Transition of HPC Towards Exascale Computing
E.H. D'Hollander et al. (Eds.)
IOS Press, 2013
© 2013 The authors and IOS Press. All rights reserved.
doi:10.3233/978-1-61499-324-7-141

On the Role of Co-design in High Performance Computing

Richard F. BARRETT [a,1], Shekhar BORKAR[b], Sudip S. DOSANJH [c], Simon D. HAMMOND [a], Michael A. HEROUX [a], X. Sharon HU [d], Justin LUITJENS[e], Steven G. PARKER [e], John SHALF[c], and Li TANG[d]
[a] *Sandia National Laboratories, Albuquerque, NM, USA*
[b] *Intel Corporation*
[c] *Lawrence Berkeley National Laboratory, Berkeley, CA, USA*
[d] *University of Notre Dame, South Bend, IN, USA*
[e] *Nvidia, Inc., Santa Clara, CA, USA*

Abstract. Preparations for Exascale computing have led to the realization that future computing environments will be significantly different from those that provide Petascale capabilities. This change is driven by energy constraints, which is compelling architects to design systems that will require a significant re-thinking of how algorithms are developed and implemented. Co-design has been proposed as a methodology for scientific application, software and hardware communities to work together. This chapter gives an overview of co-design and discusses the early application of this methodology to High Performance Computing.

Keywords. Scientific applications; high performance computing; parallel architectures.

Introduction

Computational requirements for energy research, national security and advanced science will require a thousand-fold increase in supercomputing performance during the next decade [1]. However, the transition to Exascale high-performance computing (HPC) systems that are operable within affordable power budgets will not be possible based solely on existing computer industry roadmaps [2]. The HPC community must make investments to develop new computing technologies and to accelerate industry roadmaps. Without such investments, future computing platforms will not broadly meet the needs of HPC. One example of such an investment is the U.S. Department of Energy's Fast Forward program [3,4], which is funding innovative research in processor and memory technologies. The goal is to dramatically improve energy efficiency while maintaining or improving reliability, runtimes and programmability. AMD, IBM, Intel and Nvidia are all being funded under Fast Forward.

A key strategy for Fast Forward and Exascale computing is co-design, which has received considerable attention in the HPC community for several years [5]. Because architectures are changing dramatically, software needs to adapt as well. The

[1] Corresponding Author: Center for Computing Research, Sandia National Laboratories, Albuquerque, New Mexico 87185-1319; E-mail:rfbarre@sandia.gov

"codesign" principle may offer a common basis for application and software developers to collaborate with computer architects. This principle has been developed and employed extensively by the embedded computing community. In embedded applications, systems are developed through methodical hardware/software co-simulation, co-verification, and co-optimization. Various tradeoffs are considered with the goal of minimizing, for example, cost and power.

Applying co-design to Exascale computing presents challenges and opportunities. HPC applications are very complex and often consist of millions of lines of code. HPC systems are also much more complex than embedded systems. Because it is currently not tractable to simulate a complete application executing on Exascale hardware, various approximations must be used. Two key approximations are the use of mini-applications and proxy architectures.

Mini-applications attempt to capture a key aspect of the computation in a few thousand lines of code. Mini-applications are being developed by the U.S. Department of Energy Laboratories to represent important energy, national security and science applications. Proxy architectures similarly try to capture the key aspects of the hardware that application developers need to consider when designing new algorithms. The goal is to make these proxy architectures sufficiently abstract to avoid intellectual property issues but with enough detail to still make them useful for performing hardware and algorithm tradeoff analyses.

Co-design is a tremendous opportunity for fostering collaborations between application and software developers and hardware architects. It will impact all scales of computing from desktops to racks to supercomputers because all these scales face similar challenges in energy efficiency, concurrency, data movement and programmability. For the users of Exascale machines, there will be additional challenges including scalability and reliability that are brought about by the extreme size of such systems.

This chapter presents progress on HPC co-design, including new work on mini-applications, proxy architectures and simulation/modeling. Early co-design results from two Fast Forward projects are also discussed.

1. An Overview of Codesign

HPC performance has improved by three orders of magnitude almost every decade, made possible by technology scaling and advances in the system design (hardware and software) following an evolutionary approach. Next generation hardware was delivered to software developers who ported and optimized the software stack to the delivered hardware. Exascale will be different, the entire system architecture will be vastly different, and thus to realize the Exascale performance goal a revolutionary approach will be required with strong software and hardware co-design strategy at its center.

As shown in Figure 1, co-design considers the entire system stack from underlying technologies to applications. Applications provide insight into compute patterns and data movement patterns to optimize the system, the execution model implemented in the system software is tailored to provide services and manage resources, and the programming system hides the underlying hardware and provides programming productivity. From the bottom, technological issues and opportunities need to be comprehended, and in the middle, the system architecture effort needs to devise the most optimal system architecture considering top down requirements and bottom up issues and opportunities.

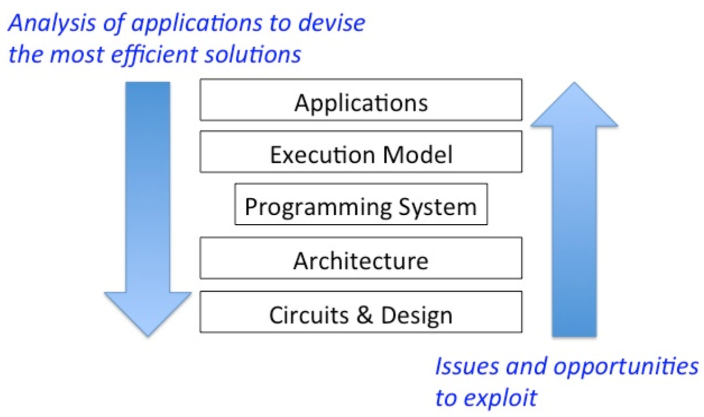
Analysis of applications to devise the most efficient solutions
Applications
Execution Model
Programming System
Architecture
Circuits & Design
Issues and opportunities to exploit

Figure 1 Co-design strategy

Conventional HPC system design involves a pipelined collaboration process that spans requirements gathering at the start of the design process and product evaluation when it is delivered 4-6 years later. However, the rapid and disruptive changes anticipated in hardware design over the next decade necessitate a more systematic and agile development process, such as the hardware-software co-design processes developed for rapid product development in the embedded space. Design methodologies on which we have relied so far never had to consider power constraints or parallelism of the scale being contemplated for Exascale systems. Furthermore, the programming model and software environment for future extreme-scale systems is anticipated to be substantially different from current practice. The designers of HPC hardware and software components have an urgent need for a systematic design methodology that reflects future design concerns and constraints.

The co-design strategy is based on developing partnerships with computer vendors and application scientists and engaging them in a highly collaborative and iterative design process well before a given system is available for commercial use. The process is built around identifying leading edge, high-impact scientific applications and providing concrete optimization targets rather than focusing on speeds and feeds (FLOPs and bandwidth) and percent of peak. Rather than asking "what kind of scientific applications can run on an Exascale system" after it arrives, this application-driven design process instead asks "what kind of system should be built to meet the needs of the most important science problems." This leverages deep understanding of specific application requirements and a broad-based computational science portfolio. Focusing on delivered scientific application performance in the design process is essential to define a common optimization target that spans hardware and applications.

Target application programs often consist of a million source lines of code involving multiple programming languages, third party library dependencies, and other complexities. To address this complexity, we are considering a software model that includes application proxies, providing a tractable means for exploring key issues associated with design goals. Different types of proxies are required to implement different aspects of the co-design process. Table 1 shows several proxies and their definitions. Figure 2 depicts the interplay between the hierarchies of reduced applications in code analysis for co-design.

Table 1 Application proxies and their definition

Surrogate	Description
Compact app.	Small app. having fewer features and simplified boundary conditions relative to full apps.
Mini-app	Small, self-contained program that embodies essential performance characteristics of key applications.
Skeleton app	Captures control flow and communication pattern of app. Can only be run in a simulator.
Mini-driver	Small programs that act as drivers of performance-impacting library packages.
Kernel	Captures node-level aspects of an algorithm

Modeling, simulation, and compiler analysis all play synergistic roles in the co-design process to cover a broad space of design parameters. Figure 3 shows that a multi-resolution approach using multiple modeling methodologies should be employed to cover both the scale of Exascale systems and the fidelity required to have confidence in our design choices. Tools such as cycle accurate hardware simulation (the parallelogram in Figure 3) offer extreme detail in their modeling capability, but limit the scale of systems that can be modeled to node size or small clusters. Constitutive models (the triangle in Figure 3) and other empirical modeling methods can cover much larger systems, but by definition only model effects included as parameters in the model. The majority of modeling is done by empirical models because they are faster to

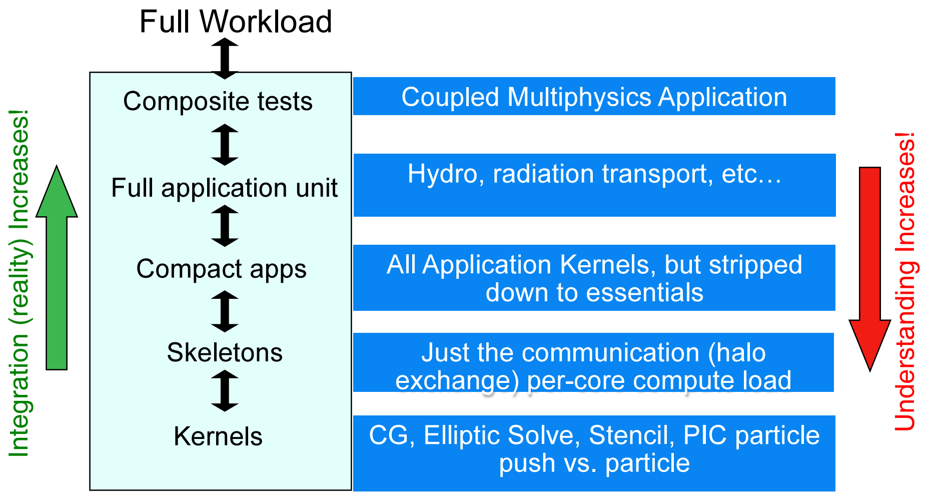
Full Workload
Composite tests
Full application unit
Compact apps
Skeletons
Kernels
Coupled Multiphysics Application
Hydro, radiation transport, etc…
All Application Kernels, but stripped down to essentials
Just the communication (halo exchange) per-core compute load
CG, Elliptic Solve, Stencil, PIC particle push vs. particle
Integration (reality) Increases!
Understanding Increases!

Figure 2 Depiction of the interplay between the hierarchy of reduced applications that we employ in the co-design process. Kernels enable rapid exploration of new languages and algorithms because of ease of rewriting. Full applications are harder to rewrite, but ensure adherence to original application requirements.

construct and evaluate, and cycle-accurate models are used to verify that the simpler model has included all important effects, and has not neglected anything any essential (but unanticipated) effects.

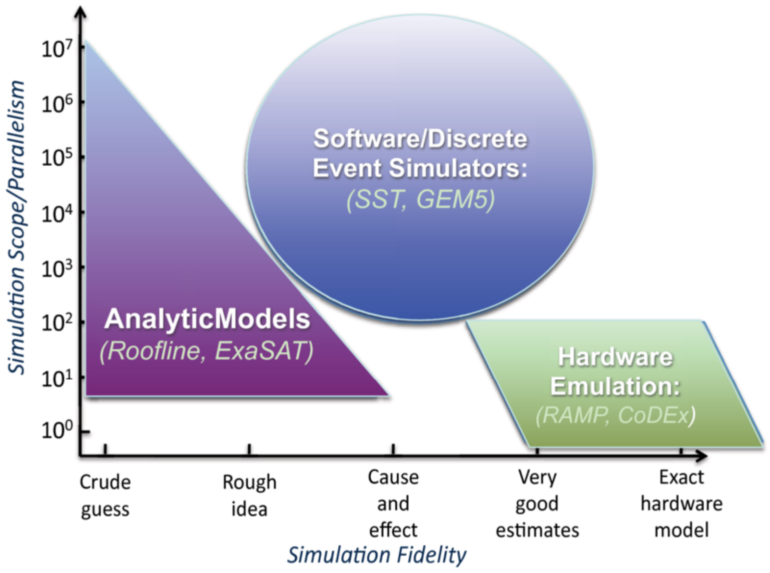

Figure 3 Multiple modeling approaches are required to cover both the scale and accuracy required to understand system design trade-offs. The two key axes for simulation and modeling techniques are fidelity of the model (horizontal axis) and the scale of system you can simulate.

Architecture simulation [9,12,14] provides an avenue to experiment with hardware configurations, programming models, and algorithms on Exascale class machines before full implementations of the hardware and software components of a system are available. Models of the hardware, the runtime system, and the application can be integrated into to the simulation to evaluate the trade-offs between design choices imposed on each aspect of the integrated system. Coarse-grained simulations (the circle in Figure 3) allow large-scale systems to be studied in a way that captures the complex interactions between various hardware components, including those interactions that only arise at the largest scales.

The co-design process involving a collaboration between an application development, code analysis, and architectural simulation has been demonstrated for applications ranging from Climate Modeling (Green Flash) [8], Seismic Imaging (Green Wave) [13], and an automated co-tuning process that includes both auto-tuning of hardware and software in the same process [16].

2. Design Space Exploration and Proxy Architectures

Given the complexity of constructing large HPC 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 can be used to rapidly assess and optimize designs as they are created. A systematic way of supporting such a dialog is through a formal design space exploration process. 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 attention to creating solutions that work across a broad range of applications and do not benefit only a single problem.

Our methodology for Exascale design space exploration [7] includes measurement on prototype hardware, experimentation in the form of re-factoring and re-implementing using a variety of programming models and algorithms, and prediction using architectural simulators. To this end, we are investigating several architectural testbeds that 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.

A fundamental hurdle to design space exploration is the scale of modern scientific applications, which limits rapid prototype assessment. Applications are typically millions of lines of source and are written to use complex algorithms and data structures. While our eventual goal is to run applications on 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 miniapp is a condensed implementation of one or multiple key performance issues that affect parent codes, written to be amenable to re-factoring or change but representative enough to be useful in the scientific problem domain.

In order to convey confidence in our use of application proxies, we have developed a validation methodology [6] for demonstrating the applicability of miniapps to their parent codes enabling HPC vendors and researchers to have a high degree of confidence in results obtained from studies using miniapps. This methodology is guided by the principles developed for experimental validation [17], where the application is analogous to the physical observation and the miniapp is the experiment. A more detailed discussion on miniapps will be given in the next section. A similar effort is being applied to hard-ware/software co-simulation tools, and initial validation results of are encouraging.

The ability to predict the performance, and more importantly the performance limitations, of hardware that 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 (AMM) or proxy architecture which defines the key architectural building blocks but no specific detail. We then are able to augment proxy architectures with details provided by performance models, architectural simulators and information obtained from our miniapps running on test-bed platforms to inform us of performance trade offs and available design decisions.

Some existing examples of architecture proxies:

- **Compiler Machine Models**: GCC and other multiplatform 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 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).

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. For example, excellent performance was obtained for a very challenging finite-element miniapp 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. (See Section 4 for more detail on this example.)

Simulators like the Structural Simulation Toolkit (SST) [5], when combined with application proxies, 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. For example, 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 a sparse linear system solver captured within the Mantevo HPCCG miniapp as well as LULESH[2], a proxy for a challenging shock hydrodynamics code.

3. Mini Applications

To aid in the rapid exploration of the new design space, we have developed a growing collection of application proxies. These enable rapid exploration of issues related to performance, programmability and system design trade-offs in a rapid and agile cycle. For example, "mini-applications", or miniapps, are small stand-alone programs that embody some key performance-impacting elements of the large-scale applications of interests. A set of miniapps identified for use across DOE is shown in Table 2.

Table 2. Initial Set of DOE Miniapps

Miniapp	Physics
CoMD	Molecular Dynamics
Lulesh	Hydrodynamics
MiniFE	Mechanics
Nek 5000	Spectral Hydrodynamics
PARTISN	Radiation Transport
SMC	Combustion

[2] https://computation.llnl.gov/casc/ShockHydro/

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. Miniapps create a meaningful context in which to explore the key performance issue, are developed and owned by application code teams, are limited to O(1K) source lines of code, and are intended to be modified with the only constraint being continued relevance to parent applications. We distinguish this from a compact-application whose purpose is to replicate a complex domain-specific behavior being used in a parent application. 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. Miniapps are also distinct from skeleton applications, which are typically designed to focus on inter-process communication, often producing a "fake" computation.

Miniapps distill from large-scale applications the most important design choices that must be made, and their refactored versions serve as examples of how the large-scale applications can be rewritten [6]. From the Mantevo project [10], for example, a finite-element miniapp, miniFE, tracked the sensitivity to memory bandwidth of Charon, an electronics device simulation application [15]; a molecular dynamics miniapp illustrated the computation of a Lennard-Jones force calculation in LAMMPS [18]; and a finite difference miniapp illuminated an inter-process communication scaling problem, seen only at very large scale, for a multi-material Eulerian code [11]. Just as important, this methodology also illustrates areas in which these miniapps did not adequately represent the full application.

Furthermore, miniapps provide system designers with concrete examples of how applications are written, so that design choices can be informed in a very tangible way by application needs. Finally, miniapps provide a concrete forum for detailed and precise communication between developers of all components of the computing platform: applications, libraries, compilers, runtime, OS and hardware. Use of miniapps has substantially increased the effectiveness of the combined efforts of these teams.

4. Early co-design results

In this section, we discuss some results from our early efforts in codesign.

4.1. An implicit finite element method code on GPUs

MiniFE is an implicit finite element based solver from the Mantevo suite. The focus was on node-level design, in particular the interplay between algorithm development and architecture exploitation. The specific application shows sensitivity to memory bandwidth, but rather than just asking for bandwidth, we sought to understand why it is sensitive to bandwidth and to explore other options for improving performance.
We focused on key performance limiters in the matrix assembly phase of the algorithm on an NVIDIA GPU using the CUDA programming model. A straightforward CUDA port was created that uses a single thread per element that adds a local element contribution into a global matrix. The computation of the element operator involves a number of floating-point heavy operations including computing the matrix determinant and the Jacobian. The large number of floating point operations suggest that the performance should be FLOP limited, but analysis using NVIDIA's Visual Profiler (NVVP) tool shows that the performance is actually bandwidth bound due to register

spilling. NVVP runs the application multiple times, collecting data from performance counters, from which it computes performance metrics. Register spilling can be identified by looking at the "Local Memory Overhead" metric. This metric indicates the percentage of bus traffic due to local memory accesses (register spilling). For miniFE this percentage is as high as 90%.

The cause of this register spilling was identified as the element operator which requires a large thread state - 704 bytes total for node-IDs, node coordinates, the diffusion matrix, and the source vector, plus additional for temporary data and pointers. The FermiGPU architecture supports up to 63 32-byte registers per thread, limiting the total register storage to 252 bytes. As a result of this limit, any additional state must be spilled to L1 and L2 caches but with 8192 simultaneous active threads, most of the spills are not contained by these caches. Consequently, registers are spilled to global memory causing the computation to become bandwidth bound.

As an alternative design, we tuned the kernel to reduce register usage, including algorithmic changes that exploit symmetry in the diffusion operator and reordered 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. While 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, improving the CPU performance as well.

The performance of the CUDA version of miniFE was compared to the MPI-parallel version of miniFE, with both versions running on a Tesla M2090 and a hex-core Intel Xeon 2.7GHz E5-2680. We tested for various problem sizes of N hexahedral elements. The speedup for each of the phases of the miniFE is reported in Figure 4. The data show that algorithm-architecture co-design provides different speedups for different phases of the program, and a comprehensive design space exploration is needed.

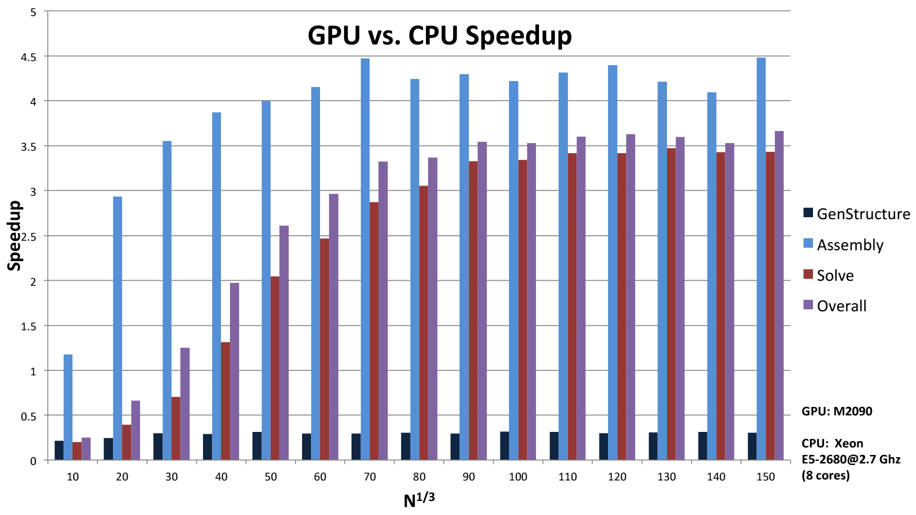

Figure 4. Speedup of CUDA version of miniFE over the MPI-parallel version of miniFE. Both implementations were run on a system consisting of a Tesla M2090 and a hex-core Intel Xeon 2.7GHz E5-2680.

Future generations of NVIDIA GPUs are expected to address some of the findings from the above example study, including an increased number of registers per thread and increases in the size of L1 and L2 caches. 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. Nonetheless, the application improvements are still likely to be beneficial. We expect that these investigations will also feed forward into subsequent GPU design cycles, thus completing the co-design feedback loop.

4.2. Design space exploration for Exascale processors

Co-design is being used by Intel for Exascale design space exploration. Design issues being considered include the use of scratchpads, developing a microarchitecture and programming model, virtualization and performance proportional bandwidth tapering.

4.2.1. Local Memory (scratchpad) vs Caches

The Exascale processor architecture operates at near threshold voltage of the transistor with reduced frequency for higher energy efficiency, thus the first level cache can be made larger since a larger cache would not impact the processor frequency. With the larger cache came the opportunity to replace the traditional caching scheme with a local memory (scratchpad) to save energy further—a suggestion that emerged bottom up. Normally you would not even think about displacing a cache with a scratchpad because it can have serious impact on programming productivity and application performance. However, detailed co-design activities with applications, programming system, and hardware designers revealed that this was a viable option—the applications strongly preferred explicitly managed local memory or scratchpad (see Figure 5).

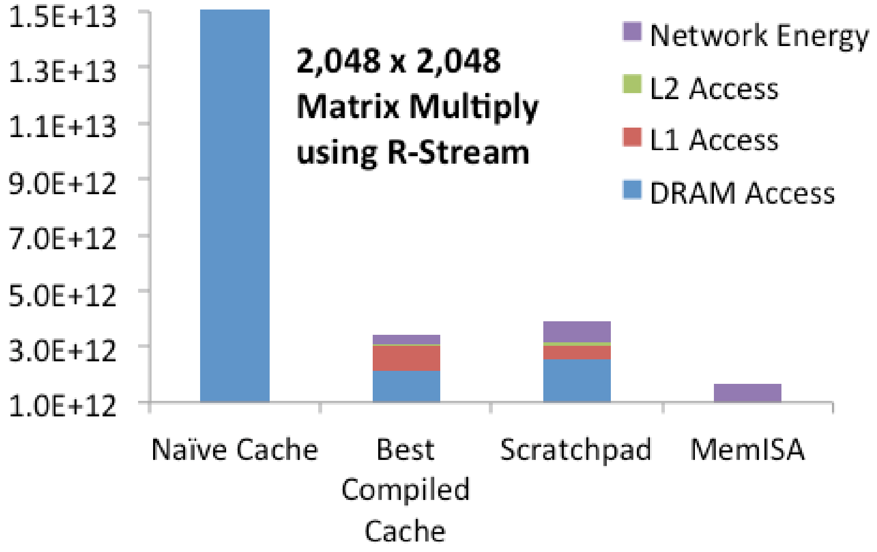

Figure 5: Benefits of scratchpads in a simple matrix multiply routine optimized using Reservoir Lab's R-Stream optimizer. Shown is the energy consumed in memory accesses using caches, scratchpads, and even optimized memory instructions. Scratchpads provide substantial benefits.

4.2.2. Microarchitecture and Programming Model

The microarchitecture comprehending separation of concerns was inspired by opportunities and issues bottom up. It is a collection of efficient execution engines (XE) optimized for the application and a control engine for system software and management—a heterogeneous architecture, now adopted by Intel's X-Stack project [19,20] (see Figure 6).

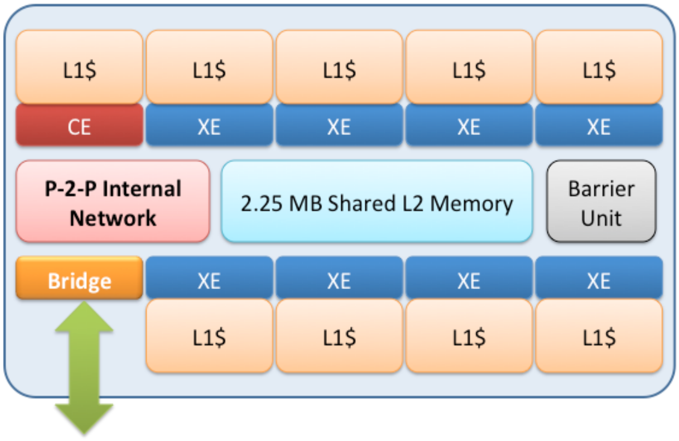

Figure 6: Heterogeneous microarchitecture.

Analysis of this architecture in the light of the applications pointed to the benefit of a codelet based programming and execution model—codelets are software components that can be dynamically scheduled without reliance on binding to any one thread or hardware component in relation to other codelets [21].

4.2.3. Virtualization

Co-design efforts have shown that removing memory space virtualization saved significant energy, yet had little to no impact to the application performance. Removing virtual memory saved significant energy by not having to do Translation Lookaside Buffer (TLB) lookups on each memory access. The exact amount of energy savings depends on details of the architecture. This successful co-design strategy will continue with the X-Stack and Fast Forward projects.

4.2.4. Performance Proportional Bandwidth Tapering

It is well understood that bandwidth per FLOP could reduce as one traverses higher hierarchy (tapering), to meet the Exascale power and energy goals. On the one hand, if the tapering is high and steep, then the energy reduction is substantial, but application performance could suffer. On the other hand, relaxed tapering could result in excessive energy consumption. Figure 7shows a starting point of naïve tapering conducive for applications, and an analysis showing excessive power consumption. At the lowest level, a Floating Point unit needs to read two operands and to write one operand (i.e., 24 bytes total). Then at each memory level we reduce the bandwidth by 4X as an experiment (see left graph in Figure 7). Corresponding data movement power is shown in the right graph in Figure 7 for a computational rate of 1 Exaflop/s. Power is calculated from energy costs for moving data (10 pJ/bit for optical, 1pJ/bit in a cabinet and on a board, and 0.03pJ/bit on a die).This result is application independent —that is, if an application demands BW at this naïve tapering level, then the power consumed in the system would be as high as 217 MW.

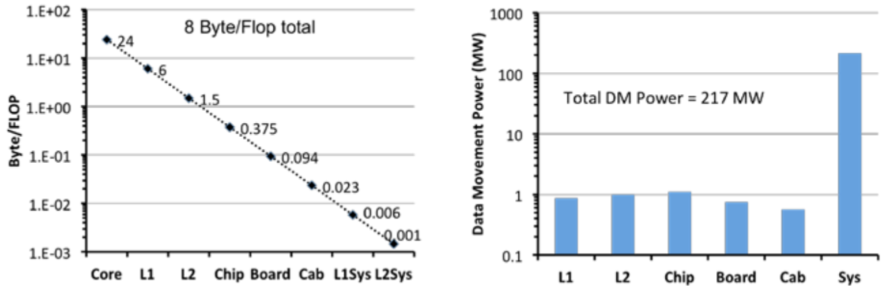

Figure 7: Power for data movement using naïve bandwidth tapering. The left graph shows the number of Bytes that have to be moved around the system for each floating point operation.

This co-design effort made it clear that we need to work together towards a compromise in tapering strategy, resulting in what we call performance proportional tapering, where tapering is increased with increased performance. The results of the analysis are shown in Figure 8, clearly showing that careful compromise using co-design is much needed for the Exascale system. In the past we assumed homogenous bandwidth between nodes, independent of the hierarchy, and expected constant bisection bandwidth. Now, we have come to grips with bandwidth tapering due to power, but we have often assumed naïve bandwidth tapering. This exercise shows that tapering has to be even more aggressive to make the system affordable in terms of power and energy, putting more burden on the programming system to enforce locality.

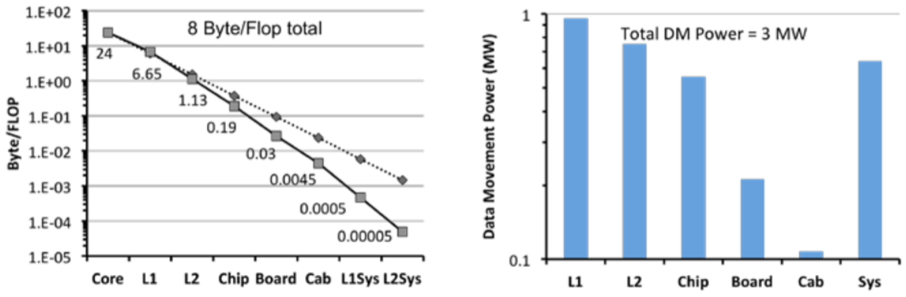

Figure 8: Power for data movement using modified bandwidth tapering.

4.3. Energy consumption

As discussed throughout this Chapter, for the first time constraints on energy consumption are critical drivers for our work. Toward this end our work has mainly focused on effective and efficient design of hardware with the goal of reducing runtime. As seen herein, this can have an important impact on power consumption.

From the application developer's perspective, a variety of issues should be considered. For example, what are the impacts of different architectures and their individual features (e.g. processor clock speed, memory speed)? Do different programming models impact performance? What role do compilers and their optimization strategies play? To begin to address these questions, we performed a set of experiments on a variety of platforms, including GPUs and ARM processors, examining the performance of the data assembly phase in finite element methods. The code, written in C++ and compiled with the Intel 13.0 compiler, operated on a 100 x 100 x 100 domain. Table 3 shows some results on an Intel i7 2600K 2.6 GHz Sandy Bridge processor. We measured the total energy required, in Joules, the average power used throughout execution in watts, and the peak power requirements, also in watts. For this processor, power was measured before the Voltage Regulator Module (VRM). Figure 9 graphically illustrates the measurement data normalized to the maximum for each category.

Table 3: Energy results for the miniFE finite element data assembly phase

	serial	*OpenMP*	*MPI*	*TBB*	*AVX*	*AVXM*
time (sec)	3.07	0.90	0.57	0.60	0.41	0.35
energy (Joules)	78.08	58.45	30.97	37.38	24.68	22.20
average watts	25.40	64.79	54.60	62.57	59.78	63.01
peak watts	51.53	69.21	67.16	65.62	62.66	64.55
Normalized to maximum values						
time	1	0.29	0.18	0.19	0.13	0.11
energy (Joules)	1	0.75	0.40	0.48	0.32	0.28
average watts	0.39	1	0.84	0.97	0.92	0.97
peak watts	0.74	1	0.97	0.95	0.91	0.93

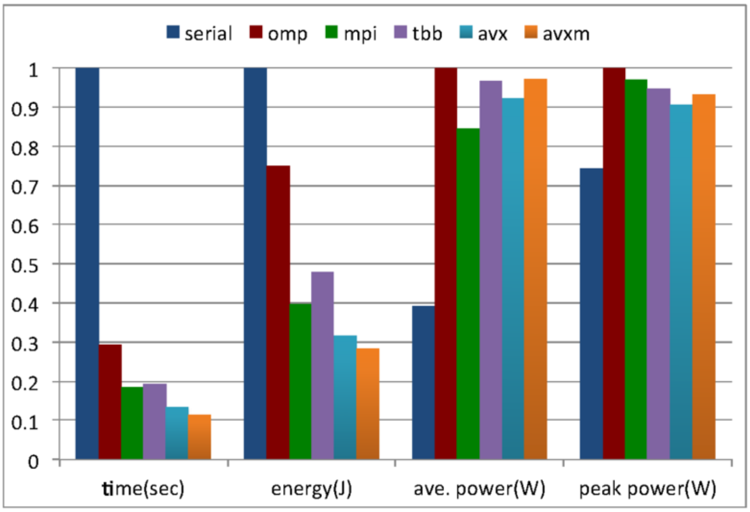

Figure 9: Normalized energy results for the miniFE finite element data assembly phase

Parallel execution (MPI and OpenMP) mapped one process to each of the four cores. The Intel Threaded Building Blocks (TBB) implementation used the maximum number of physical threads. Advanced Vector Extensions (AVX) SIMD intrinsics, added to the OpenMP implementation, fully vectorized the computation. The AVXM version added some GNU build-in primitives to the AVX version.

It can be seen from the data that the performance benefits of the parallelism in strong scaling mode, where special attention to the computation using vectorization capabilities results in super-linear speedups. While the execution time of the parallel versions ranges from 11-29% of serial execution, the associated total energy cost ranges from 28-75%. Not surprisingly, the most efficient parallel implementation, i.e., AVXM, results in the lowest energy consumption, though not the highest average or peak.

We temper these results with the understanding that the implementations of the miniapp, programming models, and their use of the underlying runtime systems are dynamic, and thus may be tunable to improve energy efficiency.

5. Summary

The accelerated push to Exascale, driven by mission requirements spanning a breadth of areas, has compelled the development of a new means to develop HPC environments for application scientists and engineers. The co-design approach, inspired by the embedded system design community, is providing the HPC community with an opportunity to strengthen its collaborative interactions internally and externally.

In this paper we have described various challenges and opportunities throughout the HPC codesign space, and have discussed the development effort for enabling this co-design practice. The results from this effort, partially shown by the data presented, demonstrate the potential that such co-design practice could have for taking us towards Exascale computing.

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] DOE Exascale Research Conference, http://exascaleresearch.labworks.org/oct2012/materials , Oct. 1-3, 2012.

[4] HPCWire. DOE Primes Pump for Exascale Supercomputers, http://www.hpcwire.com/hpcwire/2012-07-12/doe_primes_pump_for_exascale_supercomputers.html?page=1 ,2012.

[5] J. Ang et al. High Performance Computing: From Grids and Clouds to Exascale, chapter Exascale Computing and the Role of Co-design. IOS PressInc, 2011.

[6] Summary of Work for ASC L2 Milestone 4465: Characterize the Role of the Mini-Application in Predicting Key Performance Characteristics of Real Applications, R.F. Barrett (PI), P.S. Crozier, D.W. Doerfler (PM), S.D. Hammond, M.A. Heroux, H.K. Thornquist, T.G. Trucano, and C.T. Vaughan, Sandia Technical Report SAND 2012-4667, Sandia National Laboratories, 2012.

[7] R.F. Barrett, D.W. Doerfler, S.S. Dosanjh, S.D. Hammond, K.S. Hemmert, M.A. Heroux, P.T. Lin, J.P. Lutjiens, K.T. Pedretti, A.F. Rodrigues, , and T.G. Trucano. Exascale Design Space Exploration and Co-design. Under review, 2012.

[8] David Donofrio, Leonid Oliker, John Shalf, Michael F. Wehner, Chris Rowen, Jens Krueger, ShoaibKamil, and MarghoobMohiyuddin. Energy-efficient computing for extreme-scale science. Computer, 42:62–71, 2009.

[9] S. D. Hammond, G. R. Mudalige, J. A. Smith, S. A. Jarvis, J. A. Herdman, and A. Vadgama. WARPP: a toolkit for simulating high-performance parallel scientific codes. In Proceedings of the 2nd International Conference on Simulation Tools and Techniques, Simutools '09, pages 19:1–19:10, ICST, Brussels, Belgium, Belgium, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[10] M.A. Heroux et al. Improving Performance via Mini-applications. Technical Report SAND2009-5574, Sandia National Laboratories, September 2009. http://mantevo.org.

[11] E.S. Hertel, Jr., R.L. Bell, M.G. Elrick, A.V. Farnsworth, G.I. Kerley, J.M. McGlaun, S.V. Petney, S.A. Silling, P.A. Taylor, and L. Yarrington. CTH: A Software Family for Multi-Dimensional Shock Physics Analysis. In Proceedings, 19th International Symposium on Shock Waves, 1993.
[12] Curtis L. Janssen, Helgi Adalsteinsson, Scott Cranford, Joseph P. Kenny, Ali Pinar, David A. Evensky, and Jackson Mayo. A simulator for large-scale parallel architectures. International Journal of Parallel and Distributed Systems, 1(2):57–73, 2010.
[13] Jens Krueger, David Donofrio, John Shalf, MarghoobMohiyuddin, Samual Williams, Leonid Oliker, and Franz-Josef Pfreundt. Hardware/software codesign for energy-efficient seismic modeling. In Proceedings of SC2011, 2011.
[14] Sandia National Laboratories. Structural Simulation Toolkit (SST). http://www.cs.sandia.gov/sst/.
[15] Paul T. Lin, John N. Shadid, Marzio Sala, Raymond S. Tuminaro, Gary L. Hennigan, and Robert J. Hoekstra. Performance of a Parallel Algebraic Multilevel Preconditioner for Stabilized Finite Element Semiconductor Device Modeling.Journal of Computational Physics, 228(17), 2009.
[16] MarghoobMohiyuddin, Mark Murphy, Leonid Oliker, John Shalf, John Wawrzynek, and Samuel Williams. A design methodology for domain-optimized power-efficient supercomputing. In SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, pages 1–12, New York, NY, USA, 2009. ACM.
[17] W.L. Oberkampf, T.G. Trucano, and C. Hirsch. Verification, Validation and Predictive Capability in Computational Engineering and Physics. In Hopkins University, pages 345–384, 2002.
[18] S. J. Plimpton, R. Pollock, and M. Stevens. Particle-mesh ewald and rrespa for parallel molecular dynamics simulations. In Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, March 1997.
[19] Intel X-stack project, https://sites.google.com/site/traleikaglacierxstack/publications.
[20] X-stack projects funded by the U.S. Department of Energy, http://science.energy.gov/ascr/research/computer-science/ascr-x-stack-portfolio/.
[21] C. Lauderdale and R. Khan, Towards a codelet-based runtime for Exascale computing: position paper, Proceedings of the 2nd International Workshop on Adaptive Self-Tuning Computing Systems for the Exascale Era, 2012.