Considerations When Evaluating Microprocessor Platforms

Michael Anderson, Bryan Catanzaro[*], Jike Chong, Ekaterina Gonina, Kurt Keutzer,
Chao-Yue Lai, Mark Murphy, David Sheffield, Bor-Yiing Su, Narayanan Sundaram
Electrical Engineering and Computer Sciences
University of California, Berkeley

Abstract

Motivated by recent papers comparing CPU and GPU performance, this paper explores the questions: Why do we compare microprocessors and by what means should we compare them? We distinguish two distinct perspectives from which to make comparisons: application developers and computer architecture researchers. We survey the distinct concerns of these groups, identifying essential information each group expects when interpreting comparisons. We believe the needs of both groups should be addressed separately, as the goals of application developers are quite different from those of computer architects.

Reproducibility of results is widely acknowledged as the foundation of scientific investigation. Accordingly, it is imperative that platform comparisons supply enough detail for others to reproduce and contextualize results. As parallel processing continues to increase in importance, and parallel microprocessor architectures continue to proliferate, the importance of conducting and publishing reproducible microprocessor platform comparisons will also increase. We seek to add our voice to the discussion about how these comparisons should be conducted.

1 Introduction

Several recent papers, including [Lee et al., 2010] and [Vuduc et al., 2010] examine widespread claims of 10 to 100-fold performance improvements for applications running on Graphics Processing Units (GPUs) compared to applications running on traditional Central Processing Units (CPUs). These papers note that when comparing well optimized CPU and GPU implementations of throughput oriented workloads, for which GPUs are designed, GPUs hold a much more modest performance advantage, on the order of 2.5×. The 10-100× performance improvements quoted for GPU processing therefore arise from comparing against sequential, poorly optimized CPU code. Accordingly, [Lee et al., 2010] suggests that future studies comparing CPUs and GPUs should compare against thread and SIMD-parallelized CPU code in order to avoid misleading comparisons.

We agree that GPU-CPU comparisons, and particularly speedup numbers, are often taken out of context and used inappropriately. We also agree that many published comparisons are not sufficiently clear about the meaning of the comparisons they provide, which is incompatible with basic scientific methodology. The issues raised by recent papers provide the community with an opportunity to revisit the goals of and methodologies for performing comparisons.

We wish to add our voice to this discussion. We begin by examining why comparisons between processor architectures are complicated by the rise of non-traditional parallel architectures. We then investigate the motivations behind conducting such comparisons. We find two distinct points of view which hold particular importance. The first is the viewpoint of the application developer, who is focused on advancing the state of the art in a particular application domain, given implementation constraints. The second is the viewpoint of the computer architect, who is primarily concerned with assessing the strengths and weaknesses of various architectural features. Since researchers have different objectives, it is natural that their perspectives and methodologies should differ. It is not feasible for every cross-platform comparison to conduct experiments to satisfy all audiences. However, it is feasible, and should be expected, that every comparison should include or reference information which allows the comparison to be reproduced and contextualized.

2 Revisiting Comparisons

Cross-platform comparisons depend critically on a host of details. Everything from the data structures used in a benchmark to specifics of silicon implementation can strongly affect the results of a comparison. This has always been the case, but we now have an additional hurdle to deal with: the complexities of parallel programming.

Concerns about algorithms and datasets used in comparisons have historically led to the creation of well defined benchmark suites. For parallel CPU benchmarking, examples include as PARSEC [Bienia et al., 2008], SPEComp [Aslot et al., 2001], and SPLASH-2 [Singh et al., 1995]. These benchmark

[*]Correspondence should be addressed to bcatanzaro@acm.org

suites have very well defined inputs, outputs, and implementations, which makes it easier to normalize the software side of a cross-platform comparison.

However, today we are faced with a diverse array of hardware platforms which we may wish to compare, such as the Cell Broadband Engine [Chen et al., 2007], NVIDIA GPUs [Lindholm et al., 2008] and AMD GPUs [Owens et al., 2008], Intel's Many Integrated Core Architecture [Seiler et al., 2008], as well as multi-core CPUs from a host of vendors. This diversity in parallel architectures, especially due to SIMD and memory subsystem configuration, is exposed in low-level programming models. Accordingly, different hardware platforms require different source code to efficiently perform the same computation. This makes it much harder to define benchmark suites which can be efficiently targeted across this diverse array of hardware architectures. Recent efforts such as OpenCL [Khronos Group, 2010] are a step towards source code portability. However, the widely divergent characteristics of various parallel architectures require architectural consideration *even at the algorithmic level*, if efficiency is a primary concern. And efficiency is always a strong concern when evaluating parallel architectures: without the drive for increased performance, the software complexity required for utilizing parallel architectures is an unjustified expense.

Since efficient parallelized source code is not generally portable, it is difficult to form a benchmark suite comprised of a body of code which can be reused across parallel architectures. Consequently, it is likely that future benchmark suites will include functional specifications for the computations being tested, but allow for a wide variety of implementations. This further complicates reader comprehension of cross-platform comparisons, since optimized reference implementations may not be widely available.

In addition to the complexities of performing cross-platform comparisons, it is also important to differentiate between the concerns of application developers and those of architecture researchers. We examine these concerns in the following sections.

3 Concerns of Application Developers

Application developers work to advance the state of the art in their application domain, under a set of implementation constraints. These cost constraints may include developer time and skill, the freedom to rethink algorithms (or not), deployment costs - including hardware budgets in both dollars, joules and watts, as well as the need to interoperate with large legacy codebases, to name a few. Accordingly, when application developers are presented with new technologies, such as SIMD instruction set expansions, on-chip multiprocessing, or programmable GPUs, they evaluate these technologies for their potential to provide qualitative advancements in a particular domain, given their particular set of implementation constraints. For example, medical imaging researchers strive to improve the kind of imaging which can be practically performed in a clinical setting. Increased computing capabilities make more advanced algorithms for image reconstruction feasible, which are interesting to medical imaging researchers because they provide new capabilities, such as quicker image acquisition time or greater resolution.

To application developers, platform characteristics, such as performance, price, power consumption, and so forth, are useful in a qualitative sense, to answer the question: Does a particular computing platform enable an advance in the application domain, given my implementation constraints? Consequently, when application researchers publish comparisons between technologies, they naturally focus on application capabilities, and do not normally carry out architectural comparisons with careful performance characterizations. Application developers seeking to demonstrate new application capabilities naturally focus their energy on computing platforms that best showcase their application, and compare against the established code bases which form the state of the art in their domain.

As we noted earlier, there are many potential computing platforms beyond the Intel CPUs and NVIDIA GPUs considered in [Lee et al., 2010], ranging from AMD CPUs and GPUs, to Sun Niagara 2, IBM Power 7, Cell and BlueGene, as well as FPGA implementations, among others. Although each computing platform has unique performance characteristics, fully examining the implementation space to make architectural comparisons is outside the scope of application developers' concerns. There are too many potential choices to consider. In particular, documenting precisely how far behind one architecture lags behind another is of little value to application developers, since such experiments generally do not improve the state of the art in their domain of expertise.

Consequently, when application developers report $10\times$-$100\times$ speed up numbers from using a GPU, these speedup numbers should not be interpreted as architectural comparisons claiming that GPUs are $100\times$ faster than CPUs. Instead, they illustrate a return on investment: starting from a legacy, usually sequential, code base that performed at some level x, the application developer was able to take the application to some new level y, with an overall gain of y/x. The reported speedup number merely quantifies the improvements yielded by this effort; it says nothing about the architectural merits of the platforms used to implement the computations being compared.

If application developers had no implementation constraints, it would be feasible to expect them to make detailed comparisons in order to answer the question: Which architecture is optimal for my problem, and by how much? However, as we have explained, these comparisons provide the application developer with little value, because the information discovered in such an exercise is only incidental to advancing the state of the art in an application domain. Stated differently, to application developers, 10-100$\times$ speedups from using GPUs are not mythical, in fact they are commonplace and realistic: they come from porting well-established and commonly used sequential applications to GPUs, often after a clean-sheet redesign of the computations in the application and significant algorithmic reworking. The fact that application developers could have also achieved large performance improvements by rewriting their legacy code to target parallel CPU implementations is an orthogonal issue - one that justifiably falls outside the scope of their concerns. Although it is unrealistic to expect application developers to do the implementation work necessary for complete architectural comparisons, we can and should expect them to contextualize their results: *such 10-100$\times$ speedups should never be claimed as architectural comparisons between CPUs and GPUs*, but only as an advancement over the previous implementation to which comparison is being made. If the previous implementation was important to an application domain, this 10-100$\times$ speedup will have a significant impact on those who use it, and so the speedup provides value to application developers, particularly if it can be shown to improve application accuracy or capabilities.

4 Concerns of Architecture Researchers

Computer architects face a different set of problems and concerns than application developers. While application developers can focus on an application, even changing an algorithm to fit a particular processor architecture, most computer architects must make engineering tradeoffs to optimize their designs for a wide variety of applications which may run on their devices, and consider the workloads they use to test their architectures as fixed. For example, architects working on x86 microarchitectures focus on extracting performance from the x86 ISA, scaling across a broad variety of workloads and design constraints from Mobile Internet Devices all the way to the datacenter. Instead of focusing on a particular application domain, computer architects must design architectures which perform efficiently across a variety of application domains.

Consequently, computer architects often evaluate architectures based on benchmark suites which sample a broad range of application domains, where each benchmark is simple enough to be well characterized and optimized for the architectures being compared. However, conducting these comparisons is challenging, since it is difficult to normalize away implementation artifacts. For example, when comparing implementations of two different architectures, it is difficult to control for the fact that the implementations may have been carried out on different semiconductor processes, with different design budgets, targeting different customer price points, dissipating different amounts of power, etc. Although it is difficult to normalize these factors away, it is important for comparisons to present salient information about the implementations being performed, so that the reader may contextualize the results of the comparison.

More specifically, when computer architects make comparisons between architectural features, there are a number of elements which must be carefully considered. The first is the specifics of silicon implementation, which includes the style of logic implementation, such as standard-cell versus full or custom design styles. These details may lead to an order of magnitude performance difference, irrespective of the inherent value of a microarchitectural feature [Chinnery and Keutzer, 2002]. The silicon process used for implementation and its particular characteristics, as well as die size and power consumption figures may also materially impact the comparison being made.

Comparisons made for computer architects should include information about these details, so that the audience can contextualize the results. It would have been useful, for example, if [Lee et al., 2010] had devoted some space to these details in their comparison - as it stands, the paper does not mention that it compares a 65nm GPU released to market in June 2008 with a 45nm CPU released in October 2009, a 16 month gap which represents almost a full generation of Moore's law. Other important details such as die size and power consumption are also omitted, which makes it difficult for architecture researchers to understand the comparison being presented.

Additionally, in order to reproduce results from a comparison, the benchmarks and datasets used to create the comparison must be clearly defined. For example, consider Sparse Matrix Vector Multiplication, a well studied kernel on both CPUs and GPUs, e.g. [Williams et al., 2009], [Bell and Garland, 2009], which serves as one of the 14 benchmarks used in [Lee et al., 2010] to compare a CPU and a GPU.

It is well known that Sparse Matrix Vector Multiplication performance varies widely when multiplying a given matrix, depending on the data structure used to represent the matrix. For example, [Bell and Garland, 2009] found a 35-fold performance improvement when using a DIA representation compared to the Compressed Sparse Row format (CSR), on single-precision three-point Laplacian matrices, when running both computations on the same GPU. Even among CSR formats, there are multiple implementations, each of which has certain advantages for particular datasets. Concretely, [Bell and Garland, 2009] found that a vector CSR implementation was faster on the whole than a scalar CSR representation - in the extreme up to 18 times faster, while running on the same hardware. However, [Bell and Garland, 2009] also found that there were certain matrices where the scalar CSR implementation was 6 times faster than the vector CSR implementation, again running on the same hardware. The choice of Sparse Matrix representation and Sparse Matrix Vector Multiplication implementation, and Sparse Matrix dataset used for benchmarking critically impacts observed performance.

In fact, given an absolute performance target, it is possible to construct a dataset which achieves that target performance, as long as the desired target is feasible with respect to the extremes in Sparse Matrix Vector multiplication performance which have been observed on a target platform. Since the extremes are widely separated, like the 35-fold range observed in [Bell and Garland, 2009], absolute performance numbers are not interpretable without knowing more about the datasets and implementations used to generate such numbers.

Unfortunately, [Lee et al., 2010] does not detail the datasets or implementations used to perform their comparison. Besides Sparse Matrix Vector Multiplication, other benchmarks used for comparison, such as ray-casting, collision detection, and solvers for collision resolution, also exhibit strong workload- and implementation-dependent performance characteristics. Without access to details of the benchmark suite which underlies a comparison, readers are not able to interpret, contextualize or reproduce performance results, since so much of the observed level of performance depends on those details. We encourage those conducting cross-platform comparisons to make details of their work available.

5 Example Comparisons for Application Research

To give some positive examples of cross-platform comparisons for application researchers, we cite two examples.

[You et al., 2009] discussed algorithmic issues which arise in parallelization of Automatic Speech Recognition (ASR). ASR takes audio waveforms of human speech and infers the most likely word sequence intended by the speaker. Implementing ASR on parallel platforms presents two challenges to the application developer: efficient SIMD utilization and efficient core level synchronization. The authors explored four different algorithmic variations of the computation on two parallel platforms - Intel Core i7 CPU and NVIDIA GTX280 GPU.

The sequential baseline was implemented on a single core of a Core i7 quad-core processor. The authors noted that the platform characteristics of the CPU yielded different algorithmic tradeoffs than the GPU, meaning that the most efficient algorithm for parallel ASR was different for the two platforms. Overall, compared to the sequential baseline implementation, the authors achieved a 3.4X speedup on the quad-core Intel Core-i7, and a 10.5X speedup on the GTX280. As illustrated by this example, application developers use their freedom to change algorithms and data structures to suit a particular platform. Therefore, using a benchmark suite with fixed algorithms and data structures does not provide comparisons which are useful to application developers.

[Feng and Zeng, 2010] is another example which illustrates the concerns of application researchers when performing comparisons. The authors proposed parallelizing chip-level power grid analysis on modern GPUs. The power grid analysis problem requires computing the voltage at all nodes of a circut as a function of time. It can be solved by formulating the problem into a linear system $Gx = b$, with G representing the resistance between nodes, and b representing the voltage sources. Traditionally, these problems have been solved by direct methods such as LU or Cholesky decomposition. However, memory access patterns for the direct methods are not very efficient. As a result, [Feng and Zeng, 2010] proposed multi-grid preconditioned conjugate gradient (MGPCG) method to solve the linear system. The conjugate gradient method is highly data parallel, and according to the experimental results, Feng achieved 25x speedup on GPU using the MGPCG method against the serial implementation using the Cholesky decomposition method.

The authors of [Feng and Zeng, 2010] are primarily concerned with advancing power grid analysis by introducing a faster solver. Their paper details several different algorithms and their parallelism implications. To get better data access patterns, they decided not to use traditional direct solvers, but instead use an iterative method better suited to the architecture of the processor they targeted. The authors detail significant algorithmic exploration, including a pure CG solver, a preconditioned CG solver, a multigrid solver, and finally the multigrid preconditioned CG solver. The 25x speedup number comes from exploring the algorithm space and parallelizing the algorithm. Although they are comparing their parallelized GPU iterative solver with a serial CPU direct solver, the comparison is still valid and of value to those using the serial direct solver. The experimental results are not meant as an architectural comparison of CPUs versus GPUs. Instead, they show how application researchers advance the state-of-the-art by exploring the algorithm space and introducing parallelism.

6 Example Comparisons for Architecture Research

We also cite two examples of useful cross-platform comparisons which contain details useful for architecture researchers.

[Kozyrakis and Patterson, 2002] evaluates their prototype vector processor with the EEMBC benchmarks and compares their results to five existing processor architectures. The authors mention that the processors were implemented by different design teams, using different process technology, at different clock frequencies, and with differing power consumption. The results for the embedded processors are normalized to clock frequency and code size comparisons are also made. The authors performed a multi-architectural comparison and provided in-depth analysis for the embedded applications running on the 6 different platforms. The EEMBC benchmark suite is well known and characterized, and results are publically available, making it possible for readers to reproduce the experiments, given the availability of the appropriate hardware platforms.

[Williams et al., 2009] evaluates Sparse Matrix Vector Multiplication across multicore processors from Intel, AMD and Sun, as well as the Cell Broadband engine. Although not specifically targeted at architecture researchers, this comparison does provide details of the hardware platforms being compared, describes the different optimizations which were applied to achieve high performance on each of the various architectures under consideration, as well analyzes the impact of various microarchitectural features for each platform with respect to the optimizations they employ. Additionally, they describe in detail the benchmark datasets and data structures they employ, as well as justify why the datasets constitute a good sampling of the space of sparse matrices. These details allow others to reproduce their work and build on it.

7 Conclusion

As parallel architectures continue to diverge, cross-platform comparisons will be increasingly important. Conducting such comparisons is inherently difficult: there is a wide spectrum of implementation concerns, ranging from silicon implementation to algorithmic choices, all of which critically influence the results of such comparisons. Additionally, these comparisons are conducted for widely different purposes, based on the needs and motivations of the researchers conducting the comparisons.

We have outlined two important, yet divergent viewpoints for conducting comparisons: the viewpoint of application developers, as well as the viewpoint of architectural researchers. Application developers focus on algorithmic innovation, co-designing algorithms for a particular domain to perform efficiently on parallel hardware, given a set of implementation constraints. Computer architects attempt to extract maximal performance from the architectures being compared on a particular benchmark suite, in order to compare architectural features. Although it is not realistic to expect all comparisons to include all the information necessary to satisfy all potential audiences, it is crucial that researchers conducting cross-platform comparisons choose a particular audience and attempt to provide enough information for that audience. Additionally, the results of such a comparison should be reproducible.

Papers like [Lee et al., 2010] and [Vuduc et al., 2010] raise important points; we agree that many papers which compare CPUs and GPUs, find a 100$\times$ speedup by using a GPU compared to a sequential CPU implementation, and then claim that GPU architecture is more suited to that computation, are taking their results far out of context. Without performing the work necessary to understand a workload at an architectural level across platforms, these claims are unsupportable. However, the narrower claim, that a GPU implementation was 100$\times$ faster than the legacy sequential implementation, is still valid and potentially of great interest to application developers using the legacy sequential implementation.

If a comparison is being made for architecture researchers, information needs to be presented about the silicon implementation of the processors being compared. Additionally, the benchmark suite being used needs to be fully documented, so that the audience can interpret and reproduce the results. We recognize that space in publications is highly constrained, which presents an obstacle to communicating complex details of cross-platform comparisons. We suggest the use of expanded technical reports or websites with freely available datasets and source code in order to communicate these details, since without them, the comparisons are impossible to interpret.

As parallel computing continues to mature, we believe that cross-platform comparisons will play an important role in helping both architecture researchers as well as application developers understand the landscape of parallel computing. Although conducting these comparisons can be complicated and at times contentious, the discussion they engender is essential to advancing our collective understanding of parallel architectures and applications.

8 Acknowledgements

Research supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). Additional support comes from Par Lab affiliates National Instruments, NEC, Nokia, NVIDIA, and Samsung.

References

[Aslot et al., 2001] Aslot, V., Domeika, M. J., Eigenmann, R., Gaertner, G., Jones, W. B., and Parady, B. (2001). SPEComp: a new benchmark suite for measuring parallel computer performance. In *Proceedings of the International Workshop on OpenMP Applications and Tools: OpenMP Shared Memory Parallel Programming*, pages 1–10.

[Bell and Garland, 2009] Bell, N. and Garland, M. (2009). Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–11, New York, NY, USA. ACM.

[Bienia et al., 2008] Bienia, C., Kumar, S., Singh, J. P., and Li, K. (2008). The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*.

[Chen et al., 2007] Chen, T., Raghavan, R., Dale, J. N., and Iwata, E. (2007). Cell broadband engine architecture and its first implementation: A performance view. *IBM Journal of Research and Development*, 51(5):559 –572.

[Chinnery and Keutzer, 2002] Chinnery, D. and Keutzer, K. (2002). *Closing the Gap Between ASIC & Custom*. Kluwer Academic Publishers.

[Feng and Zeng, 2010] Feng, Z. and Zeng, Z. (2010). Parallel multigrid preconditioning on graphics processing units (gpus) for robust power grid analysis. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 661–666, New York, NY, USA. ACM.

[Khronos Group, 2010] Khronos Group (2010). OpenCL. `http://www.khronos.org/opencl`.

[Kozyrakis and Patterson, 2002] Kozyrakis, C. and Patterson, D. (2002). Vector vs. superscalar and vliw architectures for embedded multimedia benchmarks. In *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, MICRO 35, pages 283–293, Los Alamitos, CA, USA. IEEE Computer Society Press.

[Lee et al., 2010] Lee, V. W., Kim, C., Chhugani, J., Deisher, M., Kim, D., Nguyen, A. D., Satish, N., Smelyanskiy, M., Chennupaty, S., Hammarlund, P., Singhal, R., and Dubey, P. (2010). Debunking the 100X GPU vs. CPU Myth: an Evaluation of Throughput Computing on CPU and GPU. In *ISCA'10*, pages 451–460.

[Lindholm et al., 2008] Lindholm, E., Nickolls, J., Oberman, S., and Montrym, J. (2008). Nvidia tesla: A unified graphics and computing architecture. *Micro, IEEE*, 28(2):39 –55.

[Owens et al., 2008] Owens, J., Houston, M., Luebke, D., Green, S., Stone, J., and Phillips, J. (2008). Gpu computing. *Proceedings of the IEEE*, 96(5):879 – 899.

[Seiler et al., 2008] Seiler, L., Carmean, D., Sprangle, E., Forsyth, T., Abrash, M., Dubey, P., Junkins, S., Lake, A., Sugerman, J., Cavin, R., Espasa, R., Grochowski, E., Juan, T., and Hanrahan, P. (2008). Larrabee: a many-core x86 architecture for visual computing. *ACM Trans. Graph.*, 27:18:1–18:15.

[Singh et al., 1995] Singh, J. P., Gupta, A., Ohara, M., Torrie, E., and Woo, S. C. (1995). The splash-2 programs: Characterization and methodological considerations. *ISCA'95*, pages 24–36.

[Vuduc et al., 2010] Vuduc, R., Chandramowlishwaran, A., Choi, J., Guney, M., and Shringarpure, A. (2010). On the limits of GPU acceleration. In *Proceedings of the USENIX Workshop on Hot Topics in Parallelism (HotPar)*, Berkeley, CA, USA.

[Williams et al., 2009] Williams, S. W., Oliker, L., Shalf, J., Yelick, K., and Demmel, J. (2009). Optimization of sparse matrix-vector multiplication on emerging multicore platforms. *Parallel Computing*, pages 178–194.

[You et al., 2009] You, K., Chong, J., Yi, Y., Gonina, E., Hughes, C., Chen, Y.-K., Sung, W., and Keutzer, K. (2009). Parallel scalability in speech recognition. *Signal Processing Magazine, IEEE*, 26(6):124 –135.