Characterization and Bottleneck Analysis of a 64-bit ARMv8 Platform

Michael A. Laurenzano† Ananta Tiwari⋆ Allyson Cauble-Chantrenne⋆
Adam Jundt⋆ William A. Ward, Jr.‡ Roy Campbell‡ Laura Carrington⋆

†University of Michigan, *mlaurenz@eecs.umich.edu*
⋆EP Analytics, {*ananta.tiwari, allysonc, adam.jundt, laura.carrington*}*@epanalytics.com*
‡HPC Modernization Program, Department of Defense, {*william.ward, roy.campbell*}*@hpc.mil*

***Abstract*—This paper presents the first comprehensive study of the performance, power and energy consumption of the AppliedMicro X-Gene, the first commercially available 64-bit ARMv8 platform, for HPC workloads. Our study includes a detailed comparison of the X-Gene to three other architectural design points common in HPC systems. Across these platforms, we perform careful measurements across 400+ workloads, covering different application domains, parallelization models, floating-point precision models and memory intensities. We find that the X-Gene has an average of 1.2× better energy consumption than an Intel Sandy Bridge, a design commonly found in HPC installations, while the Sandy Bridge is an average of 2.3× faster than X-Gene.**

Precisely quantifying the causes of performance and energy differences between two platforms is an important but challenging problem that is often addressed via detailed simulation, an approach that has limited ability to scale up to full applications and broad workload mixes. Instead, this paper adopts a statistical framework called Partial Least Squares (PLS) Path Modeling to solve this problem. PLS Path Modeling allows us to capture complex cause-effect relationships and difficult-to-measure performance concepts relating to the effectiveness of architectural units and subsystems in improving application performance using readily available hardware counter measurements. We use PLS Path Modeling to quantify the causes of the performance differences between X-Gene and Sandy Bridge in the HPC domain, finding that the performance of the memory subsystem is the dominant cause of these differences.

I. Introduction

The 64-bit ARMv8 platform has captured the attention of system designers across a number of computing domains, ranging from mobile and tablet computing to large-scale cloud and HPC systems. This is evidenced by the fact that ARM has sold over 50 licenses to the technology [14]. In particular, many in the HPC community are convinced that energy-efficient ARM designs, and the 64-bit ARMv8 platform specifically due to its improved double-precision and SIMD support over previous ARM generations [28], will figure prominently into the set of solutions that allow continued progress in the scientific and engineering endeavors that rely on the massive scale of computation offered by large HPC systems [7], [37], [31], [42], [49], [23]. Yet it remains unclear what the operating characteristics of the ARMv8 platform are, and whether those characteristics position the ARMv8 as an effective design point for running HPC workloads.

In this paper, we present the first comprehensive study of the performance, power and energy consumption of the AppliedMicro X-Gene, the first of a coming wave of commercially available ARMv8 implementations. Our study is centered around comparing the X-Gene to three design points from Intel: (1) the Xeon E5-2670v1 (Sandy Bridge), a popular design point found in large-scale HPC systems today [4], (2) the Xeon E5-2667v3 (Haswell), representative of a high performance design point likely to be found in upcoming generations of HPC systems, and (3) the Atom C2758, a low performance, energy efficient design. This study captures a spectrum of comparison points, including several application domains and over 400 test cases covering different parallelization models, and floating point and memory intensities.

Our study comparing these four design points reveals that the X-Gene is an average of 7% faster than the Intel Atom, but with a far higher power draw (typically 2-2.5× higher). The Haswell consumes 1.3× less energy than the X-Gene and outperforms it by an average of 3.4×. Additionally, the X-Gene consumes 1.2× less energy than the Intel Sandy Bridge due to slower execution on the X-Gene (typically 2-3×) that is compensated for by significantly reduced power consumption.

Understanding and precisely quantifying the causes for the performance and energy differences between these platforms is typically done with detailed simulation, a laborious and computationally demanding task that is difficult to scale to broad, realistic workloads of full applications. This work is the first to leverage a sophisticated statistical modeling framework called *Partial Least Squares (PLS) Path Modeling* [38], [56] to overcome these limitations. PLS Path Modeling is a technique used in computational statistics for quantifying cause-effect relationships between *latent concepts* – conceptual elements that underpin a complex system but are difficult to directly measure. These latent concepts manifest themselves as *latent variables* in a PLS Path Model, which are used to quantify the cause-effect relationships between latent concepts.

In this work, we use observable application characteristics – e.g., cache hit rates and access counts – as indicators of some underlying concept of interests – e.g., effectiveness of last level cache in improving application performance. Constructing PLS Path Models to encompass a host of such characteristics allows us to rigorously quantify the effectiveness of a rich set of architectural units and subsystems, exposing those subsystems that should be the focus of hardware and software designers in improving application performance. Our models, validated against empirical measurements, suggest that the memory subsystem and prefetching are the primary culprits for these differences. The specific contributions of this work are:

1) **Multi-platform Characterization —** we present a detailed performance, power and energy characterization of a large set of applications on the AppliedMicro X-Gene, the first available 64-bit ARMv8 platform, as well as three other important architectural design points from Intel. This characterization covers a wide spectrum of HPC and architectural workloads, providing a number of insights into the prospects of X-Gene for meeting the performance and energy efficiency needs of upcoming HPC system designs (Section IV).
2) **Publicly Available Measurements —** we make our raw measurement data available to the public[1], providing other researchers and practitioners with performance, power and hardware counter measurements to supplement their own efforts to understand HPC application performance and energy consumption across a wide range of applications and several interesting computing platforms.
3) **Partial Least Squares Path Modeling —** we describe a novel application of PLS Path Modeling, a statistical framework for rigorously quantifying latent concepts, to quantify and rank the importance of performance bottlenecks (Section V). To the best of our knowledge, this work is the first to apply PLS Path Modeling to this problem. We use the PLS-PM framework to derive conclusions about the specific characteristics of the X-Gene platform that limit HPC application performance, providing directions for hardware designers, compiler writers and application developers to take to improve its performance and energy consumption (Section VI).

II. Key Findings

We use a comprehensive set of workloads to study the performance of four architectural design points, including the AppliedMicro X-Gene, a 64-bit ARMv8 platform (see Table I). We find that the X-Gene is an average of 2.3× slower than an Intel Sandy Bridge configured to reflect a design common in today's HPC deployments. We find also that the X-Gene is 3.4× slower than an Intel Haswell, representative of upcoming high-performance HPC deployments. Finally, the X-Gene has slightly better performance than an Intel Atom, running an average of 7% faster.

The X-Gene consumes slightly less energy than the Intel Sandy Bridge (Sandy Bridge consumes 1.2× energy on average) due to slower execution on the X-Gene that is compensated for by significantly reduced power consumption. Additionally, the X-Gene consumes 1.3× the energy of the Intel Haswell due to a large gap in power consumption between the two (Haswell draws 2.6× power than X-Gene) but an even larger performance gap (Haswell is 3.4× faster). Finally, the X-Gene consumes significantly more energy than the Intel Atom (3.5×), as the two show roughly comparable performance while the Atom consumes far less power.

In comparing the X-Gene to the Sandy Bridge, we find that improvements in the prefetching and memory subsystems on the X-Gene are critical to improving HPC application performance. Additionally, we find that other architectural factors such as branch predictors and the CPU frontend work in favor of the X-Gene but are less critical for HPC workloads, and thus are not likely to be large sources of improvement.

[1] http://epanalytics.com/data/ispass2016/

III. Methodology

This section details the platforms, applications, measurement techniques and methodological considerations for our study.

Platforms – we compare four architectural designs. First, we use an 8-core Intel Xeon E5-2670v1 (Sandy Bridge) [34], a processor characteristic of the typical processor found in HPC compute nodes at the time of this writing (e.g., it appears in 7 of the top 20 systems in the current Top500 list [4]). Second, we use a representative of the coming wave of 64-bit ARMv8 implementations: an HP Moonshot ProLiant m400, containing an 8-core AppliedMicro X-Gene (883208-X1) processor [6]. Third, we use an 8-core Intel Atom C2758 [35], representative of a low-power alternative to the ARMv8. Finally, we use an 8-core Intel Xeon E5-2667v3 (Haswell) [36], representative of a configuration we might expect to find in HPC installations in the near future. A detailed listing of the configurations of these platforms is presented in Table I.

Applications – we employ a host of test cases that cover a broad range of application domains where ARMv8 has been of intense interest. First, we use a series of single- and multi-threaded workloads typically used in the domain of computer architecture, including applications from the SPEC CPU2006 [32] and PARSEC [12] benchmark suites. In addition, we use a number of workloads from the high performance computing (HPC) domain, including applications from the NAS Parallel Benchmarks (NPBs) [8], polybench [47], Mantevo [33], Trinity [22] and Coral [2]. Finally, representing high performance embedded computing (HPEC), we use Coremark [24] and the DARPA PERFECT benchmarks [11]. The parallelization strategies, inputs, floating point model and details about how applications are configured are summarized in Table II. To characterize the floating point model of each application, we use a combination of benchmark documentation, manual inspection and binary-level analysis tools (PEBIL on x86 [40] and EPAX [25] on ARMv8). In total, our applications represent a diverse range of characteristics, with over 400 unique configurations being used.

Power and Performance Measurement – system-level power measurements for the Intel servers are collected using a WattsUp meter [1]. For HP Moonshot server housing the X-Gene processors, we use the Moonshot's integrated Lights Out (iLO) facilities [3] to measure node-level power. We run each application in a loop to guarantee at least 60 seconds of execution when collecting power measurements. Furthermore, because the surrounding server components within each of our test platforms (motherboards, fans, I/O devices, etc.) differ, we separate out the power resulting from those components and isolate only the power resulting from running the application. To accomplish this we collect system-level power during each application run, then isolate the power consumption due to running the application by subtracting idle server power from that measurement. Execution time is measured as the average end-to-end runtime of the application. For all test cases, threads

TABLE I
PLATFORM CONFIGURATIONS

	Intel Sandy Bridge	AppliedMicro X-Gene	Intel Atom	Intel Haswell
Instruction Set Architecture	x86_64 (64-bit CISC)	ARMv8 (64-bit RISC)	x86_64 (64-bit CISC)	x86_64 (64-bit CISC)
CPU	Intel E5-2670v1 Sandy Bridge	AppliedMicro 883208-X1	Intel C2758 Atom	Intel E5-2667v3 Haswell
Data Cache (n-way shared)	32KB L1 (1), 256KB L2 (1), 20MB L3 (8)	32KB L1 (1), 256KB L2 (2), 8MB L3 (8)	24KB L1 (1), 1MB L2 (2)	32KB L1 (1), 256KB L2 (1), 20MB L3 (8)
Instruction Cache (n-way shared)	32KB L1 (1)	32KB L1 (1)	32KB L1 (1)	32KB L1 (1)
Memory	32GB 1600MHz	16GB 1866MHz	32GB 1600MHz	128GB 2133MHz
Server Idle Power (W)	117	37	29	93
Chip TDP (W)	115	43.5	20	135
FP/Vector Support	AVX (256-bit)	Neon (128-bit)	SSE4.2 (128-bit)	AVX2 (256-bit)

TABLE II
APPLICATIONS, INPUTS AND CONFIGURATIONS

Domain	Suite	Applications	Input/ Configuration	Core Counts	Parallelization
HPC	NAS Parallel Benchmarks	block tridiagonal[1], conjugate gradient[2], embarrassingly parallel[2], fourier transform[2], integer sort[2], multigrid[2], lower-upper gauss-seidel[2], scalar penta-diagonal[1]	A, B and C	[1]1 and 4 [2]1 and 8	MPI and OpenMP
	PolyBench	adi, atax, bicg, cholesky, covcol, dct, doitgen, dsyr2k, dsyrk, dynprog, fdtd2d, fdtdapml, gemver, gesummv, gramschmidt, jacobi2dimper, matmulinit, mm, mvt, seidel, ssymm, stencil3d, strmm, strsm, swim, tce, tmm, trisolv	L1, L2, L3 and MM; SP and DP	1	serial
	Mantevo	CoMD, miniMD	2000 steps; SP and DP	1 and 8	serial, OpenMP and MPI
	Trinity	GTC[3], MiniFE[4], MiniGhost[5], stream[3]	SP and DP; [3]default [4]120x120x120 [5]500steps	1 and 8	serial, OpenMP and MPI
	Coral	AMGmk, MILCmk	default; SP and DP	1 and 8	serial, OpenMP and MPI
Arch.	SPEC CPU2006	perlbench, bzip2, gcc, bwaves, gamess, mcf, milc, zeusmp, cactusADM, leslie3d, namd, gobmk, soplex, calculix, hmmer, sjeng, GemsFDTD, h264ref, tonto, omnetpp, astar, xalancbmk	reference	1	serial
	PARSEC	blackscholes, bodytrack, facesim, ferret, fluidanimate, swaptions	native	1 and 8	serial and pthreads
HPEC	coremark	matrix, linked list, state machine, cyclic redundancy check	size=666; iter=2e5	1	serial
	PERFECT	SAR: pfa-interp1[8], pfa-interp2[8], bp[6] PA1: dwt53[7], 2d convolution[7], histogram equalization[7] STAP: outer product[8], system solve[8], inner product[8] WAMI: lucas kanade[8], debayer[8], change detection[8] OTHER: 1d fft[8], 2d fft[6], sort[8]	[6]small [7]medium [8]large	1	serial

and processes are pinned to cores to prevent migration. All performance and power figures presented are the arithmetic mean of 5 measurements.

Hardware Performance Counters – in addition to performance and power measurements, we make additional application runs to collect every hardware performance monitor available through Linux's `perf` interface on the Sandy Bridge as input to the architectural bottleneck model described in Section V. `perf` exposes 403 such counters, including detailed counts within the on- and off-chip caches, branch predictions/outcomes, CPU frontend/backend, TLB, power settings, on-chip network and memory controller. Our approach to collecting 403 counters per test case is to make many runs of the test case, collecting a small number of counters during each run (i.e., we do not use multiplexing). Similarly, we use the `perf` interface to collect 128 performance counters on the X-Gene.

Level Playing Field – to ensure that each of the test platforms is judged on as level a playing field as possible, we use an identical software test harness to make measurements on all systems. Each processor has 8 cores; for serial tests exactly 1 of those 8 cores is used during the test, and for multi-threaded (pthreads, OpenMP) and multi-process (MPI) tests all 8 cores are employed. A handful of the NAS Parallel Benchmarks require core counts that are squares (1, 4, 9, etc.). For these cases, we run the application on 4 cores instead of 8. The Sandy Bridge and Haswell platforms support 2-way simultaneous multi-threading (SMT), however in the interest of creating a fair core-to-core comparison we disable SMT. To the extent possible, we use similar memory configurations across platforms. While memory capacity varies significantly across systems – from 16GB in the X-Gene to 128GB in the Haswell – all of our test applications have working set sizes that fit into main memory on X-Gene, and we expect that differences in capacity have little impact on performance for our test applications.

Applications are run in identical configurations, and are built with similar software support libraries: OpenMPI is used for MPI applications, GCC 4.8.x is used for compilation, and `-O3` optimization is supplied when compiling all benchmarks. We

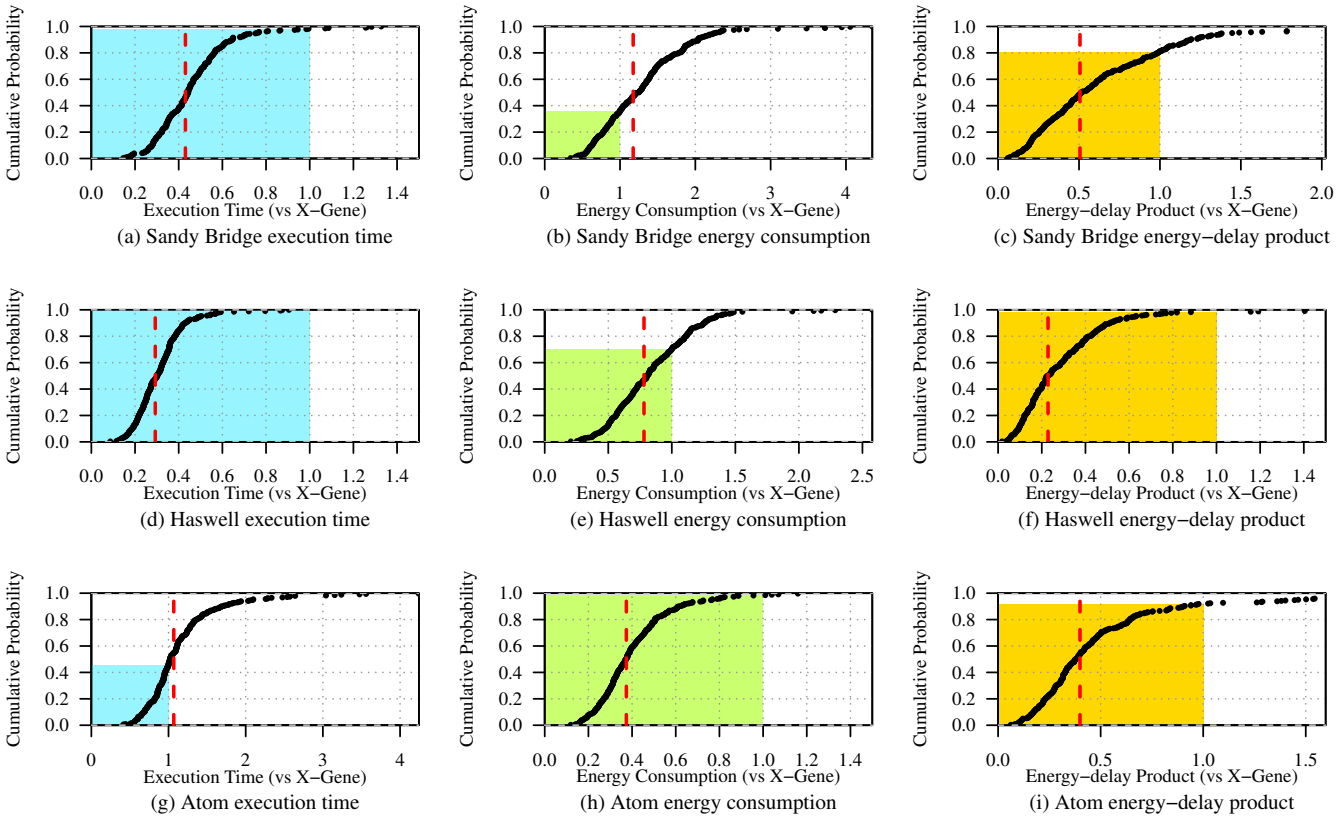


Fig. 1. Cumulative distributions of execution time, energy consumption and EDP of the X-Gene compared to three Intel platforms for 400+ test cases; shaded regions indicate tests for which the X-Gene is outperformed on the given metric and vertical red dotted lines show the average across all applications

note, however, that the ARMv8 architecture is of a much more recent vintage than the x86_64 Intel platforms, thus the maturity of compiler optimizations is not perfectly comparable between the two despite the similarity in compiler versions. Nevertheless, the compiler is part of the current hardware/software stack available for the X-Gene, and its maturity level is an integral part of our characterization. We discuss the implications of future compiler improvements further in Section VI.

IV. Characterization Results

This section presents the highlights of our characterization study that spans over 400 application configurations and four architectural design points. We also invite readers interested in getting a deeper look at the data or supplementing their own characterization efforts to find the complete data set at `http://epanalytics.com/data/ispass2016/`.

A. Performance and Energy Consumption

We first present a comparison of the execution times, energy consumption and energy-delay product (EDP) of applications on our four test platforms. EDP is calculated as the product of normalized execution time squared and normalized power draw, incorporating both power and performance but giving more weight to performance. The data at the center of this comparison is presented in Figure 1, which shows cumulative distributions of application execution time, energy consumption and EDP of the Sandy Bridge, Haswell and Atom. In each case, the metric is normalized to the measured metric on the X-Gene. Values less than 1, denoted by shaded regions, indicate those cases where the X-Gene is outperformed by the other system on the given metric. Each plot also contains a dotted vertical red line showing the geometric mean of the distribution.

Figure 1(a) shows the execution time on the Sandy Bridge. The first trend we observe is that on average, execution time on the Sandy Bridge is 43% of execution time on the X-Gene. Furthermore, in excess of 99% of the test cases run faster on the Sandy Bridge than on the X-Gene. There is significant variability, however, in this reduction; 90% of the test cases fall in the 25%-70% range. The typical case is for the Sandy Bridge to outperform the X-Gene by a factor of 2.3×. However, the X-Gene consumes about 2× less power than the Sandy Bridge, resulting in slightly lower energy consumption in the X-Gene (Figure 1(b)) and significantly higher EDP (Figure 1(c)). There is substantial variation around these averages, however, and the energy consumption characteristics depend on a number of factors that we explore in more depth in Section V.

A comparison of the X-Gene to the Haswell yields qualitatively similar conclusions about performance. Figure 1(d) shows that Haswell outperforms X-Gene, reducing execution time on average to 29% of execution time on the X-Gene and 90% of test cases show execution times of between 18%-49%. Haswell energy consumption, as seen in Figure 1(e), is reduced to an average of 79% of X-Gene energy consumption due in large part to the large improvements in execution

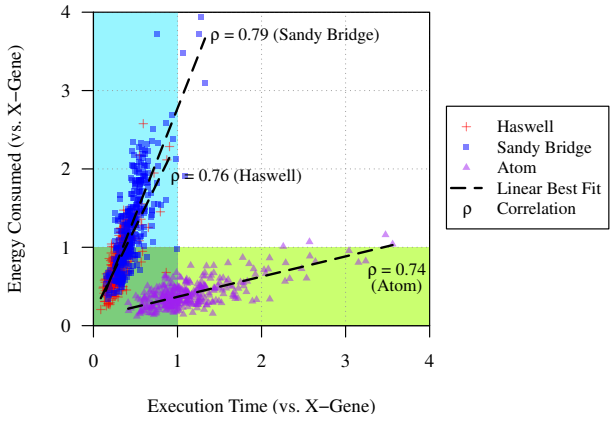


Fig. 2. Comparison of execution time to energy consumption; black dotted lines show linear regression (best fit) lines, annotated with the energy/performance correlation ρ

time. Moreover, 70% of test cases show energy consumption improvements on the Haswell.

The Atom differs significantly from the Sandy Bridge and Haswell. First, Figure 1(g) shows that the performance of the Atom is roughly comparable to that of the X-Gene, where average execution time on the Atom is 107% of the execution time on the X-Gene, and thus the Atom is outperformed slightly by the X-Gene. 58% of our test cases are slower on the Atom while the other 42% are faster. Second, as seen in Figure 1(h), the energy consumption is virtually always less on the Atom than on the X-Gene, and in the vast majority of cases reduced significantly. This occurs because the performance is similar and the power consumption of the Atom is far lower; the typical power consumption on the Atom is around 2.5× lower than the X-Gene, which is also reflected by a roughly 2.5× difference in EDP between the two platforms (Figure 1(i)).

B. Performance is Fundamental

Figure 2 presents a comparison of execution time (x-axis) and energy consumption (y-axis) for all three systems. This figure contains three data series, comparing the characteristics of individual application runs on the X-Gene to those on the three Intel systems. Each point in the graph represents a single application, where the coordinates of the point are the execution time and energy consumption for the application normalized to the X-Gene system. Like Figure 1, all measurements are normalized to the X-Gene measurements and values less than 1 indicate test cases where the X-Gene is outperformed by the other platform. These cases are highlighted as shaded regions in the graph – blue where performance is better than X-Gene and green where energy consumption is lower than X-Gene. For example, a point in the Haswell series at (0.5,0.9) would indicate that the execution time of one of the test applications on the Haswell was 50% of the execution time on the X-Gene, while the energy consumed on the Haswell is 90% of the energy consumption on the X-Gene.

Notable is the fact that there is a strong relationship between normalized execution time and normalized energy consumption. In fact, the correlation between these two metrics is 0.74, 0.76 and 0.79 for the Atom, Haswell and Sandy Bridge, respectively.

TABLE III
LATENT VARIABLES IN THE PLS PATH MODEL

Latent Variable	Description
BRANCH	Execution and prediction of branches
CACHE LLC	Last level cache
CACHE L1/L2	Level 1 and 2 caches
FP+SIMD	Floating point and SIMD operations
FRONTEND	Register alias table and reservation station
INSN FETCH	Instruction fetch and decode
MEMORY OP	Memory operations
PREFETCH	Data and instruction prefetching
PWR STATE	C-states and P-states
COHERHENCE	Cache coherence events
TLB	Translation lookaside buffer

We highlight this view of the data primarily to show that *execution time is a key component of energy consumption and that if we want to understand both normalized performance and energy consumption, focusing on performance is sufficient to tell us most of what we need to know*. This is broadly consistent with the literature on optimizing software for energy consumption, which has shown that the most effective technique for energy optimization is usually to enact performance optimization [9], [58], [59].

V. BOTTLENECK ANALYSIS

To understand the causes of these observed performance differences between platforms, we quantify and rank the performance bottlenecks on the X-Gene platform via a technique called Partial Least Squares (PLS) Path Modeling [10], [19], [26], [51]. A commonly used approach for understanding bottlenecks is to use detailed architectural simulation, which is difficult to do accurately [15], [30], [44] in addition to being prohibitively slow to be applied to full-scale applications [13], [16]. This section shows how PLS Path Modeling can be used to quantify bottlenecks on real systems and full applications.

A. PLS Path Modeling

PLS Path Modeling is a structural modeling technique quantifying the relationships between latent concepts in complex systems. A latent concept, embodied by a latent variable in a PLS Path Model, represents a concept underlying the behavior of a complex system. In this section, we describe a novel application of PLS Path Models in assessing the effectiveness of architectural units at improving application performance.

1) Model Components: The components of our PLS Path Model are depicted in Figure 3. In the model there are two types of nodes, along with edges of various types connecting them. Here we explain the fundamentals necessary to understand the construction of a PLS Path Model.

Latent Variables – illustrated as boxes are a series of inner nodes, each one representing a latent variable of interest to us. Our model uses latent variables representative of a number of architectural units and subsystems, including various levels of cache, TLB, instruction fetch, floating point/SIMD, and so forth. The complete set of subsystems covered by our model is given in Table III. Finally, we define a latent variable

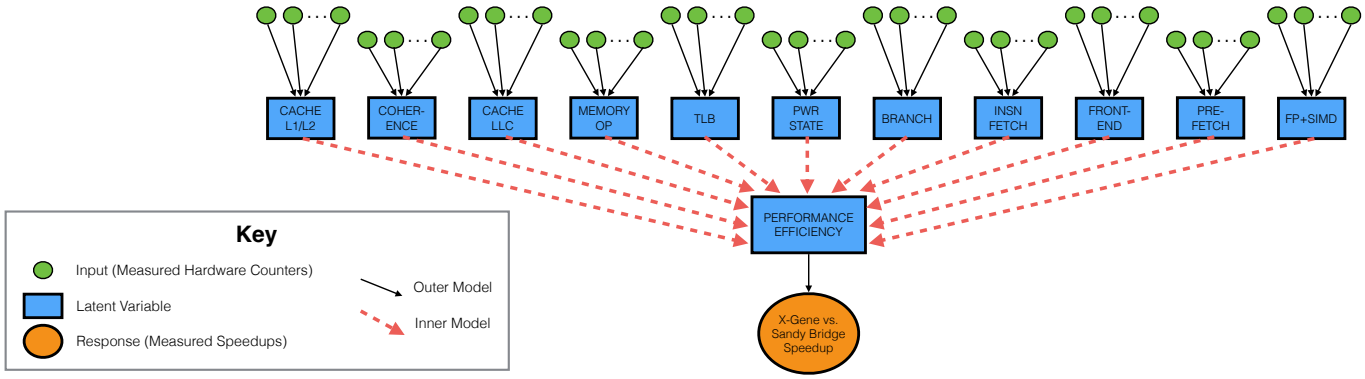


Fig. 3. Structure of PLS Path Model for performance efficiency, which links hardware counter measurement inputs to the concept of performance efficiency via latent variables linked together through the inner model; this work focuses on the weights produced for the inner model (thick red arrows), which quantify how those latent variables impact performance efficiency

representing the performance efficiency of the X-Gene platform relative to the Sandy Bridge platform.

Inner Model – the relationship between inner nodes, forming the inner model, is represented in the model as a directed acyclic graph (DAG). The directed edges between inner nodes, shown as dotted red lines in Figure 3, represent the causal relationship between the latent variables, and will automatically be assigned a coefficient during construction of the model that represents the direction and strength of the causal relationship. In our model, we define the causal relationships of this form as increasingly effective architectural units in the Sandy Bridge platform causing improved performance in the Sandy Bridge relative to the X-Gene.

Inputs and Response – the model contains a series of outer nodes representing two types of measured variables. The first type is the response variable, which we define as the execution time of each application test case on the X-Gene normalized to the execution time on the Sandy Bridge. The second are an extensive series of hardware counter measurements taken on the Sandy Bridge system during the run of each test case. As described in Section III, there are 403 hardware counters available on the Sandy Bridge; we collect all of them, then remove those that are highly correlated with others in order to make use of the 45 most informative counters in the model. These counters are standardized by dividing them all by the total instruction count of the application (i.e., branch mispredictions becomes branch mispredictions per instruction).

Outer Model – the measured input and response variables are connected to the latent variables to form an outer model. For input variables, these connections are made such that each latent variable is connected to a set of input measurements. For our model, each input measurement is a hardware counter measurement that partially reflects the activity within or the effectiveness (or lack thereof) of a particular architectural unit. For instance, the CACHE L1/L2 variable is associated with measurements regarding the L1 and L2 caches: number of L2 store misses, dirty cache lines evicted from L2 cache, cycles with outstanding L2 load misses, and so forth. By associating the relevant set of hardware counter measurements with each architectural unit we are able to encapsulate the relationship of those measurements to the associated latent variable. In turn, the edges in the inner model between latent variables and the response variables encapsulate the effect of each variable on overall performance.

2) Model Output: Once the structure of a PLS Path Model is set, the model is built using the PLS algorithm. The core of the PLS algorithm is the calculation of a set of weights for the linear combination of input variables used to estimate each latent variable. This is achieved via an iterative procedure that alternates between two kinds of estimations on the latent variables until those estimates converge. A detailed description of the statistical underpinnings and mechanical operation of the PLS algorithm is beyond the scope of this paper, and we refer interested readers to an introductory PLS Path Modeling textbook [10], [19], [26] or tutorial-level treatment of the subject [51] to learn more.

The output of a PLS Path Model contains, among other things, an assignment of coefficients to the inner edges of the graph, which define both the direction and strength of the relationships between the latent concepts in the graph. Each connection is assigned either a positive coefficient, indicating a positive causal relationship (e.g., a positive coefficient on the edge between TLB and Performance Efficiency indicates that a more effective TLB causes improved performance efficiency on the Sandy Bridge relative to the X-Gene), or a negative coefficient, indicating a negative causal relationship.

3) Bootstrap Confidence Intervals: To assess the statistical significance of the coefficients on edges we use a technique known as *bootstrapping*. We first create a bootstrap sample set by randomly sampling the input and response data with replacement to obtain a sample with size equal to the original (duplicates are highly likely). We then derive coefficient estimates using the bootstrap sample, repeating this process for 500 bootstrap sample sets to derive a probability distribution for each of the coefficients. From these distributions, we then derive empirical confidence intervals for the coefficients. For example, if we suppose that 95% of the bootstrap coefficient estimates for some edge are between 0.1 and 0.3, (0.1,0.3) is our 95% bootstrap confidence interval.

4) Model Generation: Our PLS Path Model includes 45 input variables and an output variable for over 400 test cases, as well as 12 latent variables. Constructing the model, assigning

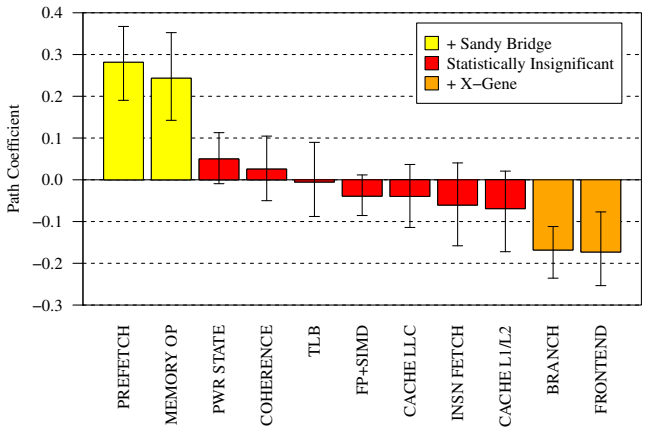


Fig. 4. Weight on PLS Path Model latent variables; yellow bars (left side) indicate those latent variables related to improved performance on the Sandy Bridge over the X-Gene; error bars show confidence intervals at 95%

coefficients, and building confidence intervals requires just a few seconds of execution on a desktop machine, and the vast majority of this time is spent building the 500 bootstrap estimates required to derive confidence intervals.

B. Model Results

With the PLS Path Model in hand, we present the resulting path coefficients in Figure 4. The height of each bar shows the path coefficient produced by the model for the corresponding latent variable, and the error bars represent the 95% confidence interval found for each coefficient. The error bars are of critical importance in interpreting the results; for those latent variables with error bars that encompass a coefficient of zero, we cannot conclude the sign of the coefficient with certainty, precluding us from making any conclusions about those variables.

The figure is organized to show three classes of results. In yellow bars (left and above zero), we depict those latent variables with positive path coefficients, which are associated with architectural units whose efficacy causes performance improvements on the Sandy Bridge relative to the X-Gene. In other words, applications that rely heavily on these architectural units or subsystems see better performance on the Sandy Bridge over the X-Gene platform. These units are PREFETCH and MEMORY OP, which both closely relate to memory hierarchy and how effectively data moves through it.

We next focus on the bars shown in orange (right and below zero). These architectural units have a negative coefficient, meaning that applications relying heavily on these units have relatively better performance on the X-Gene over the Sandy Bridge. These variables include FRONTEND and BRANCH.

Finally, we point out that the other latent variables – FP+SIMD, TLB, COHERENCE, CACHE LLC, INSN FETCH and CACHE L1/L2 – are shown in red (middle of chart with confidence interval encompassing zero) and have statistically insignificant results on the sign of the coefficient, and we therefore make no conclusions about their impact on performance.

C. Validation

To further understand the impact of these bottlenecks on our test applications, we examine how dependent the test cases are on the units causing the bottlenecks. We do this by classifying each test case as having low, medium or high intensity for memory and branch operations, as memory and branch operations are (1) associated with characteristics identified by the PLS Path Model as being linked to performance differences between the X-Gene and Sandy Bridge and (2) their activity is easy to observe in our hardware performance counter data.

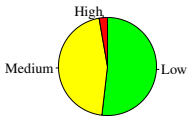
- **Memory Intensity —** we use the hardware counter measurements to compute the prevalence of memory operations in terms of memory operation count / instruction count. We classify low-intensity (Low) applications as those with fewer than 0.2 memory ops per instruction on the Sandy Bridge, medium intensity (Medium) applications as those with between 0.2 and 0.4 memory ops per instruction, and high intensity (High) applications as those with 0.4 or more memory ops per instruction.
- **Branch Intensity —** similarly, we partition test cases according to branch intensity, classifying those with fewer than 0.1 branches per instruction on the Sandy Bridge as having Low branch intensity, those with between 0.1 and 0.2 branches per instruction as having Medium intensity, and those with greater than 0.2 branches as having High intensity. Note that this definition of branches encompasses all varieties of branches – loop iterators, predicates (if, if/else, switch, etc.) and calls.

The breakdown of applications among these categories for our test cases is shown in Figure 5. The key observations we can make from categorizing the test cases this way is that test cases with medium or high memory intensity are the norm, as more than 97% of test cases have memory operations as 20% or more of their instructions. Similarly, applications with low or medium branch intensity are the norm; branch instructions account for no more than 10% of all instructions for more than 97% of test cases.

These characteristics have a significant impact on the performance and energy consumption of application code. Figure 6 shows how low, medium and high intensity in these characteristics translates to performance and energy characteristics between the X-Gene and Sandy Bridge. This comparison is presented as a series of violin plots in Figure 6, where 6(a) shows execution time on the X-Gene relative to the Sandy Bridge (less than 1 means better execution time than the X-Gene), 6(b) shows energy consumption on the X-Gene relative to the Sandy Bridge (less than 1 means lower energy consumption than the X-Gene) and 6(c) shows energy-delay product on the X-Gene relative to the Sandy Bridge. An individual violin can be interpreted as a probability distribution of values across a particular set of applications, where the violin is “fatter” at the places the distribution has a higher density and the white dot represents the mean of the distribution. The violins from among the different groupings can thus be compared by looking at their shape, position and mean to determine how application attributes impact the performance and energy characteristics.

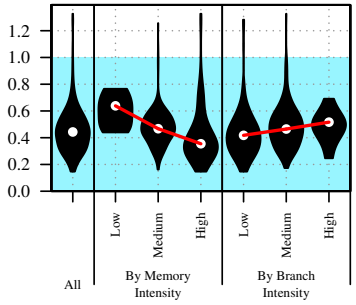
Partitioning the results in these ways is instructive. For the execution times in 6(a), the normalized execution time

(a) Memory Intensity

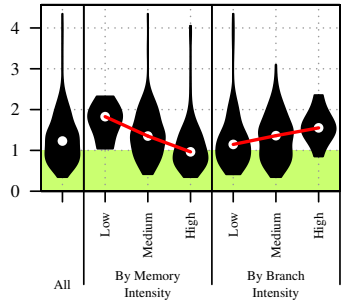


(b) Branch Intensity

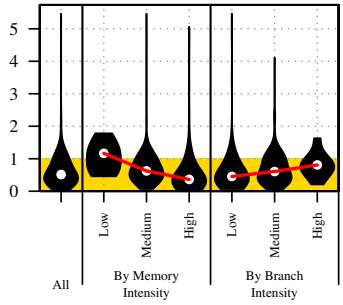
Fig. 5. Measured HPC workload composition



(a) Execution time



(b) Energy consumption



(c) Energy–delay product

Fig. 6. Execution time, energy consumption and EDP for various classes of application test cases; results are normalized to the X-Gene measurements, and shaded regions indicate test cases where the X-Gene is outperformed by the Sandy Bridge

of X-Gene can be seen to get worse (lower) as a function of increasing memory intensity, while the reverse is true of branch intensity. Similarly, energy consumption and energy-delay product in 6(b) and 6(c), respectively, exhibit similar trends as branch and memory intensity increase. For those tests with low branch intensity, energy consumption of the Sandy Bridge is only 11% higher than the X-Gene, while at high branch intensity it is 53% higher. Likewise, from low to high memory intensity, execution times on the X-Gene shift from 59% to 38% of the Sandy Bridge execution times.

The identification of these trends in simple terms provides confirmation of some of the conclusions reached by the more sophisticated PLS Path Models, in that increased memory activity is associated with diminished performance on the X-Gene platform relative to the Sandy Bridge while increased branch intensity is associated with diminished performance on the Sandy Bridge relative to the X-Gene.

VI. Discussion

Beyond the quantitative results and conclusions presented thus far, there are a variety of factors and ongoing trends that will influence the ARMv8's impact on HPC and its effectiveness as a design point.

Accelerators – ARM's licensing model makes tightly integrating accelerators with ARMv8 implementations a far more feasible task than doing so on other platforms. These accelerator-based designs offer the potential for huge performance and energy-efficiency gains, especially for particular classes of algorithms and computational patterns [27], [43], [50], [52]. Thus, even if the X-Gene and other ARMv8 implementations on their own are outperformed by other design points, the ARMv8 may be the right design point from a hardware engineering standpoint when accelerators are considered.

Software Stack – the system software stack on the ARMv8 is relatively immature. For example, GCC continues to add numerous ARMv8 performance improvements with each successive version [55], with many more optimizations planned for the near future [29]. These improvements could close the gap between the ARMv8 compiler and its x86 counterpart and improve ARMv8 performance.

Algorithm and Software Design – algorithms and software are often explicitly or implicitly designed to achieve high performance in the common case, which has been high-performance x86-based platforms in recent years. Everything from high-level algorithmic changes to low-level software architecture and coding practices that "re-customize" software for the X-Gene could expose significant performance opportunities. For instance, lower computational performance on the X-Gene may expose more opportunities to take advantage of overlapping computation and communication in communication-heavy applications, leading application developers to adopt non-blocking communication primitives to take advantage of this opportunity.

Many-core Implementations – it is anticipated that many-core implementations of the ARMv8 will become available as retail products in 2016 [18]. These implementations will pack many more cores (reportedly up to 48) into each processor, which will tightly couple a large number of cores together and allow large-scale applications to reduce or eliminate node-to-node network traffic, potentially making these many-core implementations more scalable than the AppliedMicro X-Gene used in this study. However, many-core implementations are a double-edged sword. Packing more cores onto a node may dramatically increase pressure on memory, which is a common bottleneck in HPC application performance [17], [41].

Circuits & Process – the AppliedMicro X-Gene used in this study is based on 40nm process technology, however future ARMv8 implementations are likely to be based on more aggressively scaled technology nodes. For example, X-Gene 2 and the AMD Opteron A1100 both use 28nm technology [53], [60]. We expect that these process technology improvements, as well as continued optimization of circuits and logical blocks, to offer improvements in power consumption and performance in future ARMv8 implementations.

Impact of Performance/Power Optimization – we quantify the impact of improvements in the performance and power draw of the X-Gene by presenting a model in which we scale the performance and power draw of our test measurements. Figures 7, 8 and 9 present heat maps, where a position on the map corresponds to particular improvements in the power draw (x-axis) and performance (y-axis) of the X-Gene platform. The color at each point represents the energy-delay product of the X-Gene normalized to the other platform (lower EDP values correspond to better EDP on the X-Gene). Using a black line, we also highlight the performance/power improvement curve

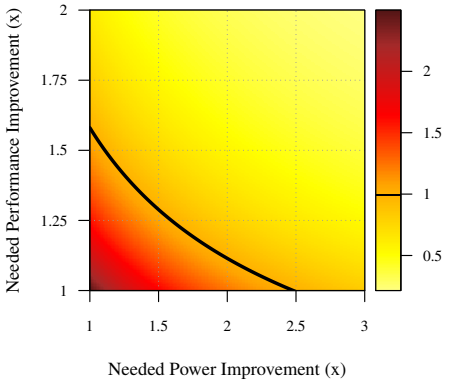


Fig. 7. Energy-delay product projections of X-Gene vs. Atom

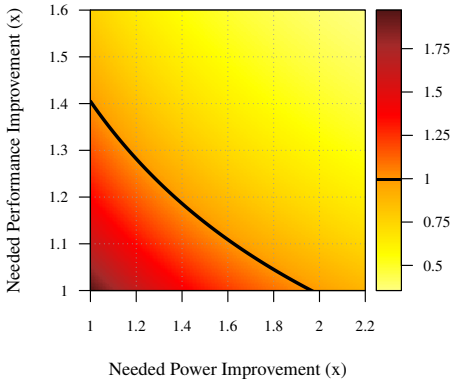


Fig. 8. Energy-delay product projections of X-Gene vs. Sandy Bridge

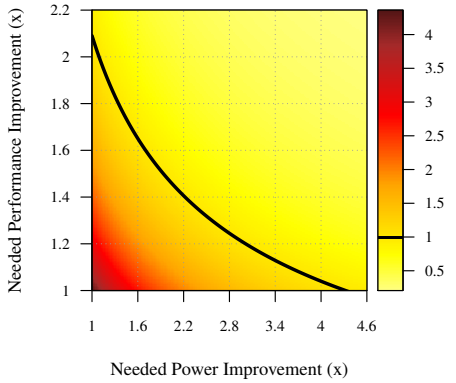


Fig. 9. Energy-delay product projections of X-Gene vs. Haswell

necessary for the X-Gene to achieve EDP parity with the other platform. For example, Figure 8 shows that the X-Gene could improve power consumption by 2× or performance by 1.4× to achieve EDP parity with the Sandy Bridge.

VII. Related Work

There have been a number of works investigating the potential of employing ARM chips in the server market [21], [54] and on scientific computing codes [5], [20], [37], [41], [45], [39], [57]. Padoin et al. describe the performance/energy trade-off of a 32-bit ARM big.LITTLE against an Intel Sandy Bridge on the NPBs, focusing on compiler options and using a different method to collect energy from each system [46]. They find that while the Sandy Bridge processor draws far more power, the ARM is not much more energy efficient due to its performance.

The Barcelona Supercomputing Center's Mont-Blanc project is actively developing an HPC prototype based on ARM cores. This project has produced results on a testbed cluster based on 32-bit ARMv7 processors and a 1GbE network, highlighting the need for an optimized software stack and high-performance network in achieving competitive performance [48].

Our work differs from earlier studies in several ways. While others have focused on the 32-bit ARMv7 platform or on narrow application domains, our work is the first to study the power, performance and energy characteristics for the 64-bit ARMv8 platform for a broad mix of applications. We also use the statistically principled technique of Partial Least Squares Path Modeling to quantify the performance bottlenecks of the X-Gene, a 64-bit ARMv8 platform, providing concrete directions for hardware designers, compiler writers and application developers to improve performance and energy consumption.

VIII. Conclusion

In this paper we present the first detailed characterization and architectural bottleneck analysis of X-Gene, a 64-bit ARMv8 platform. Through a detailed analysis covering four architectural designs and hundreds of application test cases, we show that the performance of the X-Gene is roughly on par with an Intel Atom and that the energy consumption is roughly on par with an Intel Sandy Bridge. We also develop a novel application of a sophisticated statistical modeling technique known as Partial Least Squares Path Modeling to expose the architectural bottlenecks on the X-Gene, revealing that the memory subsystem is one of the key architectural bottlenecks limiting performance and energy consumption for HPC applications.

Acknowledgements

This work was supported in part by the Air Force Office of Scientific Research under AFOSR Award No. FA9550-12-1-0476 and by DoE SBIR Award No. DE-SC0013164. This work was also supported in part by the Department of Defense High Performance Computing Modernization Program's PETTT program (Contract No: GS04T09DBC0017 through Engility).

References

[1] Watts up? PRO ES. https://www.wattsupmeters.com/secure/products.php?pn=0&wai=212&spec=3. [Online; accessed 05-Oct-2015].

[2] CORAL Benchmark Codes. https://asc.llnl.gov/CORAL-benchmarks/, 2013. [Online; accessed 05-Oct-2015].

[3] Moonshot Moves HPC Closer to ARM's Reach. http://www8.hp.com/us/en/products/servers/ilo/, 2015. [Online; accessed 05-Oct-2015].

[4] Top 500 Supercomputing Sites. http://top500.org/, 2015. [Online; accessed 05-Oct-2015].

[5] D. Abdurachmanov, B. Bockelman, P. Elmer, G. Eulisse, R. Knight, and S. Muzaffar. Heterogeneous high throughput scientific computing with apm x-gene and intel xeon phi. *arXiv preprint arXiv:1410.3441*, 2014.

[6] AppliedMicro. X-Gene. https://www.apm.com/products/data-center/x-gene-family/x-gene/, 2015. [Online; accessed 05-Oct-2015].

[7] R. V. Aroca and L. M. G. Gonçalves. Towards green data centers: A comparison of x86 and arm architectures power efficiency. *Journal of Parallel and Distributed Computing*, 2012.

[8] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, et al. The nas parallel benchmarks. *International Journal of High Performance Computing Applications*, 1991.

[9] P. Balaprakash, A. Tiwari, and S. M. Wild. Multi objective optimization of hpc kernels for performance, power, and energy. In *Workhop on High Performance Computing Systems: Performance Modeling, Benchmarking and Simulation (PMBS)*. 2014.

[10] D. Barclay, C. Higgins, and R. Thompson. The partial least squares (pls) approach to causal modeling: Personal computer adoption and use as an illustration. *Technology studies*, 1995.

[11] K. Barker, T. Benson, D. Campbell, D. Ediger, R. Gioiosa, A. Hoisie, D. Kerbyson, J. Manzano, A. Marquez, L. Song, N. Tallent, and A. Tumeo. *PERFECT (Power Efficiency Revolution For Embedded Computing Technologies) Benchmark Suite Manual*. Pacific Northwest National Laboratory and Georgia Tech Research Institute, December 2013. http://hpc.pnnl.gov/projects/PERFECT/.

[12] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: Characterization and architectural implications. In *Parallel Architectures and Compilation Techniques (PACT)*, 2008.

[13] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 2011.

[14] J. Burt. ARM Signs 50th License for 64-Bit ARMv8-A Design. http://www.eweek.com/mobile/arm-signs-50th-license-for-64-bit-armv8-a-design.html, 2014. [Online; accessed 05-Oct-2015].

[15] A. Butko, R. Garibotti, L. Ost, and G. Sassatelli. Accuracy evaluation of gem5 simulator system. In *Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, 2012.

[16] T. E. Carlson, W. Heirman, and L. Eeckhout. Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2011.

[17] L. C. Carrington, M. Laurenzano, A. Snavely, R. L. Campbell, and L. P. Davis. How well can simple metrics represent the performance of hpc applications? In *Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2005.

[18] Cavium. ThunderX Processors. http://www.cavium.com/ThunderX_ARM_Processors.html, 2015. [Online; accessed 05-Oct-2015].

[19] W. W. Chin. The partial least squares approach to structural equation modeling. *Modern Methods for Business Research*, 1998.

[20] M. F. Cloutier, C. Paradis, and V. M. Weaver. Design and analysis of a 32-bit embedded high-performance cluster optimized for energy and performance. In *Hardware-Software Co-Design for High Performance Computing (Co-HPC)*, 2014.

[21] M. Coppola, B. Falsafi, J. Goodacre, and G. Kornaros. From embedded multi-core socs to scale-out processors. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2013.

[22] M. Cordery, B. Austin, H. Wassermann, C. Daley, N. Wright, S. Hammond, and D. Doerfler. Analysis of cray xc30 performance using trinity-nersc-8 benchmarks and comparison with cray xe6 and ibm bg/q. In *Workshop on High Performance Computing Systems: Performance Modeling, Benchmarking and Simulation (PMBS)*. 2014.

[23] Cray. CRAY to Explore Alternative Processor Technologies for Supercomputing. http://investors.cray.com/phoenix.zhtml?c=98390&p=irol-newsArticle&ID=1990117, 2014. [Online; accessed 15-April-2015].

[24] E. M. B. C. (EEMBC). CoreMark: an EEMBC Benchmark. https://www.eembc.org/coremark/, 2015. [Online; accessed 05-Oct-2015].

[25] I. EP Analytics. EPAX Toolkit: Binary Analysis for ARM. http://epaxtoolkit.com, 2014. [Online; accessed 05-Oct-2015].

[26] R. F. Falk and N. B. Miller. *A primer for soft modeling*. University of Akron Press, 1992.

[27] D. Goddeke, S. H. Buijssen, H. Wobker, and S. Turek. Gpu acceleration of an unmodified parallel finite element navier-stokes solver. In *High Performance Computing & Simulation (HPCS)*, 2009.

[28] J. Goodacre. Technology preview: The armv8 architecture. In *ARM White Paper*, 2011.

[29] M. Gretton-Dann. ARM Open Source Compiler Roadmap Update. March 2015 Webinar.

[30] A. Gutierrez, J. Pusdesris, R. G. Dreslinski, T. Mudge, C. Sudanthi, C. D. Emmons, M. Hayenga, and N. Paver. Sources of error in full-system simulation. In *International Symposium on the Performance Analysis of Systems and Software (ISPASS)*, 2014.

[31] N. Hemsoth. Moonshot Moves HPC Closer to ARM's Reach. http://www.hpcwire.com/2014/09/29/moonshot-moves-hpc-closer-arm-reach/, 2014. [Online; accessed 05-Oct-2015].

[32] J. L. Henning. Spec cpu2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 2006.

[33] M. A. Heroux, D. W. Doerfler, 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 Technical Report*, 2009.

[34] Intel. Intel Xeon Processor E5-2670. http://ark.intel.com/products/64595/Intel-Xeon-Processor-E5-2670-20M-Cache-2_60-GHz-8_00-GTs-Intel-QPI, 2012. [Online; accessed 05-Oct-2015].

[35] Intel. Intel Atom Processor C2758. http://ark.intel.com/products/77988/Intel-Atom-Processor-C2758-4M-Cache-2_40-GHz, 2013. [Online; accessed 05-Oct-2015].

[36] Intel. Intel Xeon Processor E5-2667 v3. http://ark.intel.com/products/83361/Intel-Xeon-Processor-E5-2667-v3-20M-Cache-3_20-GHz, 2014. [Online; accessed 05-Oct-2015].

[37] M. Jarus, S. Varrette, A. Oleksiak, and P. Bouvry. Performance evaluation and energy efficiency of high-density hpc platforms based on intel, amd and arm processors. In *Energy Efficiency in Large Scale Distributed Systems (EE-LSDS)*. 2013.

[38] K. G. Jöreskog and H. O. Wold. *Systems under indirect observation: Causality, structure, prediction*. North Holland, 1982.

[39] K. Keipert, G. Mitra, V. Sunriyal, S. S. Leang, M. Sosonkina, A. P. Rendell, and M. S. Gordon. Energy-efficient computational chemistry: Comparison of x86 and arm systems. *Journal of chemical theory and computation*, 11(11):5055–5061, 2015.

[40] M. A. Laurenzano, M. M. Tikir, L. Carrington, and A. Snavely. Pebil: Efficient static binary instrumentation for linux. In *International Symposium on the Performance Analysis of Systems & Software (ISPASS)*, 2010.

[41] M. A. Laurenzano, A. Tiwari, A. Jundt, J. Peraza, W. A. Ward Jr, R. Campbell, and L. Carrington. Characterizing the performance-energy tradeoff of small arm cores in hpc computation. In *European Conference on Parallel Processing (Euro-par)*, 2014.

[42] F. Mantovani. High performance computing based on embedded processors. In *High Performance Computing & Simulation (HPCS)*, 2014.

[43] J. Michalakes and M. Vachharajani. Gpu acceleration of numerical weather prediction. *Parallel Processing Letters (PPL)*, 2008.

[44] T. Nowatzki, J. Menon, C.-H. Ho, and K. Sankaralingam. gem5, gpgpusim, mcpat, gpuwattch, "your favorite simulator here" considered harmful. *Workshop on Duplicating, Deconstructing and Debunking (WDDD)*, 2014.

[45] Z. Ou, B. Pang, Y. Deng, J. K. Nurminen, A. Yla-Jaaski, and P. Hui. Energy-and cost-efficiency analysis of arm-based clusters. In *Cluster, Cloud and Grid Computing (CCGrid)*, 2012.

[46] E. L. Padoin, L. L. Pilla, M. Castro, F. Z. Boito, P. O. A. Navaux, and J.-F. Méhaut. Performance/energy trade-off in scientific computing: the case of arm big. little and intel sandy bridge. *IET Computers & Digital Techniques (CDT)*, 2014.

[47] L.-N. Pouchet. The Polyhedral Benchmark Suite. http://www.cse.ohio-state.edu/~pouchet/software/polybench/, 2012. [Online; accessed 05-Oct-2015].

[48] N. Rajovic, A. Rico, N. Puzovic, C. Adeniyi-Jones, and A. Ramirez. Tibidabo: Making the case for an arm-based hpc system. *Future Generation Computer Systems*, 2014.

[49] N. Rajovic, L. Vilanova, C. Villavieja, N. Puzovic, and A. Ramirez. The low power architecture approach towards exascale computing. *Journal of Computational Science*, 2013.

[50] C. I. Rodrigues, D. J. Hardy, J. E. Stone, K. Schulten, and W.-M. W. Hwu. Gpu acceleration of cutoff pair potentials for molecular modeling applications. In *Computing Frontiers (CF)*, 2008.

[51] G. Sanchez. Pls path modeling with r. *Online, January*, 2013.

[52] T. Shimokawabe, T. Aoki, C. Muroi, J. Ishida, K. Kawano, T. Endo, A. Nukada, N. Maruyama, and S. Matsuoka. An 80-fold speedup, 15.0 tflops full gpu acceleration of non-hydrostatic weather model asuca production code. In *High Performance Computing, Networking, Storage and Analysis (SC)*, 2010.

[53] G. Singh, G. Favir, and A. Young. AppledMicro X-Gene2. In *HotChips*, 2014.

[54] P. Stanley-Marbell and V. C. Cabezas. Performance, power, and thermal analysis of low-power processors for scale-out systems. In *Workshop on High-Performance, Power-Aware Computing (HPPAC)*, 2011.

[55] T. G. Team. GCC 4.9 Release Series: Changes, New Features, and Fixes. https://gcc.gnu.org/gcc-4.9/changes.html, 2015. [Online; accessed 05-Oct-2015].

[56] M. Tenenhaus, V. E. Vinzi, Y.-M. Chatelin, and C. Lauro. Pls path modeling. *Computational Statistics & Data Analysis (CSDA)*, 2005.

[57] A. Tiwari, K. Keipert, A. Jundt, J. Peraza, S. S. Leang, M. Laurenzano, M. S. Gordon, and L. Carrington. Performance and energy efficiency analysis of 64-bit arm using gamess. In *Proceedings of the 2nd International Workshop on Hardware-Software Co-Design for High Performance Computing*, page 8. ACM, 2015.

[58] A. Tiwari, M. A. Laurenzano, L. Carrington, and A. Snavely. Auto-tuning for energy usage in scientific applications. In *European Conference on Parallel Processing (Euro-par)*, 2012.

[59] W. Wang, J. Cavazos, and A. Porterfield. Energy auto-tuning using the polyhedral approach. In *Workshop on Polyhedral Compilation Techniques (IMPACT)*, 2014.

[60] S. White. The AMD Opteron A1100 Processor Codenamed "Seattle". In *HotChips*, 2014.