Future Generation Computer Systems 30 (2014) 46–58

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Exascale design space exploration and co-design

S.S. Dosanjh [a], R.F. Barrett [b,*], D.W. Doerfler [b], S.D. Hammond [b], K.S. Hemmert [b], M.A. Heroux [b], P.T. Lin [b], K.T. Pedretti [b], A.F. Rodrigues [b], T.G. Trucano [b], J.P. Luitjens [c]

[a] Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley, CA, 94720, USA
[b] Center for Computing Research, Sandia National Laboratories, Albuquerque, NM, 87185, USA
[c] NVIDIA Corporation, 2701 San Tomas Expressway, Santa Clara, CA, 95050, USA

HIGHLIGHTS

- A codesign-based methodology is described for exploring the exascale design space.
- Codesign requires a multi-faceted approach.
- Architecture testbeds are being used to study performance issues of key algorithms.
- Network bandwidth degradation studies help define requirements for future systems.
- The Structural Simulation Toolkit is described, with some example use cases.

ARTICLE INFO

Article history:
Received 2 June 2012
Received in revised form
21 February 2013
Accepted 13 April 2013
Available online 2 May 2013

Keywords:
High performance computing
Scientific computing
Co-design
Exascale preparation

ABSTRACT

The co-design of architectures and algorithms has been postulated as a strategy for achieving Exascale computing in this decade. Exascale design space exploration is prohibitively expensive, at least partially due to the size and complexity of scientific applications of interest. Application codes can contain millions of lines and involve many libraries. Mini-applications, which attempt to capture some key performance issues, can potentially reduce the order of the exploration by a factor of a thousand. However, we need to carefully understand how representative mini-applications are of the full application code. This paper describes a methodology for this comparison and applies it to a particularly challenging mini-application. A multi-faceted methodology for design space exploration is also described that includes measurements on advanced architecture testbeds, experiments that use supercomputers and system software to emulate future hardware, and hardware/software co-simulation tools to predict the behavior of applications on hardware that does not yet exist.

Published by Elsevier B.V.

1. Introduction

The United States Department of Energy's mission needs in energy, national security and science are predicted to require a thousand-fold increase in supercomputing performance during the next decade [1]. However, the transition to Exascale systems that are operable within affordable power budgets will not be possible based solely on existing computer industry roadmaps [2]. The conclusion is therefore that we not only need to support an acceleration of industry roadmaps to deliver power efficient architectures but that we also need to augment this with modifications, or in some cases rewriting of applications to utilize new approaches to hardware design and significantly increased scale [3]. The benefits of doing so are profound as they impact the entire computing industry, addressing cross-cutting issues such as energy efficiency, concurrency and programmability for users of single workstations, data centers and large supercomputers. For the users of Exascale machines there will be additional challenges including the scalability and reliability that are brought about by the extreme size of such systems.

Given the complexity of constructing such large systems and the problems associated with modifying applications to run on them, a dialog needs to be established between computer companies and application developers where feedback is able to rapidly assess and optimize designs as they are created. In this process we envisage assessment based on balancing performance benefit versus cost in terms of software complexity, portability, silicon area, *etc.* In order for trust to exist in this dialog a number of approaches might be considered including execution on prototype or early design hardware, the construction of application models including simulators or analytic performance models and an attention to creating solutions that work across a broad range of applications and do not benefit a single problem.

* Corresponding author.
E-mail address: rfbarre@sandia.gov (R.F. Barrett).

0167-739X/$ – see front matter. Published by Elsevier B.V.
http://dx.doi.org/10.1016/j.future.2013.04.018

The scale of modern scientific applications is however a potential limiter for rapid prototype assessment. Applications are typically millions of lines of source and are written to use complex algorithms and data structures. Whilst our eventual goal is to run applications on such large systems the effort required to port them is likely to be prohibitive if multiple platforms must be assessed in short time frames. It is in this context that the notion of a *mini-application* has been developed—a mini-app is a condensed implementation of one or multiple key performance issues that affect parent codes, written to be amenable to refactoring or change but representative enough to be useful in the scientific problem domain.

Our proposed methodology for Exascale design space exploration, which we discuss in this paper, includes measurement on prototype hardware, experimentation in the form of refactoring and re-implementation using a variety of programming models and algorithms and prediction using architectural simulators. To this end, we are investigating several architectural testbeds which are representative of industry trends including Intel's Many Integrated Core (MIC) processors, GPUs from NVIDIA, Fusion APUs from AMD and nodes from Convey and Tilera. Such studies are providing useful feedback to computer architects, application developers and algorithm researchers. Our experimentation is also wider than just hardware, including evaluation of execution models such as ParalleX and low-level activities such as direct measurement of energy use in contexts such as the variation of network injection bandwidth.

The ability to predict the performance, and more importantly the performance limitations, of hardware which does not currently exist or is significantly different from contemporary systems is a key facet of design exploration. Since many proposed Exascale point designs are currently proprietary or encumbered with intellectual property, many of our early evaluations are being conducted using the notion of an Abstract Machine Model (or AMM) which defines the key architectural building blocks but no specific detail. We then are able to augment AMMs with details provided by performance models, architectural simulators and information obtained from our mini-applications running on test-bed platforms to inform us of performance trade offs and available design decisions.

In this work we provide a detailed overview of our Exascale methodology including descriptions of currently running projects to produce relevant mini-applications, accurate and relevant architectural simulation tools and our prototype test bed program which is being used to drive programming model assessments and improvements in our simulation and modeling capabilities. In addition we describe a validation methodology which is being developed to demonstrate the applicability of mini-applications to their parent codes enabling HPC vendors and researchers to have a high degree of confidence in results obtained from studies using mini-apps.

2. Miniapplications

Full-scale computational science and engineering (CSE) applications are often large and complex, depend upon numerous third-party libraries and require substantial systems programming expertise in order to compile and execute. Because of this, we are compelled to use application performance proxies for early-phase design studies meant to target a particular suite of applications. Numerous types of proxies are useful for design studies, depending on the specific context. Fig. 1 summarizes some of the key proxies used by the systems performance community.

Application performance is determined by a combination of many choices: hardware platform, runtime environment, languages and compilers used, algorithm choice and implementation, and more. In this complicated environment, we find that the use of mini-applications is an excellent approach for rapidly exploring the parameter space. Furthermore, use of mini-applications enriches the interaction between application, library and computer system developers by providing explicit functioning software and concrete performance results that lead to detailed, focused discussions of design trade-offs, algorithm choices and runtime performance issues.

Unlike a benchmark, the result of which is a metric to be ranked, the output of a miniapp is a richer set of information, which must be interpreted within some, often subjective, context. We distinguish this from a *compact-application* whose purpose is to replicate a complex domain-specific behavior being used in a parent application. Miniapps are designed specifically to capture some key performance issue in the full application but to present it in a simplified setting which is amenable to rapid modification and testing. Note that this is also distinct from a *skeleton application*, which is typically designed to focus on inter-process communication often producing a "fake" computation. Miniapps instead create a meaningful context in which to explore the key performance issue. Within many of the ASC programs, miniapps are developed and owned by application code teams; are limited to $O(1\,\mathrm{K})$ source lines of code (SLOC) and are intended to be modified with the only constraint being the continued relevance to parent applications.

2.1. Mantevo

The Mantevo project [4] provides a set of proxies, or "miniapps", which enable rapid exploration of key performance issues that impact a board set of scientific applications of interest to the ASC and broader HPC community. Mantevo miniapps are tools with uses throughout the co-design space [5]. They are intended to be fluid, and a mechanism to explore issues relating to hardware performance, programmability, porting, etc. As part of the ongoing work in developing miniapps under Mantevo, a comprehensive initial validation exercise [6] has recently been conducted to ensure the first full release of codes is able to provide strong behavioral correlation to parent physics and engineering application currently in use.

The Mantevo Project started in 2006 as an effort to develop tools and environments for studying computational science and engineering application performance. Very early on in the project, miniapps emerged as important tools. They provide the right balance of complexity – capturing the nuances of performance coupling between distinct computational phases – and ease of use and refactoring to be accessible and meaningful co-design tools.

The initial miniapp HPCCG was designed to study the performance characteristics of preconditioned iterative methods used in Trilinos [7]. System developers were looking for a small representative code to answer questions about the direction of some coding implementations targeting emerging and expected future architectures, including multi-core, many-core, and GPU-accelerated high performance computers. HPCCG was frequently used for compiler studies, processor comparison and more. Furthermore, the depth of conversation between algorithms and systems developers grew with a small, concrete target as the means for exploration. Based on this experience we started identifying and deploying miniapps across other application areas of interest.

Each Mantevo miniapp is designed to focus attention on one or a few key performance characteristics of an application or class of applications, enabling agile exploration of a variety of issues that impact performance, ranging from low-level hardware capabilities to the application.

The current set of miniapps in the Mantevo project are listed in Table 1. All of the miniapps in Table 1 are available via the GNU

Proxy Type	Characteristics	Refactoring Scope	Performance Modeling Scope	Size (LOC)
Kernels	Small, self contain code fragments that represent key "hotspots" in an application.	Can often be rewritten, but small size limits options.	Narrow study and refactoring of hotspots.	10–100
Benchmarks	Typically meant to be static code with precise input and usage restrictions.	Limited, if allowed at all.	System performance for a prescribed software implementation.	100–10K
Compact App	Simplified, but complete physics simulation.	Can be complete rewritten, but may be difficult to do if code is large.	Useful in most settings except very early design studies.	10K–100K
Skeleton app	Accurate interprocessor communication model with synthetic computation.	Some, but only at the level of communication strategies.	Interprocessor communication performance; asymptotic communication complexity and scheduling.	500-5K
Miniapp	Focused on one or a few performance-impacting aspect of the application.	Complete rewrite or reorganization, including new languages, new system features, etc.	Any phase of system design, especially an app-system co-design environment.	1K–10K

Fig. 1. Summary of application performance proxies.

Table 1
List of current Mantevo miniapp efforts.

Miniapp	Description
HPCCG	Sparse linear algebra (Krylov) solver
miniFE	Unstructured implicit FEM/FVM
phDMesh	Explicit FEM, contact detection
miniMD	Molecular dynamics for force computations
miniXyce	Circuit RC ladder
miniExDyn[a]	Explicit Dynamics Finite Element
miniITC[a]	Implicit Thermal Conduction Finite Element
miniGhost[a]	FDM/FVM
miniAero[b]	Aero/fluids
miniDSMC[b]	Particle-based simulation of low-density fluids

[a] New.
[b] Under development.

Lesser General Public License (LGPL) [8] and are downloadable from the Mantevo website.[1]

2.2. Validating miniapps

Validation is the process of determining the degree to which a model is an accurate representation of the "real world" from the perspective of the intended uses of the model.[2] For our purposes, validation is the process of assessing the evidence of how closely the miniapp resembles the full application in the performance domain of interest. That is, within the context of the intent of the comparisons of a model with the "real world", we must verify that the applications ("real world") and miniapps ("model") compare well in the performance dimensions of interest. All of the work (and possibly art) in this methodology will be defining a set of comparisons that allow us to draw conclusions of this kind about the miniapps. We must also understand how close these comparisons should be for us to be able to conclude that the miniapps are suitably accurate models of real code performance, or that they are not. There will clearly be significant components of judgment embedded in this methodology given the difficult nature of this problem.

Our methodology, developed in the spirit of experimental validation as described in [9–12], is designed to answer the question: "Under what conditions does a miniapp represent a key performance characteristic in a full app?" This approach requires extensive knowledge of, and experience developing, executing, profiling, maintaining, and extending multi-scale, multi-physics scientific and engineering application software, targeting highest performance computing platforms. It also requires a strong understanding of the miniapps and their intended use: what they are intended to represent and what they are not intended to represent.

The methodology is as follows: For a set of diagnostic runtime performance characteristics or elements, which we loosely refer to as the *performance domain*,

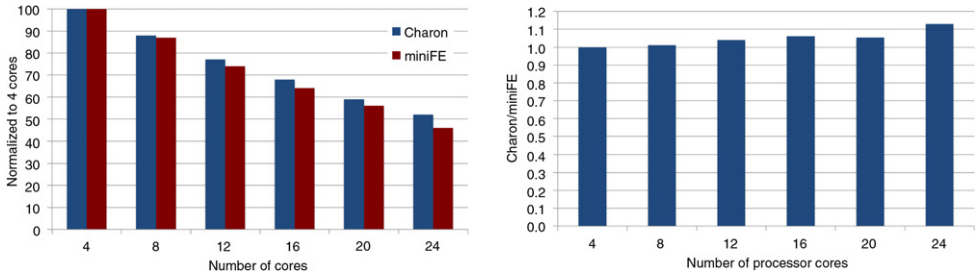
$$\{D\} = D_1, D_2, \ldots, D_m, \tag{1}$$

let

$$\{B\} = B_1, B_2, \ldots, B_n, \tag{2}$$

[1] http://mantevo.org.

[2] These terms are as defined by the American Society of Mechanical Engineers (ASME, 2006) and the American Institute of Aeronautics and Astronautics (AIAA, 1998), and this usage has basically been adopted by the United States Departments of Energy (DOE) and Defense (DoD). IEEE definitions (IEEE, 1991) are also useful and relevant in this context.



(a) Efficiency. (b) Metric.

Fig. 2. Effects of the number of cores per node on the FEA and solver phases of Charon and miniFE.

be a corresponding set of baseline full application observational referents, (the "validation data") and let

$$\{A\} = A_1, A_2, \ldots, A_p, \tag{3}$$

be a set of corresponding miniapp measurements, for p n.

We then consider the difference between the application referents and the miniapp measurements in the performance domain defined by (1) as some kind of mathematical norm, which we will also call a *validation metric*:

$$X_i = B_i - A_i \quad i, \qquad i. \tag{4}$$

Then assessment of the validation metric information might then be posed as:

$$V_i = \begin{cases} \text{pass}, & \text{for } T_i^1 \quad X_i \quad T_i^2 \\ \text{caution}, & \text{for } T_i^2 \quad X_i \quad T_i^3 \\ \text{fail}, & \text{for } X_i \quad T_i^1 \text{ or } X_i \quad T_i^3 \end{cases} \tag{5}$$

where V_i is a validity statement attached to performance domain dimension i for some thresholds T_i^j, for $j = 1, \ldots, 3$.

While Eq. (5) looks like a generally useful algorithm for assessment, we caution there is a great deal of overloading going on in this simple expression. For example, the choice of thresholds could clearly be extremely difficult. The willingness to even evaluate validity based on a relatively direct threshold assessment is open to debate, and developing the set $V_i, i = 1, \ldots, n$ leaves open the issue of how all of this information is combined into a single appraisal of the validity of the minapp. Nonetheless, this logic is a clear illustration of the kind of ideal thinking that should underlie the validation assessment of miniapps.

This framework provides direct advantages. First, the input information D, B, and A and are open to challenge and refinement, are mutable and extensible, and thus the role interpretive judgment in the final results of validity assessment is transparent within the context of use. For example, new diagnostics, new or corrected baseline observations, and new or corrected measurements could be added to the model in the service of better assessment. Second, the way the results are computed can be easily subjected to peer-review scrutiny.

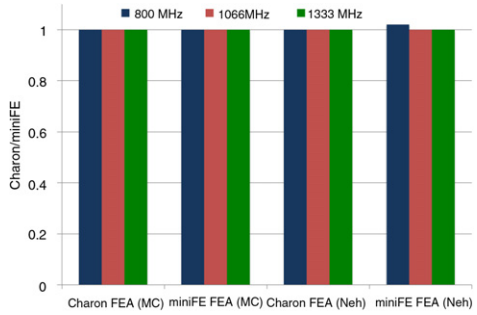
We illustrate this method by examining the relationships between an application named Charon and the Mantevo miniapp called miniFE. Charon [13,14] is an electronic device simulation application code, solving the drift–diffusion equations that relate the electric potential to the electron and hole concentrations in these devices. One of the discretization approaches employed in Charon is a finite element method (FEM) on unstructured meshes. Execution is characterized by a finite element assembly step (FEA) followed by the solution of the nonlinear system using a fully-implicit Newton–Krylov (NK) solution approach. Performance of NK is dominated by the Krylov solver, either BiCGSTAB [15] or GMRES [16]. Charon employs the solvers in Trilinos[7], including the Krylov solvers from the Trilinos Aztec package [17]. MiniFE is an implicit finite-element miniapp which includes a FEA step as well as a solver phase using the Conjugate Gradient method [18].

We examine performance on three distinct architectures. Cielo, a Cray XE6, consists of dual-socket 8-core AMD Opteron Magny-Cours processor based nodes connected by a custom Gemini network configured as a three dimensional torus. Chama consists of dual-socket 8-core Intel Sandy Bridge processor based nodes connected by a Qlogic InfiniBand network configured as a fat tree. Red Sky consists of dual-socket quadcore Intel Nehalem processor based nodes connected by a Mellanox InfiniBand network configured as a three dimensional torus. We supplement these machines with related architectures that allow for more flexible experiments.

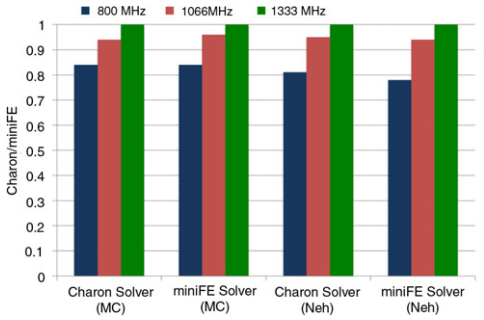
We begin by specifying on-node memory bandwidth as a diagnostic. A typical means for exploring this issue is to vary the number of processor cores employed on the node and comparing the resulting performance efficiency. Fig. 2(a) illustrates the results of this experiment applied to the solver phases on a Cray XE6 node configured using dual-socket 12-core AMD Opteron Magny-Cours processors. As has been observed in a variety of cases [19–21], the efficiency of each socket decreases as the number of cores per node increases. A proportional comparison (Fig. 2(b)) reveals that the responses by Charon and miniFE are within about 13% at worst, suggesting that the miniapp is predictive of the effects of memory bandwidth on Charon. However, in a validation study, stronger evidence is needed to make this claim.

Using a dual-socket quadcore Intel Nehalem 5560 clocked at 2.8 GHz processors and a dual-socket 8-core AMD Magny-Cours 6136 clocked at 2.4 GHz, experiments were configured to better focus on memory bandwidth. The machines were set up to provide memory speeds of 800 MHz, 1066 MHz, and 1333 MHz. Results, illustrated in Fig. 3, show that the FEA phases for miniFE and Charon are not impacted by the change in bandwidth, while their solvers are. A proportional comparison (Fig. 3(c)) shows miniFE is within 4% of all measures of Charon, leading us to claim that miniFE is predictive of Charon with regard to on-node memory bandwidth.

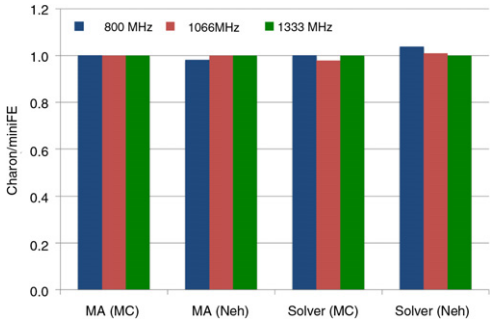
The next diagnostic considered cache performance, again with the separation of between the FEA and solver phases, and again using the Nehalem and Magny-Cours nodes, each with three levels of cache (L3 is shared across cores in a socket for Nehalem, and across cores in a die for Magny-Cours). The hit rate, defined as the proportion of the number of times the processor finds needed data in a cache with the total number of times it looks for data in that cache, plays a significant role in processor performance. Results are shown in Fig. 4. For the FEA phase, Charon and miniFE show strong use of level 1 cache, with a proportional difference of no more than 3%. However, level 2 and 3 hit rates are significantly different, with miniFE 3 and 6 times, respectively, from Charon, leading us to claim that the cache performance of FEA in miniFE is not predictive of that for Charon. For the solver phase, we believe that cache performance is predictive. Although the thresholds for acceptance for level 2 and 3 are arguably high (approximately 20%), the trends are clear.



(a) FEA, normalized data.

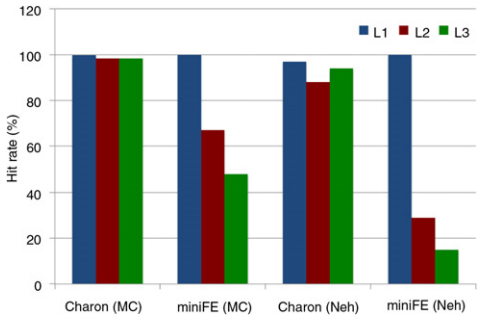


(b) Solver, normalized data.

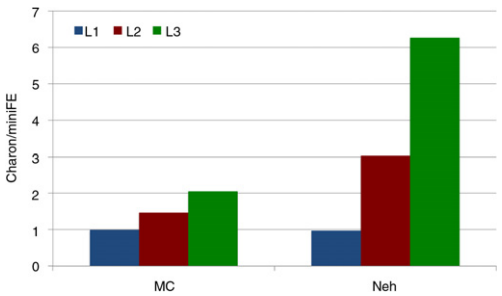


(c) Metric.

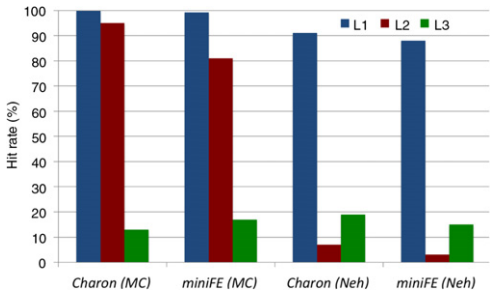
Fig. 3. Effects of memory speeds on the FEA and solver phases of Charon and miniFE. Performance is relative to 1333 MHz.



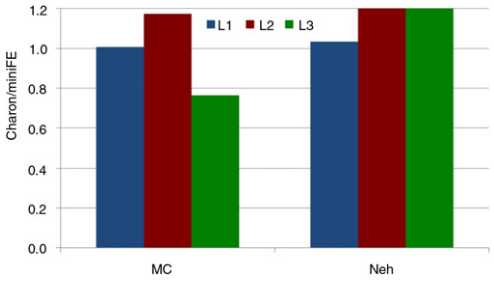
(a) FEA hit rate.



(b) FEA metric.



(c) Solver hit rate.

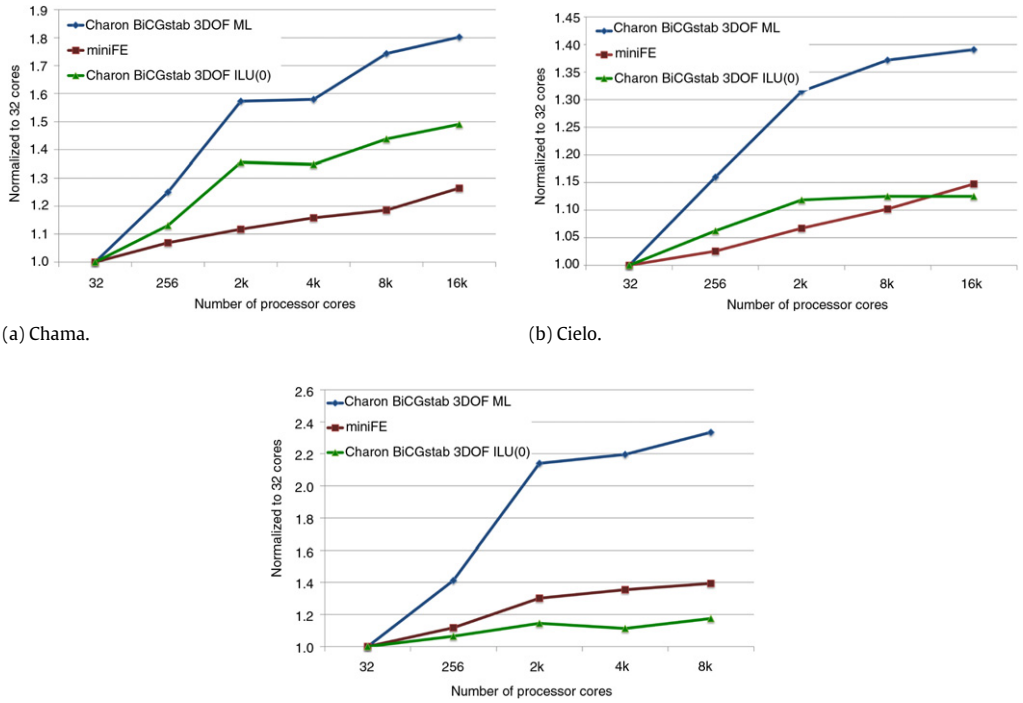


(d) Solver metric.

Fig. 4. Cache behavior of the FEA and solver phases of Charon and miniFE.

Most interesting is that Charon's surprisingly low level 2 hit rate seen on the Nehalem is also seen with miniFE. This is unexpected based on past observations on related architectures, but care must be taken with regard to attribution. Additional experiments are required to make strong causal claims, since it is possible that measurement intrusion is to blame, or perhaps a hardware configuration issue.

Next we examine weak scaling characteristics of Charon and miniFE to large core counts. Diagnostics include the Charon/Aztec BiCGSTAB solver with two preconditioning strategies, an incomplete factorization algorithm with no fill (ILU(0)) and a multilevel ("ML", e.g. multigrid) algorithm [22]. Results for each are analyzed in comparison to miniFE's unpreconditioned Conjugate Gradient solver. The general idea is that Krylov solvers perform common computations (e.g. addition and scaling of vectors, inner products, and sparse matrix–vector products). Further, applications typically use a breadth of preconditioners, so our goal is to understand where specificity is required and where it is not necessary.



(a) Chama.

(b) Cielo.

(c) Red sky.

Fig. 5. Relative scaling of solvers.

Performance is illustrated in Fig. 5. We have not yet determined an effective means for analytically comparing scaling behavior. Instead we can reason about the curves by first noting that performance is different on different architectures, which we speculate is a function of the difference preconditioning strategies. Although the difference between miniFE and Charon with ML preconditioning is large, this is not reason enough to reject the relationship. Instead, we claim that miniFE is not predictive of Charon with ML because miniFE does not include the sorts of computations found in ML. For example, an analysis of the interprocess message passing requirements shows that ML sends over 40% more messages per core than the non-multilevel preconditioners.

The difference between Charon with ILU(0) preconditioning and miniFE is less clear, with reasoning driven from different perspectives in the co-design space. For example, from the perspective of some hardware architects, these two approaches are not predictive. However, an algorithm developer investigating new programming models could view miniFE performance as reasonably predictive. Therefore we assign this diagnostic a *caution* assessment.

In summary, ensuring that a miniapp completely fulfills its intent is a difficult and probably ongoing task. Further, the runtime behavior of a complex scientific application is typically problem dependent, and therefore it is important to understand the different ways that a code can be used and to have a means for configuring the miniapp to mimic the important features under consideration. Thus our approach is to continue to build up "a body of evidence" in support of the goals of a miniapp, combining formal validation techniques with our knowledge and experience bases. This includes statistical analysis of multiple runs, adding, for example, error bars to the graphs and thus informing our analysis.

3. Architectural testbeds

Advanced architecture testbeds provide a means for exploring performance issues that are expected to impact our applications on future architectures. These have proven to be useful tools for examining many of the issues throughout the design space, focusing our efforts and enabling reasoning about specific characteristics and capabilities as they relate to our goals.

In this section we describe three such systems, and illustrate how one such architecture, in conjunction with direct vendor interaction, can be used to prepare our applications for what we believe will be significant changes.

3.1. Teller—AMD fusion cluster

Teller is a 104-node cluster of single-socket AMD A8-3850 Llano Fusion APU nodes. Each APU-die comprises four K-10-class AMD x86-64 cores running at 2.90 GHz. Cores have private 64 kB L1 instruction and data caches and a 1 MB level-2 (giving a 4 MB L2 in total). The GPU portion of the APU is a modified 400-core Radeon HD 6550D which runs at 600 MHz. GPU cores are currently 5-way SIMD. In order to enable the study of performance as a function of memory frequency, 100 cluster nodes contain 16 GB DDR3 1600 MHz memory and 4 nodes contain 8 GB DDR3 memory at 1866 MHz. All nodes are equipped with a single 256 GB Micro C400 SSD storage device enabling us to study local checkpointing strategies. The machine interconnect employs QLogic QSFP QDR InfiniBand arranged in a 2-level fat-tree topology.

3.2. Arthur—Intel MIC cluster

Arthur is a 42 node cluster of Intel MIC (Many Integrated Core) accelerator cards. Nodes in the cluster comprise a dual-socket hex-core Westmere-EP Xeon processors, clocked at 3.47 GHz with 24 GB of DDR3-1600 MHz memory, connected via a PCI-Express connection to a Knights Ferry co-processor (giving 84 cards/co-processors in total). Each Knights Ferry processor is itself a 30-core chip clocked at 1.05 GHz and connected to 2 GB of GDDR5-1800 MHz memory. The machine interconnect is provided by Mellanox Infiniscale IV QDR Infiniband via a fat tree topology with five 36-port switches.

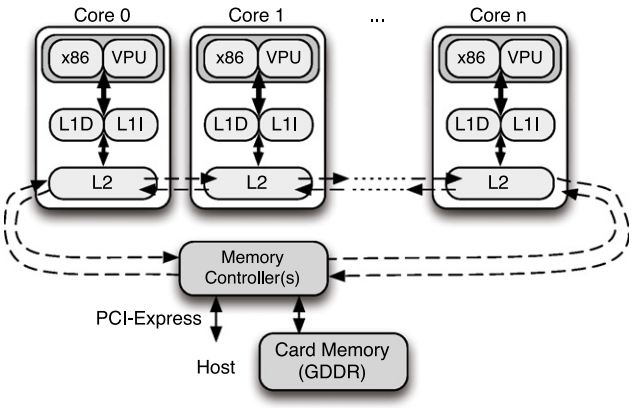


Fig. 6. Intel MIC architecture.

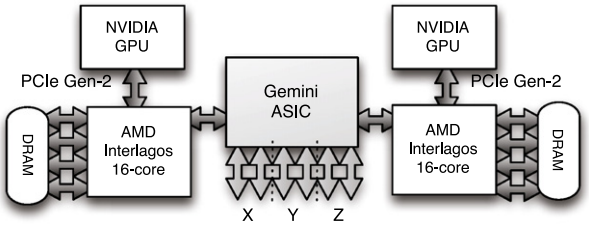


Fig. 7. Cray XK6 node.

Each processor core on the Knights Ferry device is a modified variant of Intel's x86 architecture in which computation has been augmented with a 16-wide vector processing unit (VPU) and four-way hardware threading (see Fig. 6). Cores have access to a 32 Kilobyte (KB) L1 instruction cache, 32 kB L1 data cache, and an inclusive 256 kB L2 cache. A coherent ring network connects L2 caches to the memory controller and provides high bandwidth core-to-core data transfers. Transfers too and from the host occur via DMA transfers over the accelerator's PCI-Express connection.

3.3. Curie—NVIDIA/cray cluster

The XK6 is the latest refinement to the X-family of machines developed by Cray. In this new series one processor is removed from each compute node and replaced with an NVIDIA Tesla-series GPU using a PCI-Express Gen-2 bus for connectivity (see Fig. 7).

At the time of writing the GPU used for node design is an NVIDIA "Fermi"-class GPU with 6 GBytes of GDDR5 memory and 16 streaming multiprocessors (SM), each containing 32 thread processing cores, are provided on the GPU card, enabling high speed transfers to/from the on-chip compute units. The Curie testbed system consists of 52 nodes, with AMD Opteron 2.1 GHz 16-core Interlagos 6272 processors and 32 GB of system memory. The Interlagos is also of interest in our studies since it utilizes the recently announced Bulldozer core in which a core-pair each have two integer processing units (one per core) but a unified floating-point pipeline. This design is intended to reduce power for scientific computations where multiple integer operations are executed to obtain array addresses before a floating-point calculation can be issued.

3.4. Understanding computation on a GPU

Computation on a GPU can be significantly different from that on a CPU. In this section we focus on the FEA (finite element assembly) phase of miniFE. Implemented using the CUDA programming model and executed on an NVIDIA Fermi GPU of the sort found on Curie, this work illustrates a key performance issue, suggests an algorithmic modification, and predicts how an expected hardware modification will improve performance of this computation.

The FEA phase involves computing the element operators for each element which are then summed into the matrix. Threads operate on separate elements with the computation of the element operator and the summation into the linear system performed within a single kernel. Although the computation of the element operators is embarrassingly parallel, the summing into a linear system requires synchronization to avoid data race conditions. The use of a single kernel is preferred in this instance because it avoids having to store the state for the element operator and then having to later re-read that state during summing into the linear system.

In order to minimize code modifications, one thread operates on one element in the construction of the element operator. However, the code was modified so that matrix coefficients are stored in ELL sparse matrix format rather than compressed-row (CSR) form, shown to result in stronger performance in the subsequent solver phases [23]. Atomic addition operations prevent race conditions in updating the global matrix. Computation of the element operator is floating-point intensive, including computing the matrix determinant and the Jacobian. However, analysis shows that the performance is bandwidth bound due to register spilling.

The cause of this register spilling was traced to the element operator which requires a large thread state, including 32 bytes for node-IDs, 96 bytes for node coordinates, 512 bytes for the diffusion matrix, 64 bytes for the source vector as well as data to store the Jacobian and matrix determinant. The Fermi architecture supports up to 63 32-bit registers per thread, limiting the total register storage to 252 bytes. Thus any additional state will be spilled to (at least) L1 cache and potentially further.

The L1 cache, up to 48 kB, is shared by 512 threads, providing (on average) 96 bytes to each thread. The L2 cache is 768 kB shared by 8192 threads, again leaving only 96 bytes of storage per thread. This is not sufficient to store the operator state, so registers are spilled to global memory, causing the computation to become bandwidth bound.

One method to improve the performance of bandwidth bound kernels is to increase the occupancy. However, in this case, the kernel's occupancy is limited by register usage. Since the register usage is higher than is available in hardware it is not possible to increase this occupancy without further increasing register spilling.

We tuned the kernel to reduce register usage, including algorithmic changes that exploiting symmetry in the diffusion operator and reordering computations so that data is loaded immediately prior to being used. We have also applied several traditional optimization techniques including pointer restriction, inlining of functions, and unrolling of loops. Finally, we also position a portion of the state in shared memory and experimented with L1 cache sizes. The best performance is achieved by placing the source vector into shared memory and enabling a larger L1 cache. Whilst these optimizations greatly reduce register spilling, 512 bytes of state is still spilled per thread. To ensure fair comparison, all optimizations that were applicable to the original CPU code were back ported, and resulted in improved CPU performance.

The performance of the CUDA version of miniFE was compared to the MPI-parallel version of miniFE running on a Tesla M2090 and a hex-core Intel Xeon 2.7 GHz E5-2680, using various problem sizes of N^3 hexahedral elements. The speedup for each of the three phases of the algorithm is reported in Fig. 8.

The assembly realizes a four times speedup and the solve phase is three times faster. The generation of the matrix structure exhibits a slowdown because it is computed on the host in CSR format, transferred to the device, and then converted to ELL format. Whilst it is possible to move this computation to the device, this is a low priority based on its contribution to total runtime.

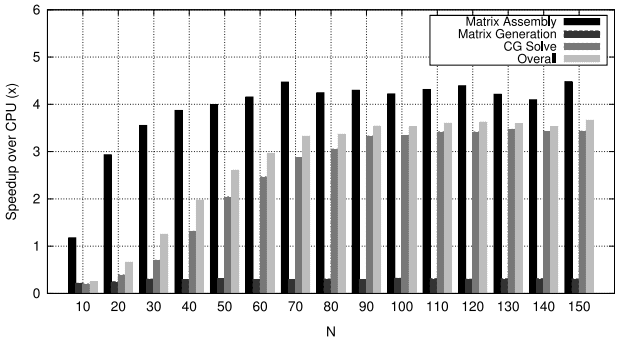


Fig. 8. Speedup of miniFE CUDA implementation NVIDIA Fermi M2090 vs. hex-core 2.7 GHz E5-2680.

Future generations of NVIDIA systems are expected to address some of the findings from this study, including an increased number of registers per thread and increases in the size of L1 and L2 memories. Improvements in the CUDA compiler may also lead to a reduction in the number of register spills or the impact that register spills will have on execution time.

4. Experimentation

While small to medium sized testbeds can be used to gain much needed experience with new technologies, many issues will only manifest themselves at relatively large scales, making it necessary to use full scale systems, usually reserved for production workloads, as forward looking "testbeds". Short periods of dedicated access to large systems can provide a wealth of useful data. For example, such dedicated time can be used to gain insight into application bottlenecks and the effects of system software on application performance. One study [24] used kernel-level noise inject techniques to study the effect of operating system (OS) noise on applications. This study was carried out during dedicated system time using a custom modified kernel to implement controlled injection of OS noise. Full scale application runs were then made using the modified kernel and the effects of various noise signatures were studied. Such studies would have been impossible without dedicated time and the ability to modify the operating system used on the supercomputer.

It is also desirable to do empirical experimentation using current systems configured to emulate the expected performance characteristics of future systems. Having flexibility in system configuration enables specific studies, where the number of system variations can be minimized. This allows the researcher to isolate the effects of changing specific system attributes. The next section briefly describes how a Cray XT5 testbed system was used to gain insight into the sensitivity of several scientific computing applications to network injection bandwidth. A more detailed description is provided elsewhere [25]. Results from experiments such as these can be useful for informing analytic modeling efforts, validating simulation models, and, ultimately, influencing the design of future supercomputer systems and applications.

4.1. Bandwidth degradation studies

As a proof-of-concept of our approach, we used a small 160-node, 1920-core Cray XT5 testbed system to evaluate a set of production applications from the Department of Energy's Advanced Simulation and Computing (ASC) code suit for their sensitivity to network injection bandwidth. Working with Cray, we modified the XT5 system's boot firmware to configure each compute node's link to the network for reduced bandwidth. All other system characteristics, such as compute node computational performance and memory bandwidth were left unchanged.

The resulting experimental platform was then used to dial-in several different network to compute system balance ratios, and evaluate the performance of the test applications at each operating point. The key benefits of this approach are that it is much faster than software-based simulation and that full-scale application testing can be performed, rather than the short application traces used in most simulation studies. The key disadvantage of our approach is that it is obviously limited in how far the emulated future system can diverge from the underlying physical system.

Fig. 9 presents results for four applications: CTH, SAGE, xNOBEL, and Charon. Each application was evaluated with our experimental platform configured for full (3.2 GB/s), half (1.6 GB/s), quarter (800 MB/s), and eighth (400 MB/s) network injection bandwidths. The results are presented as performance relative to full injection bandwidth, and are therefore unit-less. For example, a value of 1.5 indicates that the performance measured with the degraded injection bandwidth configuration was 1.5 times slower than with no degradation.

The results show that each application is affected by the reduction in network bandwidth differently. Charon, which is a semiconductor device simulation code, is known to be sensitive to communication latency and sends many small messages. The experimental results confirm this, showing Charon to be essentially unimpacted by network injection bandwidth. It would therefore be possible to save considerable network power for this application, since network power usage is proportional to bandwidth.

On the other hand, CTH and SAGE are more strongly impacted because they send relatively large messages that must complete before moving on to the next time step. For example, the experimental results show over a factor of two slowdown for CTH when the system is configured for one-eighth network injection bandwidth. This would likely not be a good power-performance tradeoff, since it would represent a 30% power saving for a 100% increase in runtime, assuming a system with an equal power split between CPU, memory, and network. The most energy efficient configuration would in fact be the one with full bandwidth. The xNOBEL results show a similar falloff past 384 cores, likely due to the loss of the ability to overlap computation and communication.

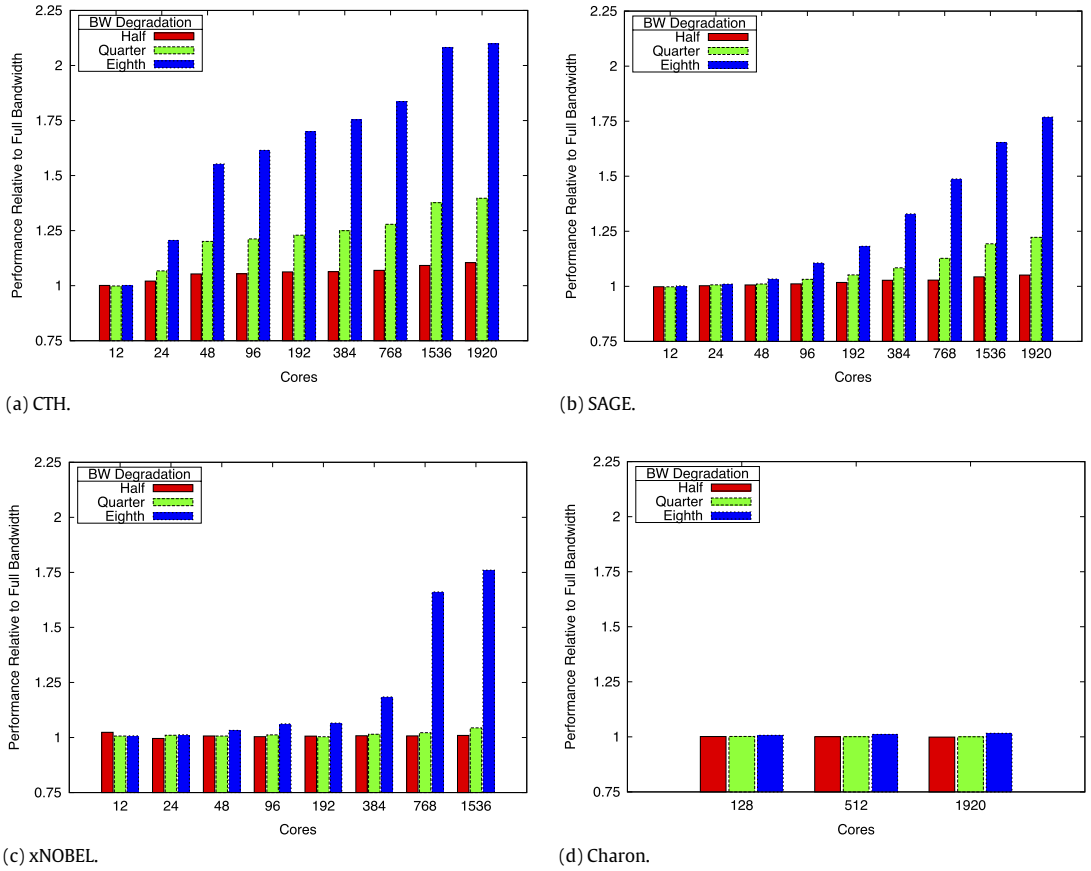
Our ongoing work involves extending our proof-of-concept study to a much larger machine, such as the 143 K core Cielo system. Even the small-scale results presented here are enlightening, and provide motivation for some form of network power-performance configurability in future systems. The planned larger-scale experiments will provide further evidence and, we hope, provide useful input for the co-design process leading up to exascale systems.

5. Prediction

It has been said that it is difficult to make predictions, especially about the future. This is particularly true in High Performance Computer architecture, where even making "predictions" about the present is a notoriously difficult problem. Many of the great advancements in architecture, which have improved average performance, have come at the cost of increasing system complexity and decreasing transparency. Branch predictors, memory prefetchers, caches, adaptive routing, even parallel computing itself have all made it more difficult to understand, reason about, and predict the performance of modern computers.

For HPC systems, the prediction problem is complicated by a number of factors.

- *Scale*: The size of HPC systems has been steadily growing. The number of processor cores, nodes, and threads in a large HPC system has been increasing much faster than Moore's Law. Because detailed simulation is much slower than actual hardware, the ability to simulate large systems has always lagged



(a) CTH. (b) SAGE.

(c) xNOBEL. (d) Charon.

Fig. 9. Application results.

actual system deployments. Simulators and models must abstract away many details of machine performance to provide timely predictions.

- *Audience*: Traditionally, system performance prediction was the domain of system architects. However, with co-design applications and algorithms, writers also need access to predictions about future systems and need to start reasoning about how to program them. Additionally, procurement agencies need predictive capabilities to make decisions on future machine purchases and technology investments. Language, runtime, and OS developers also need to understand how future machines will perform and be programmed. Simulators and models must be accessible to a number of audiences.
- *Complexity*: The set of HPC applications is getting more complex. Traditional physics applications have moved from 1D to 2D to 3D and are now becoming coupled multi-physics applications with greater adaptivity, load balancing, and visualization/analysis requirements. Additionally, new application areas such as informatics are emerging in the HPC space. These new application areas may not map well to existing programming models and practices. Simulators and models must be flexible enough to model emerging applications.
- *Objective functions*: In addition to performance, systems must be optimized with a growing number of new objective functions. Several studies have estimated that an exascale-class supercomputer will require 100s of Megawatts of power unless the system is reorganized in a number of ways [26,27]. As these systems increase in size, the sheer number of components threatens overall system reliability. New packaging techniques such as 3D stacking or Silicon carriers, may have an impact on system temperature. Accurate temperature modeling is required for accurate power and energy modeling due to its effect on leakage current. Temperature also impacts reliability by leading to electromigration and dielectric breakdown failures. Some failure modes, such as mechanical failures, are dependent in temperature history and the number of thermal cycles [28]. The most critical objective function may also be the most difficult to measure: Cost. In addition to technical factors (chip area, yield rates, pin counts), market forces can play a major role as can transient disasters (e.g. the 2011 monsoon floods in Thailand caused global hard disk prices to increase by 20%–40%). Simulators and models must address a growing number of evaluation criteria for HPC systems.
- *Proprietary data*: Commercial hardware vendors often keep secret critical implementation choices which have a major impact on performance. This lack of information has a severe impact on the accuracy of simulations and makes it difficult to model existing systems.

To mitigate these hurdles, a design space exploration needs to use multiple models of the system and multiple prediction techniques. These techniques range from "Back of the Envelope" estimates and "spreadsheet models" to simulation and hardware prototyping. These efforts require close collaboration and consultation with semiconductor fabrication and packaging groups to allow understanding of emerging technologies such as 3D die stacking, silicon photonics, and subthreshold circuits.

For each of these prediction techniques, a design team requires different representations of both the application problem and the hardware as well as different sets of tools. To meet these needs, we can use techniques such as mini-applications (e.g. Mantevo, Section 2.1), Abstract Machine Models (AMMs) (Section 5.1), and Simulation frameworks, such as the Structural Simulation Toolkit (SST) (Section 5.2).

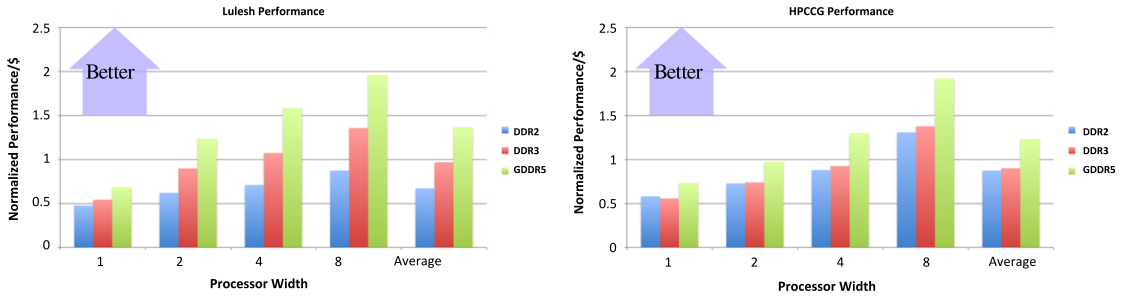


Fig. 10. Application performance with different memory systems.

5.1. Abstract machine models

An Abstract Machine Model (AMM) is a simplified description of a computer system that allows reasoning about that system.

Such a model may take many forms. It may be standardized and machine parsable or may be an informal description for users of a system. There may be multiple models for a machine at multiple resolutions and multiple levels of abstraction. These models may have details on the latencies and bandwidths of different system components or other characteristics such as the power and energy cost of different operations or reliability properties like failure modes and frequencies or soft error rates.

Some existing examples of Abstract Machine models:

- *Compiler machine models*: GCC and other multi-platform compilers will often include a machine model that describes the processor core, including internal latencies and instruction support.
- *Tool models*: The input files to commonly used simulators or design tools such as Simplescalar or McPAT represent a type of machine model. These models are often very specific and are used as the input to tools like performance analyzers, adaptive runtimes, compilers, and simulators.
- *Analytical models*: Abstract mathematical models such as the PRAM algorithmic complexity or LogP network performance model provide very simple, but potentially useful tools to reason about and estimate performance. Educational or theoretical models like the von Neumann architecture, Knuth MIX, or Patterson DLX also fall in this category. Usually the system is characterized by a small number of parameters, allowing simple and easy analysis.
- *Detailed*: ISA and Principles of Operation manuals can provide very detailed descriptions of exactly how a machine works and its capabilities.

Many of these models are focused on only one part of the system (e.g. a compiler machine model only addresses the processor), and may neglect other parts of the system (memory, network, IO).

Like any model, it is difficult to produce a complete AMM before you start reasoning about the system. Instead, it is created and refined throughout the design process. The ability to evolve a machine model is critical, especially when performing design space exploration for future machines because the co-design process is filled with moving targets like the applications, architectures, fabrication processes, etc.

5.2. Structural simulation toolkit

Future Exascale Systems must incorporate substantial advances in execution model, system architecture, processor architecture, interconnect, memory, I/O and system software. Coherent exploration of this design space will require a common simulation platform to enable the multidimensional hardware/software co-design required by a project of this scope. To meet these requirements, a number of institutions have been developing the Structural Simulation Toolkit (SST [29,30]) as a common platform for co-design. The SST provides a scalable parallel architectural simulator for simulating the huge numbers of architectural components in an exascale system. The SST contains interfaces to a variety of technology models for estimating power and other system characteristics. It is constructed in a modular fashion and contains a mix of abstract and detailed models for processors, memory systems, and interconnection networks [31–35].

Many other architectural simulators, such as M5 [36], NS-3 [37], and A-SIM [38], are in widespread use for designing system components. In addition, there is a long history of packages to model power dissipation [39,40]. The novel approach of SST is to include several individual component models in a parallel, scalable, and open-source framework.

5.2.1. Example design space exploration

Simulators like the SST, when combined with miniapps, allow fast and efficient design space exploration. This process can be used to make decisions such as which memory technology and processor core is best suited to a given problem. One such experiment looked at the impact of different memory technologies (DDR2, DDR3, and GDDR5), and processor core issue widths (1, 2, 4, 8 wide issue) on the power, performance, and cost of the HPCCG and Lulesh [41] miniapps.

For this exploration, the SST simulator was used with the GeM5/x86 processor model and DRAMSim2 memory model. Power estimates were done with DRAMSim2's internal models and McPAT for the processor. Cost estimates were done using the IC Knowledge [42] chip cost estimation program and memory costs from the DRAM Spot Price Index (www.dramexchange.com).

The memory system options compared DDR2 (cheap, low power, but antiquated performance), DDR3 (higher performance, reasonable power), and GDDR5 (expensive, high power, very high bandwidth).

Based solely on performance (Fig. 10), GDDR5 provides a much better solution. Across a range of processor core widths, GDDR5 was 26%–47% faster than DDR3 (on Lulesh) and 32%–41% faster than DDR3 (on HPCCG). Examined only from this perspective, GDDR5 would be the clear winner.

However, though GDDR provides better raw performance, DDR3 generally has better performance per Watt. On both miniapps DDR3's performance per Watt is roughly equal to GDDR5 for wide processor cores and up to 107% higher for narrow processors (Fig. 11).

Performance per Dollar was also more varied. For Lulesh, DDR3 was better then GDDR5 for narrow cores (1–2 wide), roughly the same at 4-wide and worse for 8-wide. For HPCCG, DDR3 was better for 1–4 wide cores, but lost out for 8-wide (see Fig. 12).

Similar effects can be seen with processor issue width. Wider processor cores can issue more instructions/cycle and always perform better than narrow cores. However, wide cores consume

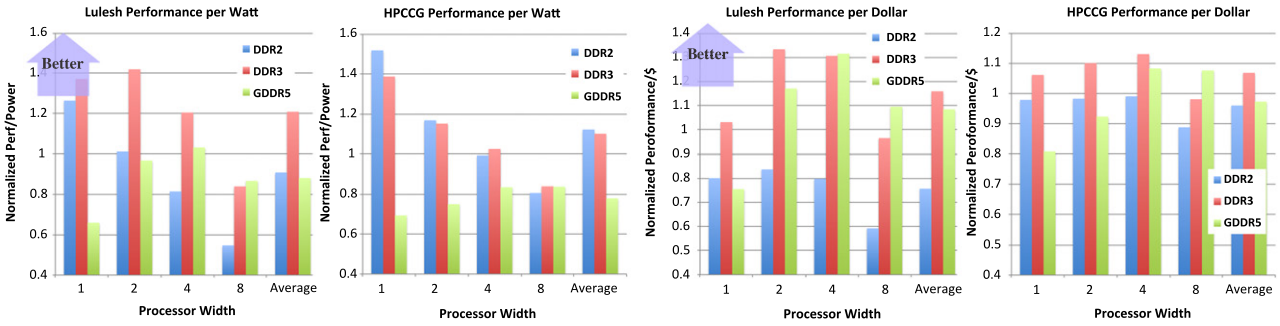


Fig. 11. Power and cost with different memory systems.

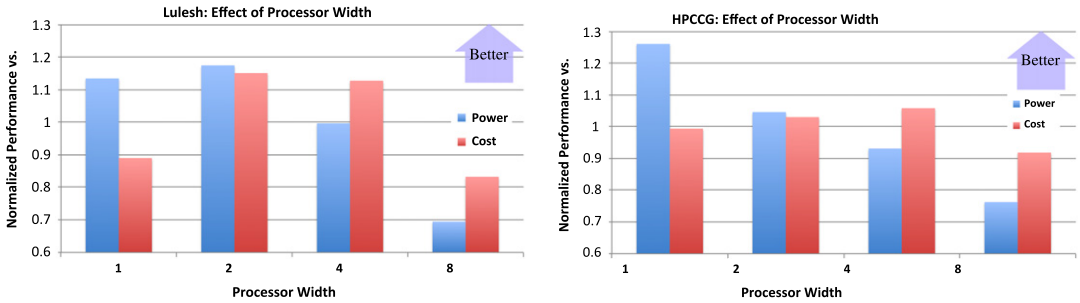


Fig. 12. Cost and power efficiency for different processor issue widths.

more area and power. Often the increase in performance is linear or sub-linear, while the increase in cost and area is super-linear. This is because many processor structures scale super-linearly with the processor's issue width. For example, register file energy per access and area scales at roughly $O(w^{1.8})$ [43]. Chip cost is directly related to area because as chip area increases the number of chips that can fit on a wafer decreases. Also, because of the distribution of defects on a wafer, larger chips tend to have much lower manufacturing yields.

In our simulations, wider processors were indeed faster. On Lulesh, an 8-wide processor was 78% faster than a single-issue core. However, it also used 123% more power. In general, for both apps, 1–2 wide cores were the most power efficient and 2–4 wide cores were the most cost efficient.

So, while wider cores require a shorter time to reach a solution, they tend to require much more energy to reach a solution.

5.2.2. Role of simulation in co-design

These simple design space explorations show that the fastest memory technology is not always the best (DDR beats GDDR) due to power and cost concerns. Additionally, it is hard to declare a single "best" processor because of the complex tradeoffs between cost, performance, and power.

Simulations can provide a better understanding of which configurations are best for a given application and can be used as a basis for application optimization and vendor guidance.

6. Conclusions

A methodology for comparing a mini-application and a full application code was postulated. It was used to study differences between a finite-element miniapp and a device physics code. The miniapp did remarkably well on sensitivity to memory bandwidth; there was greater variation between the application code and the mini-application on cache behavior.

A variety of advanced architecture testbeds are being used to study performance issues of key algorithms. These studies are helping guide algorithms research and are providing useful feedback to computer architects. Excellent performance was obtained for a very challenging finite-element mini-application on a Nvidia GPU. Register spilling was identified as a key issue that must, in the future, be overcome through architecture and system software enhancements, as well as through algorithms research.

Network bandwidth degradation studies on supercomputers are helping define network requirements for future systems. These studies are also helping identify methods for improving application performance.

And finally, validated hardware/software co-simulation tools are needed to enable co-design. One such multi-fidelity software tool, the Structural Simulation Toolkit, was described in this paper. Initial validation results are encouraging. Several architecture studies using SST were also discussed.

Acknowledgments

Support for this work was provided through the Advanced Simulation and Computing (ASC) program funded by US Department of Energy's National Nuclear Security Agency, and through the Scientific Discovery through Advanced Computing (SciDAC) program funded by the US Department of Energy's Office of Advanced Scientific Computing Research.

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the US Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[1] R. Stevens, A. White, S. Dosanjh, et al., Scientific grand challenges: architectures and technology for extreme-scale computing report, Technical Report, 2011.

[2] ITRS International Roadmap Committee, International Technology Roadmap for Semiconductors, 2011.

[3] J. Ang, et al., Exascale computing and the role of co-design, in: High Performance Computing: From Grids and Clouds to Exascale, IOS PressInc., 2011, pp. 43–64.
[4] M. Heroux, D. Doerfler, P. Crozier, J. Willenbring, H. Edwards, A. Williams, M. Rajan, E. Keiter, H. Thornquist, R. Numrich, Improving performance via mini-applications, Technical Report SAND2009-5574, Sandia National Laboratories, 2009.
[5] A. Geist, S. Dosanjh, IESP exascale challenge: co-design of architectures and algorithms, International Journal of High Performance Computing Applications 23 (2009) 401–402.
[6] R. Barrett, P. Crozier, S. Hammond, M. Heroux, P. Lin, T. Trucano, C. Vaughan, Assessing the validity of the role of mini-applications in predicting key performance characteristics of scientific and engineering applications, Technical Report SAND2012-4667, Sandia National Laboratories, 2012 (in preparation).
[7] M.A. Heroux, R. Bartlett, V. Howle, R. Hoekstra, J. Hu, T. Kolda, R. Lehoucq, K. Long, R. Pawlowski, E. Phipps, A. Salinger, H. Thornquist, R. Tuminaro, J. Willenbring, A. Williams, K. Stanley, An overview of the Trilinos project, ACM Transactions on Mathematical Software 31 (2005) 397–423.
[8] GNU Lesser General Public License—GNU Project, 2009. http://www.gnu.org/licenses/lgpl.html.
[9] W.L. Oberkampf, T.G. Trucano, Verification and validation in computational fluid dynamics, Progress in Aerospace Sciences 38 (2002).
[10] W.L. Oberkampf, C.J. Roy, Verification and Validation in Scientific Computing, Cambridge University Press, 2010.
[11] M. Pilch, T. Trucano, J. Moya, G. Froehlich, A. Hodges, D. Peercy, Guidelines for Sandia ASCI verification and validation plans—content and format: version 2.0, Technical Report SAND2000-3101, Sandia National Laboratories, 2000.
[12] T.G. Trucano, M. Pilch, W. Oberkampf, General concepts for experimental validation of ASCI code applications, Technical Report SAND2002-0341, Sandia National Laboratories, 2002.
[13] G.L. Hennigan, R.J. Hoekstra, J.P. Castro, D.A. Fixel, J.N. Shadid, Simulation of neutron radiation damage in silicon semiconductor devices, Technical Report SAND2007-7157, Sandia National Laboratories, 2007.
[14] P.T. Lin, J.N. Shadid, M. Sala, R.S. Tuminaro, G.L. Hennigan, R.J. Hoekstra, Performance of a parallel algebraic multilevel preconditioner for stabilized finite element semiconductor device modeling, Journal of Computational Physics 228 (2009) 6250–6267.
[15] H. van der Vorst, Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems, SIAM Journal on Scientific and Statistical Computing 13 (1992) 631–644.
[16] Y. Saad, M.H. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM Journal on Scientific and Statistical Computing 7 (1986) 856–869.
[17] R.S. Tuminaro, M. Heroux, S.A. Hutchinson, J.N. Shadid, Official Aztec user's guide-version 2.1, Technical Report SAND99-8801J, Sandia National Laboratories, Albuquerque NM, 87185, 1999.
[18] M. Hestenes, E. Stiefel, Methods of conjugate gradients for solving linear systems, Journal of Research. National Bureau of Standards 49 (1952) 409–436.
[19] J. Dongarra, D. Gannon, G. Fox, K. Kennedy, The impact of multicore on computational science software, CTWatchQuarterly 3 (2007).
[20] P.T. Lin, J.N. Shadid, Towards large-scale multi-socket, multicore parallel simulations: performance of an MPI-only semiconductor device simulator, Journal of Computational Physics 229 (2010) 6804–6818.
[21] S. Alam, R. Barrett, J. Kuehn, P. Roth, J. Vetter, Characterization of scientific workloads on systems with multi-core processors, in: IEEE International Symposium on Workload Characterization.
[22] M. Gee, C. Siefert, J. Hu, R. Tuminaro, M. Sala, ML 5.0 smoothed aggregation user's guide, Technical Report SAND2006-2649, Sandia National Laboratories, 2006.
[23] N. Bell, M. Garland, Implementing sparse matrix–vector multiplication on throughput-oriented processors, in: SC'09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, ACM, New York, NY, USA, 2009, pp. 1–11.
[24] K.B. Ferreira, P. Bridges, R. Brightwell, Characterizing application sensitivity to OS interference using kernel-level noise injection, in: SC'08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing, IEEE Press, Piscataway, NJ, USA, 2008, pp. 1–12.
[25] K. Pedretti, R. Brightwell, D. Doerfler, K.S. Hemmert, J. Laros, The impact of injection bandwidth performance on application scalability, in: Recent Advances in the Message Passing Interface, in: Lecture Notes in Computer Science, vol. 6960, 2011, pp. 237–246.
[26] P.M. Kogge, et al., Exascale computing study: technology challenges in achieving exascale systems, Technical Report, University of Notre Dame CSE Department Technical Report, TR-2008-13, September 28, 2008.
[27] K. Bergman, G. Hendry, P. Hargrove, J. Shalf, B. Jacob, K. Hemmert, A. Rodrigues, D. Resnick, Let there be light!: the future of memory systems is photonics and 3D stacking, in: Proceedings of the 2011 ACM SIGPLAN Workshop on Memory Systems Performance and Correctness, MSPC'11, ACM, New York, NY, USA, 2011, pp. 43–48.
[28] M. Pecht, R. Radojcic, G. Rao, Guidebook for Managing Silicon Chip Reliability, CRC press, 1999.
[29] A.F. Rodrigues, R.C. Murphy, P. Kogge, K.D. Underwood, The structural simulation toolkit: exploring novel architectures, in: SC'06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing, ACM, New York, NY, USA, 2006, p. 157.
[30] A. Rodrigues, et al., The structural simulation toolkit, SIGMETRICS Performance Evaluation Review 38 (2011).
[31] N.B. Lakshminarayana, H. Kim, Effect of instruction fetch and memory scheduling on GPU performance, in: Workshop on Language, Compiler, and Architecture Support for GPGPU, in conjunction with HPCA/PPoPP, 2010.
[32] D. Wang, et al., DRAMsim: a memory-system simulator, SIGARCH Computer Architecture News 33 (2005) 100–107.
[33] D. Burger, T. Austin, The SimpleScalar Tool Set, Version 2.0, SimpleScalar LLC, 2000.
[34] R. Srinivasan, J. Cook, O. Lubeck, Ultra-fast CPU performance prediction: extending the Monte Carlo approach, in: Proceedings of the IEEE International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD).
[35] G.H. Loh, S. Subramaniam, Y. Xie, Zesto: a cycle-level simulator for highly detailed microarchitecture exploration, in: Proc. of the Int. Symp. on Performance Analysis of Systems and Software.
[36] N.L. Binkert, R.G. Dreslinski, L.R. Hsu, K.T. Lim, A.G. Saidi, S.K. Reinhardt, The M5 simulator: modeling networked systems, IEEE Micro 26 (2006) 52–60.
[37] T. Henderson, T.R. Henderson, S. Roy, NS-3 project goals, 2006.
[38] D. Nellans, V.K. Kadaru, E. Brunv, Asim—an asynchronous architectural level simulator, 2004.
[39] D. Brooks, V. Tiwari, M. Martonosi, Wattch: a framework for architectural-level power analysis and optimizations, in: Proceedings of the 27th Annual International Symposium on Computer Architecture, pp. 83–94.
[40] D.T.S. Thoziyoor, D. Tarjan, S. Thoziyoor, Cacti 4.0, Technical Report, HP Labs, 2006.
[41] Lawrence Livermore national laboratory, Hydrodynamics challenge problem Lawrence Livermore national laboratory, Technical Report LLNL-TR-490254, Lawrence Livermore National Laboratory, 2010.
[42] ICKnowledge LLC, Ic cost model 0909b, Software, 2009.
[43] V. Zyuban, Inherently lower-power high-performance superscalar architectures, Ph.D. Thesis, University of Notre Dame, 2000.

S.S. Dosanjh is Director of the National Energy Research Scientific Computing (NERSC) Center at Lawrence Berkeley National Laboratory. NERSC's mission is to accelerate scientific discovery at the US Department of Energy's Office of Science through high performance computing and extreme data analysis. NERSC deploys leading-edge computational and data resources for over 4500 users from a broad range of disciplines. NERSC will be partnering with computer companies to develop and deploy pre-exascale and exascale systems during the next decade. Previously, Dr. Dosanjh headed extreme-scale computing at Sandia National Laboratories. He was co-director of the Los Alamos/Sandia Alliance for Computing at the Extreme-Scale from 2008–2012. He also served on the US Department of Energy's Exascale Initiative Steering Committee for several years. Dr. Dosanjh had a key role in establishing co-design as a methodology for reaching exascale computing. He has numerous publications on exascale computing, co-design, computer architectures, massively parallel computing and computational science. He earned his bachelor's degree in Engineering Physics in 1982, his master's degree (1984) and Ph.D. (1986) in Mechanical Engineering, all from the University of California, Berkeley.

R.F. Barrett is a Principal Member of the Technical Staff in the Extreme-scale computing group at Sandia National Laboratories. As a Member of the Scalable Architectures department in the Center for Computing Research, he leads the Application Performance Modeling and Analysis Team (PMAT), whose goals are to understand and characterize application performance on key HPC platforms that are currently deployed, and to predict performance on future platforms using mathematical modeling methods and techniques and an empirical knowledge base.

D.W. Doerfler is a Distinguished Member of Technical Staff at Sandia National Laboratories. Doug is the ACES Cielo Architect and his research interests include high-performance computer architectures and performance analysis.

S.D. Hammond is a Member of the Scalable Computer Architectures group at Sandia National Laboratories. His research focuses on the optimization and porting of applications for advanced computer architectures. He is also a Member of the performance analysis and computer science teams for the Materials and Combustion Office of Science Exascale Co-Design centers.

K.S. Hemmert is a Principal Member of the Technical Staff in the Extreme-scale computing group at Sandia National Laboratories.

M.A. Heroux is a Distinguished Member of the Technical Staff at Sandia National Laboratories, working on new algorithm development, and robust parallel implementation of solver components for problems of interest to Sandia and the broader scientific and engineering community. He leads development of the Trilinos Project, an effort to provide state of the art solution methods in a state of the art software framework. Trilinos is a 2004 R&D 100 award-winning product, freely available as Open Source and actively developed by dozens of researchers. In addition to Trilinos, Dr. Heroux works on the development of scalable parallel scientific and engineering applications and maintains his interest in the interaction of scientific/engineering applications and high performance computer architectures. He leads the Mantevo Project, which is focused on the development of Open Source, portable mini-applications and mini-drivers for scientific and engineering applications. Dr. Heroux is a telecommuter for Sandia, maintaining an office at home in rural central Minnesota and at St. John's University where he is Scientist in Residence in the Computer Science Department. He is a Member of the Society for Industrial and Applied Mathematics (SIAM) and past chair of the SIAM Activity Group on Supercomputing. He is a Distinguished Member of the Association for Computing Machinery (ACM). He is the Editor-in-Chief for the ACM Transactions on Mathematical Software, Subject Area Editor for the Journal on Parallel and Distributed Computing and Associate Editor for the SIAM Journal on Scientific Computing.

P.T. Lin received his Ph.D. in Mechanical and Aerospace Engineering from Princeton University. He is currently a Principal Member of the Technical Staff at Sandia National Laboratories. His research interests include high performance computing, scalable linear solvers, preconditioners, multigrid methods, and algorithms for multicore processors.

K.T. Pedretti is a senior Principal Member of the Technical Staff at Sandia National Laboratories. His research focuses on operating systems for massively parallel supercomputers, techniques for improving resilience to hardware faults, high-performance networking, and software-directed power management strategies. Pedretti has a B.S. and an M.S. in Electrical and Computer Engineering from the University of Iowa.

A.F. Rodrigues is a Principal Member of the Technical Staff in the Extreme-scale computing group at Sandia National Laboratories.

T.G. Trucano is currently a Senior Scientist in the technical staff at Sandia National Laboratories in the Optimization and Uncertainty Estimation Department. He received his Ph.D. in Mathematical Physics from the University of New Mexico in 1980. He began work at Sandia in August of that year. For the first 15 years of his career at Sandia, his focused on research, development, and applications of computational shock wave physics. Over this period of time Trucano worked on problems of shock hydrodynamics, equation of state and constitutive modeling, hypervelocity impact phenomena, nuclear weapons effects, and radiation-hydrodynamics in high energy density physics applications. Beginning in 1995, his work began to focus on issues of computational science verification and validation (V&V), as well as the application of uncertainty quantification. Currently, he works on technical and programmatic challenges for the NNSA Advanced Simulation and Computing (ASC) V&V Program, as well as research in the application of uncertainty quantification in computational prediction and the accompanying decision environment. For the past five years, Trucano has also been working on the role of V&V and uncertainty quantification in the application of complex social models to national security problems.

J.P. Luitjens received a Ph.D. in Scientific Computing from the University of Utah in 2011. He is now a Member of the developer technologies team at NVIDIA Corporation where he focuses on parallel algorithm development and optimization.