Exascale Workload Characterization and Architecture Implications

Prasanna Balaprakash, Darius Buntinas, Anthony Chan, Apala Guha†,
Rinku Gupta, Sri Hari Krishna Narayanan, Andrew A. Chien†, Paul Hovland, and Boyana Norris
Argonne National Laboratory, Mathematics and Computer Science Division
†Department of Computer Science, University of Chicago

***Abstract*—We describe a hybrid methodology for characterizing scientific applications and apply it to proxy applications (mini-apps and PETSc applications) representative of the DOE's future high performance computing workloads. The methodology uses source code analysis, performance counters, and binary instrumentation to capture instruction mix and memory access patterns for a range of model-sized datasets.**

With this empirical basis, we create statistical models that extrapolate application properties (instruction mix, memory size, and memory bandwidth) as a function of problem size. We validate these models empirically and use them to project the first quantitative characterization of an exascale computing workload, including computing and memory requirements. This exascale application extrapolation requires classification of applications as runtime or memory-capacity limited.

We evaluate the potential benefit of a radical new exa-scale architecture, stacked DRAM, and processing-under-memory (PUM). Our results show while the entire exascale workload is memory bandwidth limited, PUM-enabled tenfold increases in memory bandwidth can produce 1.4 to 4.2-fold speed improvements and convert the majority of these workloads to compute-limited. Additionally, the programming effort required to exploit these PUM advantages appears to be low.

I. Introduction

Application characterization is a key ingredient in understanding the impact of proposed architectural changes and identifying opportunities for architectural innovation. However, no single analysis tool provides a complete characterization of applications. We describe a methodology that employs multiple tools to build a more complete picture of application behavior, and we project these characteristics for future architecture scenarios. We apply this methodology to a collection of proxy applications representative of scientific workloads, especially the types of applications written and used by the U.S. Department of Energy. A number of these applications are being employed in architecture, system software, and application codesign.

We begin by describing a set of proxy applications, ranging from mini-drivers for parallel numerical libraries to simplified configurations of full applications. We then describe the set of tools used to characterize the applications, including tools based on sampling, binary instrumentation, and source code analysis. We also discuss some anticipated features of future architectures onto which we would like to project these applications. Each of the tools is well suited for gathering different kinds of information, and we describe the quantities measured and the observed characteristics for the applications under consideration. For this study, we focus on instruction mix and memory characteristics.

Using the analytical and empirical proxy application characterization as a base, we build statistical models in order to project the characteristics of each proxy application at exascale. In each case, shaping the extrapolation appropriately based on runtime or memory size limits for projected exascale machines [1]. Collectively, these projections represent the first quantitative characterization of an exascale workload. Using this characterization, we explore different exascale architecture scenarios, not only exposing the applications that are likely to benefit from radical changes such as *processor-under-memory* (PUM) but also quantifying to what degree they benefit and at what programming effort. This paper includes the following specific technical contributions.

- Hybrid characterization of scientific computing "proxy applications" representing future high-performance computing (HPC) computing, producing quantitative application properties and requirements.
- A projected exascale workload, derived by statistical projection of the proxy applications, that provides a first estimate of a range of quantitative characteristics, including operation mix, compute and memory intensity, and memory bandwidth.
- Evaluation of one promising exascale architecture organization, PUM, indicating 1.4- to 4.2-fold performance increases, a benefit that may be available for many applications at a modest (localized) programming effort.

The rest of the paper is organized as follows. In Section II, we survey related work and background. In Section III, we analyze proxy application properties. In Section IV, we extrapolate these results to exascale and evaluate architecture scenarios. Section V summarizes our current efforts and briefly discusses future research.

II. Background

In this section we survey related work and describe the application workload, characterization tools, and exascale architecture scenarios we consider.

A. Related Work

Many instances of workload characterizations for existing architectures can be found in the literature. Here we include

only those that aim to characterize multi-application scientific computing workloads in order to enable performance projections for potential future architectures. However, none of these explicitly address exascale workloads.

Marin and Mellor-Crummey [2] introduce a performance characterization approach that separates the contribution of application-specific factors from the contribution of architectural features to overall application performance. Using a combination of static and dynamic analysis of application binaries, they create models that can be used for cross-architecture prediction with accuracy within 20% for the Sweep3D application.

Carrington et al. [3] characterize performance on different architectures by convolving application signatures (e.g., based on memory or communication event traces) with an architecture model. The cost of tracing is the main limiting factor in this approach. Mills et al. [4] address this by introducing a compression scheme to capture data access patterns without storing a full trace.

Some characterization and prediction approaches (e.g., [5]) employ neural networks to make performance predictions. These approaches typically require a large training set (e.g., hundreds or thousands of instances) to produce low-error predictions. Because the neural networks are constructed with data collected on existing architectures, they may not be able to accurately model significantly different architectures.

Analytic models are also used to characterize the behavior of applications. Heroux et al. [6] introduce performance models for two of the Mantevo mini-apps (miniFE and miniMD), including characterization on commodity architectures and scalability studies.

Guha et al. [7] analyze a collection of 34 programs using clustering along operation and data types, as well as memory characteristics. This analysis produces 25 multi-loop clusters corresponding to 90% of the computations, showing that relatively few patterns can be identified that span a much larger and varied collection of computations. This work in part motivated the study in this paper.

The "memory wall" or von Neumann bottleneck has been a critical challenge in computer architecture for many years. Extensive architecture research explores the viability of integrating computing in memory, so-called processor-in-memory (PIM) systems [8]. Such systems, however, typically suffer from relatively slow logic and tight power/energy envelopes implied by Si process integration. The PUM approach is a symbiotic approach placing computing closer to memory with die-stack integration, with heterogeneous process technologies.

Notable in the HPC space is the HTMT superconducting processor project (late 1990s and early 2000s [9]). Here, most relevant is the intelligent memory processor that was used in complementary fashion to the HTMT superconducting processor, an organization similar to our "processor under memory" scenario. Of course, other technology and application differences abound, so direct comparison is difficult.

We use the DOE's mini-applications, designed as examples to facilitate codesign of exascale architectures, to create extrapolated exascale workloads. To our knowledge, this work is the first quantitative analysis of an exascale application workload. Many workshops and reports (e.g., [10]) by DOE, the international exascale community, and the broader scientific computing community have motivated the need for exascale systems and potential driving applications, but to date, none include detailed quantitative analysis for projected exascale workloads.

B. Proxy applications

We considered ten proxy applications for our initial characterization: three from the Mantevo [6] suite (miniFE, miniMD, and HPCCG), three Nek5000-based applications [11] (eddy, vortex and turbChannel), and four PETSc [12] applications.

Mantevo's miniFE is a proxy for implicit unstructured finite-element codes, and miniMD is a proxy for classical molecular dynamics simulation with short-ranged Lennard-Jones interactions. HPCCG is a simpler proxy for the same application class but is missing some algorithmic steps.

Nek5000 is a spectral-element computational fluid dynamics code used in a variety of applications (e.g., nuclear reactor simulations). The Nek5000-based proxy applications simulate fluid flow in simple 2-D (eddy) and 3-D geometries (vortex and turbChannel).

PETSc is a popular parallel numerical library for solving the linear and nonlinear systems of equations resulting from discretized PDEs. Three of the PETSc proxy applications (Ex19, Ex20, and Ex30) represent 2-D and 3-D problems with different stencil sizes and varying physics complexity and rely on an iterative Newton-Krylov solver. Ex10 solves a sparse linear system using ILU-preconditioned GMRES.

C. Characterization tools

The following tools were used in the characterization.

a) Sampling-based analysis: HPCToolkit [13] is a suite of tools for measuring and analyzing parallel and serial application performance, including support for fine-grained (e.g., loop-level or single-line) hardware counter measurements, such as cache misses or floating-point operations. HPCToolkit includes visualization tools for viewing profile and tracing data together with the source code.

b) Binary instrumentation: Pin [14] is a dynamic binary instrumentation tool and a set of application programming interfaces (APIs) for implementing analysis tools. Pin is capable of parsing the entire executable as well as all dynamically loaded libraries, sections, functions, and instructions. Additionally, Pin can gather dynamic data, such as instruction counts, branch outcomes, register values, memory addresses accessed, and memory values (however, Pin is a user-level tool and therefore cannot instrument system-level code).

c) Source code analysis: PBound [15] is a static analysis tool that computes an upper bound on the performance of an application. PBound takes as input C/C++ source code along with a simple architecture description file and generates parameterized closed-form expressions quantifying different types of memory accesses and computations.

D. Exascale and "processor under memory"

Scientific computing is driven by the desire to model computationally complex natural phenomena at increasing levels of fidelity and precision and has successfully utilized terascale (e.g., ASCI Red) and petascale (e.g., ORNL's Jaguar) resources. The majority of exascale systems research is based on projections of Moore's law [16] that reflects an increasingly poor scaling of semiconductor technologies characterized by continued increases in transistor density but a decreasing ability to scale voltage (due to threshold voltage limits) and decreasing improvements in transistor speed. A number of exascale machine projections point to an aggregate compute capacity of an exaflop and total memory capacity of 100 petabytes [1]. These trends present significant challenges for both hardware and software designers, motivating a recent international effort by the HPC community to investigate exascale challenges and solutions for hardware/software codesign and software development [17].

Memory bandwidth is widely recognized as a critical factor in HPC systems, and the same is anticipated for exascale systems. Current single-chip systems such as Intel's MIC and Nvidia's Kepler have memory bandwidths of $\approx$ 100-200 GB/s = 0.2 TB/s, with only slow increases in conventional DDR/GDDR technologies [18].

Recently, several vendors have proposed the possibility of a new hardware organization *processor-under-memory* systems, that increases memory bandwidth dramatically by placing logic chips under DRAM die-stacks. Major new industry consortia and standards have been established to push this new model [19]. These efforts are similar in spirit to prior efforts to build *processor-in-memory* systems but differ in using heterogeneous semiconductor process technology for the underlying logic die (fast transistors) and stacked DRAM dies (low-leakage, slow transistors). PUM-based *hybrid memory cube* (HMC) can potentially increase node memory bandwidth tenfold to 10 TB/s. Motivated by this important potential architecture change, we examine the impact of PUM scenarios for exascale using extrapolated exascale application models.

III. Characterizing Applications

We characterize our scientific application workload using the analytical and empirical tools described in Section II-C. We focus on three attributes: basic application properties (e.g., compute operations, memory operations), number, identity, and importance of performance-critical regions (hotspots), and variation of basic application properties across the workload (e.g., memory bandwidth requirements). All these attributes are characterized as a function of application input (dataset) size. We use varied tools for this characterization and compare the results across the tools in order to demonstrate close alignment of results. For consistency and clarity of presentation, unless otherwise specified, quantitative application results are based on Pin data. We use only cache miss counts from HPCToolkit in the results. In Section IV, we leverage this workload characterization to extrapolate an exascale workload.

A. Testbed and Experimental Methodology

The testbed environment for profiling applications consisted of a single Ubuntu 10.04.4 Linux box with two quad-core Intel Xeon E5520 CPUs running at 2.27 GHz with Hyper-threading enabled, 8 MB L3 cache and 24 GB of DDR3 memory. The compiler used in our experiment is Intel compiler suite 11.0.081.

As a part of our experimental methodology, we use several different tools for profiling applications, with a two-fold goal: (1) comprehensive characterization data for all applications, and (2) validated result data.

In this paper, we differentiate between *instructions* and *operations*. For example, a machine *instruction* (e.g., the `FMA` floating-point multiply add instruction) may perform multiple *operations* (e.g., `FMA` performs two floating-point operations).

We used the HPCToolkit software to profile the applications using the hardware counters. We gathered counts of load, store, floating-point, and branch operations as well as total instruction counts, cycle counts, and L3 cache misses. In addition to HPCToolkit, we also used Pin to gather counts of load, store, floating-point, branch, and integer operations and counted the number of bytes referenced in the load and store operations.

Moreover, we used PBound to analyze applications. PBound was applied to analyze kernels of interest in HPCCG (which were simplified as required). Further, the expressions (i.e., floating-point loads, floating-point stores, floating-point operations) generated by PBound were manually interpreted and processed by using knowledge about the application. In its current form, PBound cannot analyze the entire PETSc library, the Mantevo proxy applications, or Fortran-based applications such as Nek5000.

We base the majority of our analysis on operation counts obtained with Pin. Notably, we use cache miss counts from HPCToolkit for memory bandwidth analysis. The remainder hybrid tool data is used for validation purposes only.

B. Characterizing Applications: Results

To expose scaling characteristics, we exercise our applications with a variety of problem sizes. The proxy applications are executed with varied problem sizes labeled as classes A through G. The smallest benchmark is class A, which takes about one second to run. The runtimes for classes C and E are approximately 10 and 100 seconds, respectively. The largest is class G, which takes about 1,000 seconds or longer. The intermediate-size classes B, D and F are defined in between the classes A and C, C and E, and E and G, respectively. The runtime for each larger class is increased either by enlarging the physical problem size (e.g., for PETSc Ex20) or by increasing the complexity of the solution (e.g., for Nek5000). Not all proxy applications could be configured for all classes; specifically, PETSc Ex10 cannot be run for classes B, D, or F. Similarly, the Nek5000 proxy applications eddy, vortex, and turbChannel could be run only for A and B classes, C and D classes, and E and F classes, respectively. We found that the runtimes in class A were too short to produce

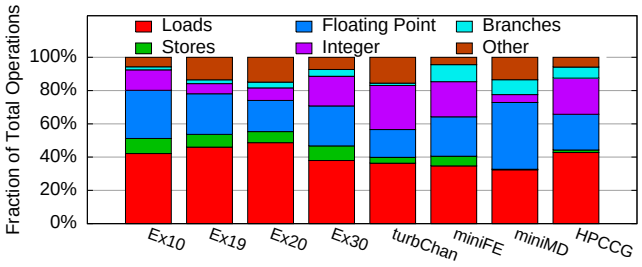


Fig. 1. Operation type vs. application

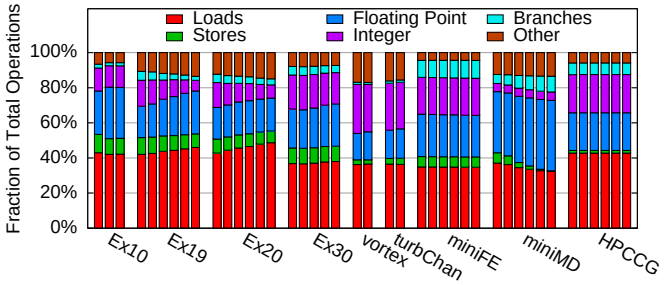


Fig. 2. Operation type vs. problem size. For each application, class increases from left to right

TABLE I
VALIDATION THROUGH PERFORMANCE COUNTERS: PIN VS. HPCTOOLKIT VS. PBOUND

Application [CLASS]	Tool	Loads	Stores	Branches	Total Instruct.
Mantevo miniFE[G]	Pin	5.61E+11	1.11E+11	2.52E+11	1.83E+12
	HPCT	5.95E+11	1.13E+11	2.78E+11	1.94E+12
Nek5000 turbChan[E]	Pin	7.62E+11	6.74E+10	2.75E+10	1.77E+12
	HPCT	7.62E+11	6.75E+10	2.75E+10	1.77E+12
PETSc Ex30[G]	Pin	1.31E+12	3.34E+11	1.59E+11	2.71E+12
	HPCT	1.33E+12	3.37E+11	1.77E+11	2.79E+12
Mantevo HPCCG[G]	Pin	6.60E+11	7.59E+09	1.29E+11	1.53E+12
	HPCT	6.60E+11	7.60E+09	1.29E+11	1.52E+12
	PBound	6.83E+11	7.60E+09	n/a	n/a

interesting results but can serve as a validation test for some of our tools. Hence, class A is not considered in our subsequent analysis.

Figure 1 shows the operation-type composition for various applications (in the G class, except for turbChannel which belongs to class F). More specifically, we show the percentage composition of significant operations such as loads, stores, floating point, integers, and branches against the total number of operations in the overall application. We note that load operations dominate majority of the PETSc applications (Ex10, Ex19, Ex20, and Ex30), followed by floating-point operations and integer operations. Along with PETSc, we selected the Nek5000 turbChannel application and the Mantevo set of applications (miniFE, HPCCG, and miniMD) because they show a diverse operation composition of loads, stores, floating-point operations, and integers. This data for varied applications and application sizes (i.e., classes) helps evaluate other application characteristics and extrapolate the characteristics to future new machines (see the following sections).

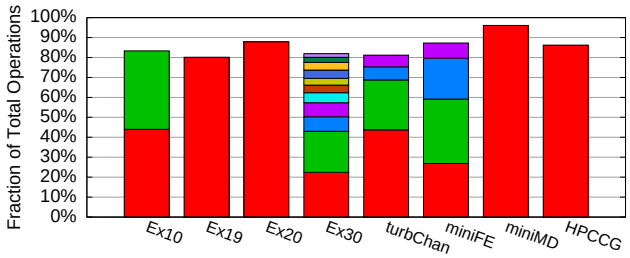
To validate our measurements, we compare the Pin data with HPCToolkit and PBound results. The HPCToolkit and PBound tools can count only instructions and hence we use instructions for validating data from the HPCToolkit, Pin, and PBound sources. Table I shows the raw counter values for significant instructions for diverse applications and runtime classes. As seen from the table, there is negligible difference between the results obtained from the three tools. Based on this validation, we deem Pin results to be accurate, and they form the basis for the rest of our analysis.

Figure 2 shows application operation-type composition with increasing input sizes (classes) from left to right. For majority of the applications, the composition is stable across the size classes (PETSc Ex10, PETSc Ex30, Nek5000, miniFE and HPCCG). However, two applications (PETSc Ex19 and PETSc Ex20) vary in operation composition; both loads and floating-point operations increase steadily with the input size at the expense of the fraction of branches and integer operations. These applications will become increasingly memory-intensive and floating-point operation intensive for larger sizes. For miniMD, however, the fraction of branches increases with input size, while the fraction of loads and stores decreases. The Nek5000 applications (vortex and turbChannel) show similar operation composition, and that composition is constant with increasing input sizes.

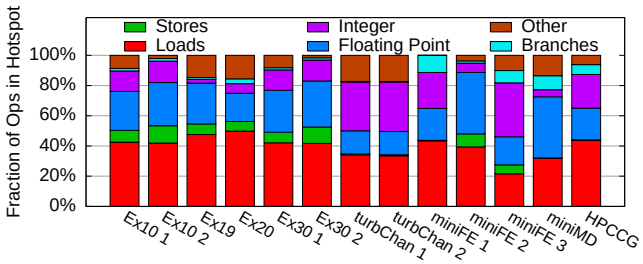
The previous graphs focus on analyzing the operation composition in various applications. As a next step, we identify hotspots, that is, functions where the application conducts majority of its operations. Hotspots, if present, can provide a unique opportunity to: (1) improve application performance by optimizing the hardware that is used by the operations during these hotspots and, (2) improve application performance by optimizing the hotspot code itself.

Figure 3(a) shows the top hotspots in the application codes using the largest class, ranked based on the amount of operations performed. As seen in the graph, several applications spent significant amounts of time in a small subset of routines. We label as a hotspot any routine that has over 15% of the overall application operation count. Note the diversity in hotspot composition for the various applications. For applications such as PETSc Ex19 and Ex20, Mantevo miniMD, and HPCCG, the majority of the time is spent in a single routine. For example, in HPCCG, sparse matrix computation constitutes over 80% of the application runtime. Ex10 has two hotspots, and miniFE has three, which together constitute about 80% of the total operation count. In contrast, PETSc Ex30 has two hotspots, but they constitute less than 50% of the total operation count.

Figure 3(b) shows the operation-type composition for the hotspots in the applications. The figure considers the total number of operations in the hotspots and shows what fraction was contributed by each operation type. For applications such as PETSc Ex19 and Ex20, Mantevo miniMD, and HPCCG, where a single hotspot dominates the application, we see that

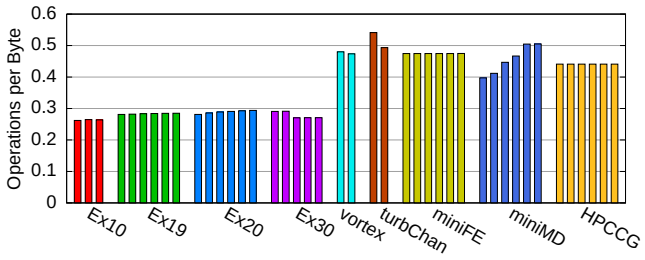


(a) Hotspots for various applications

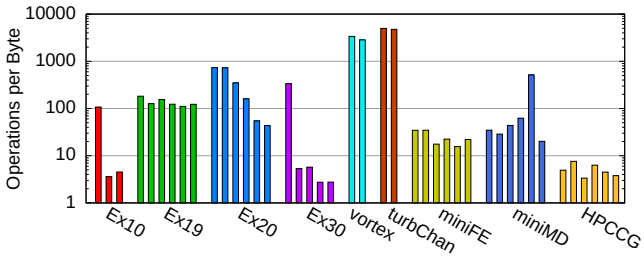


(b) Operation types within identified hotspots

Fig. 3. Application hotspots



(a) Core level



(b) Chip level

Fig. 4. Operations per byte in the top hotspots of each application for various classes. For each application, class increases from left to right.

the trends are similar to those in Figure 2. We note that two of the three hotspots of miniFE are integer intensive, whereas one hotspot is floating-point intensive.

In Figure 4 we show the compute intensity of the top hotspots in the applications for different classes. There is one bar per class per application, and the bars are grouped by application. Each bar represents the ratio of operations to bytes transferred at the core and chip levels. In Figure 4(a) we show bytes transferred at the core level by counting the bytes transferred in each load or store operation, while in Figure 4(b) we show bytes transferred at the chip level by counting the bytes transferred due to L3 cache misses. Counting transfers at the core level gives an upper bound on memory traffic, while counting transfers at the chip level counts the actual traffic generated on the specific architecture used in this evaluation. As expected, the operations-per-byte ratio at the chip level increases significantly over the ratio at the core level because most of the traffic is handled by the cache. For the most part, the operations-per-byte ratio at the core level is relatively constant for different classes within each application. At the chip level, however, we see that the smaller problem sizes have larger operations-per-byte ratios, most likely because the data fits better in the cache. In Figure 4(a), the operations-per-byte ratio for miniMD increases with problem size. The reason is that for smaller problem sizes there are two hotspots with different operations-per-byte ratios, and as the problem size increases, one of the hotspots increasingly dominates. We see a similar increase in operations-per-byte ratio for miniMD in Figure 4(b).

In Figure 5, we present the measured memory bandwidth consumed in the top hotspots of the proxy applications. We see that Ex10, Ex30 and HPCCG all achieved over 10 GB/s

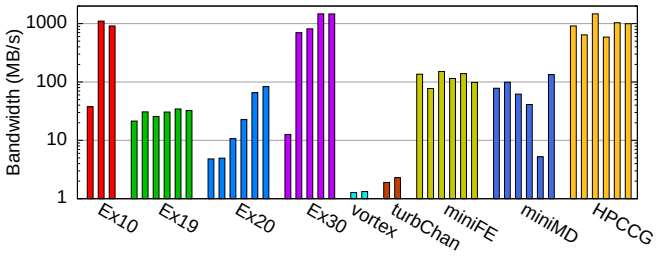


Fig. 5. Memory bandwidth for top hotspots, class increases left to right.

indicating they are already bandwidth limited, will likely require even more bandwidth at larger sizes. Ex20 shows an increase in bandwidth utilization as the problem size increases indicating that it may eventually become memory bound at larger problem sizes. The remaining applications show relatively constant bandwidth utilization.

IV. Exascale Insights

A. Extrapolating an Exascale Workload

Using the characterization data, we create an extrapolative model that projects key characteristics of proxy applications to a range of machine and application configurations. We perform statistical validation of these models in order to ensure their accuracy, and we then use them to extrapolate an exascale workload. This exascale workload projection aims to understand exascale application requirements - compute and memory. And, then to use these requirements to assess potential exaflop machines [1]. We focus on two scenarios among the wide range of potential exascale architectural features.

TABLE II
EXASCALE WORKLOAD PROJECTION MODELS

Appl.	Exascale Projection Models, where $N = n_1 \cdot n_2 \cdot n_3$ and $c_i, i \in [1,5]$ are constants
Ex19	$f(n_1,n_2,n_3) = c_1 + c_2 \cdot (N \cdot n_1)$
Ex20	$f(n_1,n_2,n_3) = c_1 + c_2 \cdot (n_1 \cdot n_2) + c_3 \cdot (n_1 \cdot n_2)^2 + c_4 \cdot n_3^2 + c_5 \cdot n_3^3$
Ex30	$f(n_1,n_2,n_3) = c_1 + c_2 \cdot N \cdot (n_1 \cdot n_2)$
miniMD	$f(n_1,n_2,n_3) = c_1 + c_2 \cdot N + c_3 \cdot N^2$
miniFE HPCCG	$f(n_1,n_2,n_3) = c_1 + c_2 \cdot N$

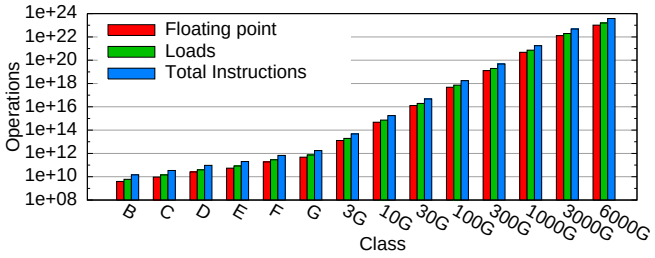


Fig. 6. Projection for HPCCG

TABLE III
MEMORY CONFIGURATION MODELS

Application	Memory Configuration Models, where CS = cache size, $N = n_1 \cdot n_2 \cdot n_3$, and $c_i, i \in [1,5]$ are constants
Ex19	$f(n_1,n_2,n_3) = CS + c_1 + c_2 \cdot N$
Ex20	$f(n_1,n_2,n_3) = CS + c_1 + c_2 \cdot (n_1 \cdot n_2) + c_3 \cdot (n_1 \cdot n_2)^2 + c_4 \cdot n_3^2 + c_5 \cdot n_3^3$
miniFE,miniMD Ex30, HPCCG	$f(n_1,n_2,n_3) = CS + c_1 + c_2 \cdot N + c_3 \cdot N^2$

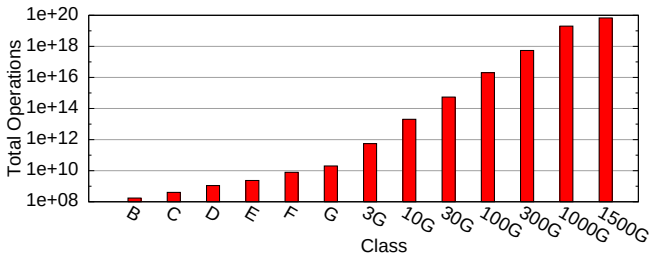


Fig. 8. HPCCG projection for memory usage

Specifically, our model projects the properties of each application as a whole to exascale problem sizes, producing estimates of the number of instructions, loads, stores, branches, and floating-point operations to quantify the instruction mix and resulting performance requirements. We use the data obtained from the Pin tool to build the empirical models. We model the metrics collected from six proxy applications, Ex19, Ex20, Ex30, miniFE, miniMD, and HPCCG, because they all have results for classes B-G.

We consider linear, quadratic, and cubic models. To evaluate model accuracy, we adopt the leave-one-out cross-validation technique. In this approach, in order to predict a metric for an input size $s \in \{\text{B, C, D, E, F, G}\}$, all sizes except s are used for training. This process is repeated such that each input size is used once as the prediction input size. We compute the coefficient of determination (R^2 goodness of fit) between the predicted values and the original values on all the input sizes. For each application and for each metric, a model with minimum R squared value is selected to project the metrics for large input sizes. Table II shows the model type that best matched each application (for all the metrics). The distinct model for each metric and application pair has a unique derived set of coefficients. In total, we obtain 48 models (8 metrics $\times$ 6 applications).

For each application, we project the metrics for a geometric series of input sizes, 3$\times$G, 10$\times$G, 30$\times$G, 100$\times$G,. . .,6000$\times$G. Figure 6 shows the plots of the projection results for HPCCG. We found a similar cubic growth trend on all applications. While paper limits preclude full presentation of the model validation, we found high R^2 values between 0.96 and 0.98.

B. Projecting Exascale Workloads

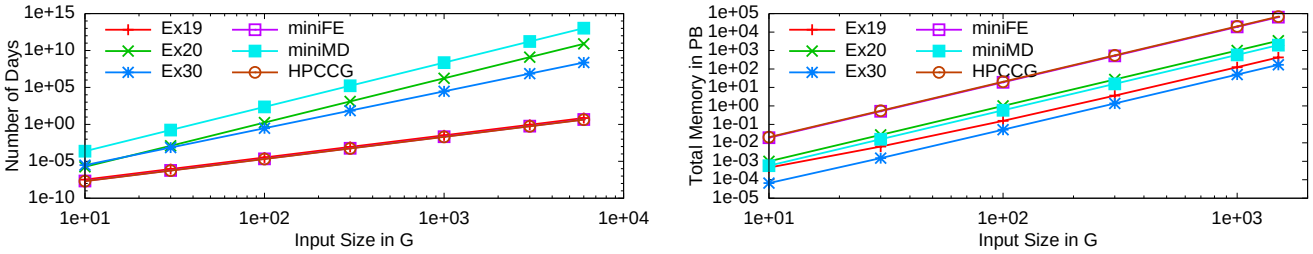
First we classify the applications based on compute intensity. For this purpose, we estimate the time required to run the exascale workloads on exaflop machines. This is given by the ratio of number of instructions in the exascale workloads for each application and exa-ops (10^{18}) and is shown in Figure 7(a). Sorting the applications by runtime, we observe that the most computationally intensive application is miniMD, followed by Ex20, Ex30, Ex19, miniFE, and HPCCG.

Second, we classify the applications based on memory intensity. We used Valgrind's Massif tool to collect the total memory usage. As described in Section IV-A, we model the memory usage for each application as a function of workload size, and we project the memory usage for exascale workload. The models are shown in Table IV. Exemplary results on HPCCG are shown in Figure 8. Total memory requirements of exascale workloads for different applications are shown in Figure 7(b). The most memory intensive application is HPCCG, followed by miniFE, Ex20, miniMD, Ex19, and Ex30.

Using Figures 7(a) and 7(b), we project the feasible application sizes on an exascale system based on realistic runtimes (24 hours at exaflop sustained rate) and memory capacity (100 PB). Because of different scaling properties, the applications separate as shown in Table V with three (miniMD, Ex20, and Ex30) compute-limited and three (Ex19, minFE, HPCCG) memory-capacity limited. The memory capacity constraints are extreme for miniFE and HPCCG, as the 100 PB constraints them to runs of approximately 30 exa-ops, around thirty seconds if high speedups can be achieved. Ex19 is also memory capacity limited, but at a run size approximately 100 times larger than miniFE and HPCCG, less than one hour at high speedups.

C. Exascale Workload Requirements and Evaluating PUM

We characterize the projected exascale workload in Figures 9 and 5. These characteristics provide an empirical basis for shaping future architectures. Exascale workload instructions



(a) Runtime (b) Memory size

Fig. 7. Projected exascale application requirements on an exaflop machine.

TABLE IV
EXASCALE WORKLOAD: MEMORY SIZE PROJECTION MODELS

Appl.	Exascale Projection Models, where $N = n_1 \cdot n_2 \cdot n_3$, and c_1, c_2, c_3, and c_4 are constants
Ex19,Ex30	$f(n_1, n_2, n_3) = c_1 + c_2 \cdot N + c_3 \cdot (n_1 \cdot n_2) + c_4 \cdot n_1$
Ex20	$f(n_1, n_2, n_3) = c_1 + c_2 \cdot (n_1 \cdot n_2) + c_3 \cdot (n_1 \cdot n_2)^2$
miniFE,miniMD HPCCG	$f(n_1, n_2, n_3) = c_1 + c_2 \cdot N$

TABLE V
SCALING LIMITS - RUNTIME (EXAOP) AND MEMORY (100 PB)

Appl	Exascale limit		Critical Limit	Feasible size
	24 hrs	100PB		
Ex19	5000G	1000G	Memory	1000G
Ex20	92G	500G	Compute	92G
Ex30	130G	1500G	Compute	130G
miniMD	41G	600G	Compute	41G
miniFE	5000G	250G	Memory	250G
HPCCG	5000G	250G	Memory	250G

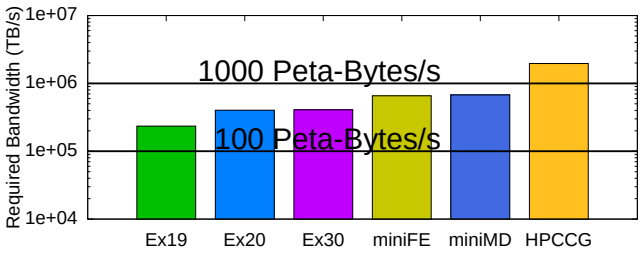
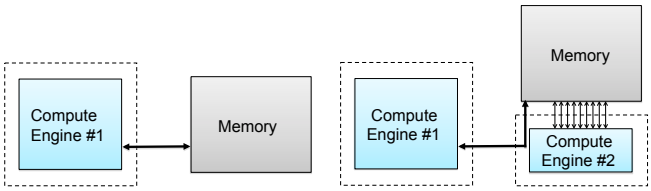


Fig. 10. Exascale application memory bandwidth requirements.



(a) Traditional (b) Traditional + PUM

Fig. 11. Two exascale architecture scenarios.

mixes, compared with the small-dataset data shown in Figures 1 and 2, exhibit several interesting characteristics. The floating-point fraction is slightly higher across the board (as overheads reduce) and ranges from 20% to as much as 40% for miniMD. The memory fraction increases steadily on the load side for several applications (Ex19, Ex20), reaching over 40%. This increase appears to correspond directly to a reduction in integer (presumably control and indexing overhead). There is no corresponding increase in the store fraction; and in fact, for several applications, notably miniMD, but also Ex20 and Ex19 to a lesser degree, the store fraction decreases steadily.

Using the projected feasible dataset sizes for each scientific application, we estimate exascale memory bandwidth requirements. Using exascale problem sizes, we estimate runtime as the ratio of total number of operations and exaops, assuming good speedup. The off-chip memory bandwidth required is the cache misses per second times block size (64 bytes). Results are shown in Figure 10. The imposition of exa-scaling constraints produces results different from what one might expect, given the small-scale studies in Figure 5. For example, Ex19, miniFE, and miniMD do not appear to be memory bandwidth limited at small scale, but become memory-limited at exascale because of data size increases. The rapid growth trends of Ex20 does not make it significantly more memory bandwidth limited than are other applications at exascale.

Subsequently, we used the memory bandwidth projections to

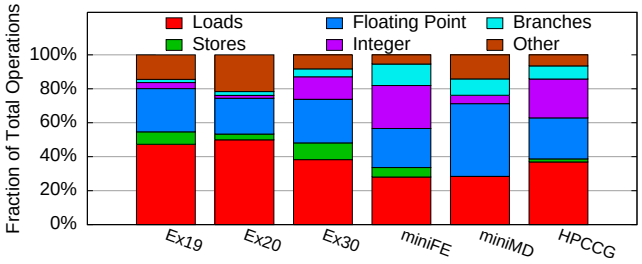


Fig. 9. Exascale workload instruction mix by application

TABLE VI
TWO EXASCALE ARCHITECTURE SCENARIOS (EACH IS 100K NODES)

Node Size	CPU Compute	CPU Mem	CPU MemBW	PUM Compute	PUM MemBW
Trad.	10 Teraop	100 GB	1 TB/s	n.a.	n.a.
Trad. + PUM	10 Teraop	100 GB	1 TB/s	1 Teraop	10 TB/s

TABLE VII
TRADITIONAL AND TRADITIONAL+PUM PERFORMANCE AT EXASCALE

App	Scaling Limit Exascale	PUM Improvement	Key Limit PUM	Programming Effort
Ex19	Memory	4.26	Compute	Low
Ex20	Compute	2.49	Compute	Low
Ex30	Compute	2.45	Compute	High
miniMD	Compute	1.48	Compute	Low

evaluate the potential benefits of radical new architecture organizations such as "processor-under-memory". We considered two types of exascale node architecture varying processor-memory interconnection, as captured in Table VI and Figure 11. The first is a traditional processor and memory, with compute rate on the CPU able to achieve 10 TF thanks to advanced integration, but bandwidth limited to 1 TB/s (Figure 11(a)). The second adds a processor-under-memory (Figure 11(b)), which by virtue of its broad, close physical connection to a stacked set of memory dies [19] can achieve a tenfold greater memory bandwidth, or 10 TB/s. With mobile wide I/O, a single DRAM die can deliver upwards of 13 GB/s (2015) with 2x energy efficiency advantage. HMC, even more aggressive, can achieve even 10x further bandwidth increases through the use of larger numbers of thru-silicon-vias.

Our application studies show that exascale architectures are memory bandwidth limited, despite projected increases ($<$ 200 GB/s to 1 TB/s). We estimate the potential runtime reduction increased PUM memory bandwidth for exascale problem sizes, by dividing the total memory traffic by the increased bandwidth. Table VII shows potential PUM benefits range from 1.48x to 4.26x. Note that we did not include miniFE and HPCCG in the final analysis since the runtime, which is limited by memory requirement, is too short (less than a minute) to be significant.

Of course, the correct partition and binding of computation to the processor and PUM are critical to achieving high performance. The cost of data movement between primary core and PUM is also a critical factor. Looking the breakdown for application hotspots, we see that the programming effort to exploit PUM may be manageable for many key applications. Ex19, Ex20, miniMD, and HPCCG all exhibited only a single hotspot, suggesting that partition won't be difficult; in fact the CPU may not be heavily exploited. The Ex10 and miniFE codes have multiple hotspots, requiring greater programming effort, and the Ex30 example represents high difficulty.

V. SUMMARY AND FUTURE WORK

We have created a quantitative model of an exascale workload and applied it to evaluate a new architectural approach. However, there remains much opportunity to increase our insight into the likely application properties and architectural opportunities for exascale systems. For example, we plan deeper analysis of the characterization data, and perhaps creation of richer statistical models of the exascale workloads that can be used for architecture design space exploration. Here we have considered only one architectural feature, but many other system architecture choices remain including node granularity, interconnect structure, or accelerator structure, which could be addressed by this methodology.

Acknowledgments

This work was supported by the U.S. Department of Energy Office of Science DE-AC02-06CH11357 and NSF OCI-1-57921.

REFERENCES

[1] P. Kogge and et al., "Exascale computing study: Technology challenges in achieving an exascale systems," 2008, DARPA IPTO Study Report for William Harrod, available from http://tinyurl.com/6c284o9.

[2] G. Marin and J. Mellor-Crummey, "Cross-architecture performance predictions for scientific applications using parameterized models," *SIGMETRICS Perform. Eval. Rev.*, vol. 32, no. 1, pp. 2–13, Jun. 2004.

[3] L. Carrington, A. Snavely, and N. Wolter, "A performance prediction framework for scientific applications," *Future Generation Computer Systems*, vol. 22, no. 3, pp. 336–346, Feb. 2006.

[4] C. Mills, A. Snavely, and L. Carrington, "A tool for characterizing and succinctly representing the data access patterns of applications," in *, 2011 IEEE International Symposium on Workload Characterization (IISWC)*, nov. 2011, pp. 126 –135.

[5] E. S. Sarioglu, C. Bayrak, and K. Iqbal, "Neural network based performance prediction with feature extraction," in *Proceedings of the 2006 IEEE International Conference on Engineering of Intelligent Systems*.

[6] M. A. Heroux, D. W. Doerer, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich, "Improving performance via mini-applications," Sandia National Laboratories, Tech. Rep. SAND2009-5574, Sept. 2009.

[7] A. Guha, P. Cicotti, A. Snavely, and A. A. Chien, "The 10x10 foundation for heterogeneity: Clustering applications by computation and memory behavior," University of Chicago, Tech. Rep. TR-2012-01, Feb. 2012.

[8] C. Kozyrakis, "A media-enhanced vector architecture for embedded memory systems," Berkeley, CA, USA, Tech. Rep., 1999.

[9] E. Upchurch, T. Sterling, and J. Brockman, "Analysis and modeling of advanced PIM architecture design tradeoffs," in *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, ser. SC '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 12–.

[10] S. Ashby and et al., "ASCAC subcommittee report: The opportunities and challenges of exascale computing," 2010, available from http://tinyurl.com/7dcwn82.

[11] H. M. Tufo and P. F. Fischer, "Terascale spectral element algorithms and implementations," in *ACM/IEEE conference on Supercomputing*, Portland, OR, 1999.

[12] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, "Efficient management of parallelism in object oriented numerical software libraries," in *Modern Software Tools in Scientific Computing*, E. Arge, A. M. Bruaset, and H. P. Langtangen, Eds. Birkhäuser Press, 1997, pp. 163–202.

[13] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent, "Hpctoolkit: tools for performance analysis of optimized parallel programs," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 6, pp. 685–701, 2010.

[14] C. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. Reddi, and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," in *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, 2005.

[15] S. H. K. Narayanan, B. Norris, and P. D. Hovland, "Generating performance bounds from source code," Argonne National Laboratory, Tech. Rep., 9 2010, also available as Preprint ANL/MCS-P1685-1009.

[16] S. Borkar and A. A. Chien, "The future of microprocessors," *Communications of the ACM*, vol. 54, pp. 67–77, May 2011.

[17] "International exascale software project," http://tinyurl.com/cvy9uzs.

[18] A. Shilov, "DDR4 memory projected to debut in 2014," http://tinyurl.com/5uebg5g.

[19] J. T. Pawlowski, "Hybrid memory cube: breakthrough DRAM performance with a fundamentally re-architected DRAM subsystem," in *Proceedings of the 23rd Hot Chips Symposium*, ser. HotChips '11, 2011.