to be captured in the $quanta$ calculation. For example, $quanta$ might equal $m*n*o*mult-bit-width$ for differing multiplication sizes. Using $quanta$ and ideal measurements of time T, energy E, and cost C can be divided by $quanta$ to give a per quantum value. For example, T (in seconds) divided by $quanta$ results in a *seconds per quantum* value.

With per-quantum values, we can compare the SUTs based on these values to show the quality of the systems independent of design details, tools, and the technologies and architecture of the system. These values, together with open designs and the $Context$ of the system, allow for detailed comparisons allowing us to better show improvements.

One note, however, is the importance of agreement on how to calculate the $quanta$ of a benchmark $Instance$. Researchers need to consider this and document their process so other researchers can fairly compare results. The $quanta$ calculation is not trivial either. Consider the challenges in simulation where there can be objects (varied in number and type), space for the objects (dimensions and scales), a time-scale, and time-steps (or event-driven) of simulation; the $quanta$ calculations must be thought out carefully.

V. Discussion of Ideal Benchmarking for Heterogeneous HPC

Ideal benchmarking for HPC needs to be a community pursuit. As our brief survey of benchmarks (table I) shows, there are many benchmarks in a number of domains, and a constant effort is being made to create new benchmarks. Still, a number of researchers implement their own applications (mostly without releasing their design details) to try and demonstrate how their ideas improve the state of our technologies. In this discussion, we describe some of the challenges we face, and we attempt to make a case why a community effort is needed.

For the ideal benchmarking methodology we propose above, two particular challenges are creating appropriately complex designs and providing the micro and parallel design pattern benchmarks. Our computation technologies are continuously improving, and with this comes an ever-increasing set of capabilities; in particular, our technologies can handle larger and larger problems. Benchmark suites tend to be updated because they just don't measure the capabilities of what emerging computing machines can handle. This challenge, however, is easier solved by using high-level benchmarks that are descriptions so that researchers need to implement the $Instance$. Similarly, as the capabilities of SUTs improves, the set of micro and parallel design pattern benchmarks will need to be updated constantly. In other words, a fixed benchmark can only capture a single point in time for a SUT of that time, and adopting a community based *living* benchmark will address these challenges.

In relation to the two above challenges, we cannot underemphasize the importance of Tenet 1 of ideal benchmarking – openness. This practice of releasing our applications should be the *modus operandi* for all of our research. This allows the science and engineering to be independently verified. These open practices are emerging in granting agencies such as the National Science Foundation where the data dissemination plan is part of the granting process. Open Science (maybe we need Open Engineering?) is a movement that should be adopted in benchmarking as well as our research [53].

Finally, one of the hardest challenges for benchmarking SUTs is finding the appropriate metric. An interesting analogy is that of horse power and cars. An engine can be measured in term of equivalent horses (horsepower), but in most cases what really matters is how quickly a car can get from point A to point B, and how much energy this costs. Similarly, in computing, max frequencies, floating point operations per second (FLOPs), and Million instructions per second (MIPs) are all interesting metrics more useful for analyzing the SUT, but a measure of time taken to execute a task, is ideal. We understand measurement is difficult, but per quantum measurements as described above will provide meaningful values that can be easily compared.

VI. Conclusion and Future Work

This report looks at existing benchmarks and provides a set of tenets and ideas that should be adopted in a community effort to push our collective research forward. We describe details of how we believe ideal benchmarking and comparison should be done, but the reality is that this work requires a community effort. The question is who will do this and why would anyone do this? One possibility is to create a community benchmarking space where contributors to these efforts would be included as authors on a year-to-year paper called "State of the HPC Benchmark", which would be published in a leading conference or journal publication. Regardless, the sooner we make these efforts, the sooner our field will benefit.

References

[1] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[2] P. Wang, J. McAllister, and Y. Wu, "Soft-core stream processing on fpga: An fft case study," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 2756–2760.

[3] J. Palmer and B. Nelson, "A parallel fft architecture for fpgas," in *International Conference on Field Programmable Logic and Applications*. Springer, 2004, pp. 948–953.

[4] T. Becker, P. Jamieson, W. Luk, P. Cheung, and T. Rissa, "Towards Benchmarking Energy Efficiency of Reconfigurable Architectures," in *Field-Programmable Logic and Applications*, 2008, pp. 691–694.

[5] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*. Ieee, 2009, pp. 44–54.

[6] P. Mistry, Y. Ukidave, D. Schaa, and D. Kaeli, "Valar: a benchmark suite to study the dynamic behavior of heterogeneous systems," in *Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units*. ACM, 2013, pp. 54–65.

[7] W.-c. Feng, H. Lin, T. Scogland, and J. Zhang, "Opencl and the 13 dwarfs: a work in progress," in *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*. ACM, 2012, pp. 291–294.

[8] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W.-m. W. Hwu, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," *Center for Reliable and High-Performance Computing*, vol. 127, 2012.

[9] J. Shun, G. E. Blelloch, J. T. Fineman, P. B. Gibbons, A. Kyrola, H. V. Simhadri, and K. Tangwongsan, "Brief announcement: the problem based benchmark suite," in *Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures*. ACM, 2012, pp. 68–70.

[10] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "The scalable heterogeneous computing (shoc) benchmark suite," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*. ACM, 2010, pp. 63–74.

[11] Q. Gautier, A. Althoff, P. Meng, and R. Kastner, "Spector: An opencl fpga benchmark suite," in *Field-Programmable Technology (FPT), 2016 International Conference on*. IEEE, 2016, pp. 141–148.

[12] G. Ndu, J. Navaridas, and M. Luján, "Cho: towards a benchmark suite for opencl fpga accelerators," in *Proceedings of the 3rd International Workshop on OpenCL*. ACM, 2015, p. 10.

[13] S. Gupta and V. K. Nandivada, "Imsuite: A benchmark suite for simulating distributed algorithms," *Journal of Parallel and Distributed Computing*, vol. 75, pp. 1–19, 2015.

[14] M. Adams, "Hpgmg 1.0: a benchmark for ranking high performance computing systems," 2014.

[15] V. Marjanović, J. Gracia, and C. W. Glass, "Performance modeling of the hpcg benchmark," in *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*. Springer, 2014, pp. 172–192.

[16] R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang, "Introducing the graph 500," 2010.

[17] M. H. Jamal, G. Mustafa, A. Waheed, and W. Mahmood, "An extensible infrastructure for benchmarking multi-core processors based systems," in *Performance Evaluation of Computer & Telecommunication Systems, 2009. SPECTS 2009. International Symposium on*, vol. 41. IEEE, 2009, pp. 13–20.

[18] P. R. Luszczek, D. H. Bailey, J. J. Dongarra, J. Kepner, R. F. Lucas, R. Rabenseifner, and D. Takahashi, "The hpc challenge (hpcc) benchmark suite," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. ACM, 2006, p. 213.

[19] A. Petitet, "Hpl-a portable implementation of the high-performance linpack benchmark for distributed-memory computers," *http://www. netlib. org/benchmark/hpl/*.

[20] "SPEC MPI2007. http://www.spec.org/mpi2007/," 2007.

[21] "SPEC OMP2001. http://www.spec.org/omp/," 2007.

[22] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga, "THE NAS PARALLEL BENCHMARKS," 1994.

[23] I. Glendinning, "The genesis distributed-memory benchmark suite release, 3.0. technical report, high performance computing centre, university of southampton, 1994." 1994.

[24] J. Gustafson and Q. Snell, "HINT: A New Way to Measure Computer Performance," in *Proceedings of the Twenty-Eight Hawaii International Conference on System Sciences*, 1995.

[25] J. Gustafson, D. Rover, S. Elbert, and M. Carter, "The design of a scalable, fixed-time computer benchmark," *J. Parallel Distrib. Comput.*, vol. 12, no. 4, pp. 388–401, 1991.

[26] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed., Philadelphia, PA, 1999.

[27] F. McMahon, "The livermore fortran kernels: A computer test of the numerical performance range," Lawrence Livermore National Laboratory, Livermore, CA, Tech. Rep. UCRL-53745, December 1986.

[28] " Embedded Microprocessor Benchmark Consortium (EEMBC). http://www.eembc.org.."

[29] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 118–121, September 2007.

[30] "SPEC CPU2006. http://www.spec.org/cpu2006/," 2007.

[31] "SPEC JVM98. http://www.spec.org/jvm98/," 2007.

[32] "The Fhourstones Benchmark (version 3.1). http://homepages.cwi.nl/~tromp/c4/fhour.html."

[33] "BDTI Video Kernel Benchmarks. http://www.bdti.com/products/services_video_benchmark.html," 2008.

[34] " IWLS 2005 Benchmarks. http://iwls.org/iwls2005/benchmarks.html."

[35] "ITC'02 SoC Test Benchmarks. http://www.hitech-projects.com/itc02socbenchm/," 2002.

[36] "http://www.cs.nthu.edu.tw/$\sim$ylin/," 2001.

[37] "http://www.engr.scu.edu/mourad/benchmark/RTL-Bench.html," 1998.

[38] "ITC99 Benchmark. http://www.cerc.utexas.edu/itc99-benchmarks/bench.html," 1999.

[39] S. M. Inc., "Pico-Java Processor Design Documentation," 1999.

[40] D. W. Chang, C. D. Jenkins, P. C. Garcia, S. Z. Gilani, P. Aguilera, A. Nagarajan, M. J. Anderson, M. A. Kenny, S. M. Bauer, M. J. Schulte *et al.*, "Ercbench: An open-source benchmark suite for embedded and reconfigurable computing," in *Field Programmable Logic and Applications (FPL), 2010 International Conference on*. IEEE, 2010, pp. 408–413.

[41] P. Jamieson, T. Becker, W. Luk, P. Cheung, and T. Rissa, "Benchmarking Reconfigurable Architectures in the Mobile Domain," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 2009.

[42] "The BDTI DSP Kernel Benchmarks. http://www.bdti.com/products/services_bdti_benchmarks.html," 2008.

[43] "BDTI Communications Benchmark. http://www.bdti.com/products/services_comm_benchmark.html," 2008.

[44] J. Babb, M. Frank, E. Waingold, R. Barua, M. Taylor, J. Kim, S. Devabhaktuni, and A. Agarwal, "The RAW Benchmark Suite: Computation Structures for General Purpose Computing," in *IEEE Symposium on FPGAs for Custom Computing Machines*, 1997, pp. 134–143.

[45] K. Pereira, P. Athanas, H. Lin, and W. Feng, "Spectral method characterization on fpga and gpu accelerators," in *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*. IEEE, 2011, pp. 487–492.

[46] L. Yang, S. C. Chiu, W.-K. Liao, and M. A. Thomas, "High performance data clustering: a comparative analysis of performance for gpu, rasc, mpi, and openmp implementations," *The Journal of supercomputing*, vol. 70, no. 1, pp. 284–300, 2014.

[47] "11 myths about embedded benchmarking - is it lies, damn lies, and benchmarks? debunking 11 myths that plague embedded benchmarking," http://www.electronicdesign.com/embedded/11-myths-about-embedded-benchmarking, accessed: 2018-03-08.

[48] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. D. Kubiatowicz, E. A. Lee, N. Morgan, G. Necula, D. A. Patterson *et al.*, "The parallel computing laboratory at uc berkeley: A research agenda based on the berkeley view," 2008.

[49] J. Infantolino, S. Park, and D. Shires, "Selecting a benchmark suite to profile high-performance computing (hpc) machines," ARMY RESEARCH LAB ABERDEEN PROVING GROUND MD, Tech. Rep., 2014.

[50] P. Jamieson, T. Becker, P. Y. K. Cheung, W. Luk, T. Rissa, and T. Pitkänen, "Benchmarking and evaluating reconfigurable architectures targeting the mobile domain," *ACM Transactions on Design Automation of Electronic Systems*, vol. 15, no. 2, pp. 1–24, 2010. [Online]. Available: http://doi.acm.org/10.1145/1698759.1698764

[51] P. Wolfgang, "Design patterns for object-oriented software development," *Reading Mass*, p. 15, 1994.

[52] T. G. Mattson, B. Sanders, and B. Massingill, *Patterns for parallel programming*. Pearson Education, 2004.

[53] M. Nielsen, *Reinventing discovery: the new era of networked science*. Princeton University Press, 2011.