# Characterizing the Performance-Energy Tradeoff of Small ARM Cores in HPC Computation

Michael A. Laurenzano[†*], Ananta Tiwari[†§], Adam Jundt[†], Joshua Peraza[†], William A. Ward, Jr.[+], Roy Campbell[+], and Laura Carrington[†§]

[†]EP Analytics
[*]Dept. of Computer Science and Engineering, University of Michigan
[§]Performance Modeling and Characterization Lab., San Diego Supercomputer Center
[+]High Performance Computing Modernization Program, U.S. Dept. of Defense
{michaell,ananta.tiwari,adam.jundt,joshua.peraza,
laura.carrington}@epanalytics.com
{william.ward,roy.campbell}@hpc.mil

**Abstract.** Deploying large numbers of small, low-power cores has been gaining traction recently as a system design strategy in high performance computing (HPC). The ARM platform that dominates the embedded and mobile computing segments is now being considered as an alternative to high-end x86 processors that largely dominate HPC because peak performance per watt may be substantially improved using off-the-shelf commodity processors.

In this work we methodically characterize the performance and energy of HPC computations drawn from a number of problem domains on current ARM and x86 processors. Unsurprisingly, we find that the performance, energy and energy-delay product of applications running on these platforms varies significantly across problem types and inputs. Using static program analysis we further show that this variation can be explained largely in terms of the capabilities of two processor subsystems: single instruction multiple data (SIMD)/floating point and the cache/memory hierarchy; and that static analysis of this kind is sufficient to predict which platform is best for a particular application/input pair. In the context of these findings, we evaluate how some of the key architectural changes being made for upcoming 64-bit ARM platforms may impact HPC application performance.

## 1 Introduction

As large-scale high performance computing (HPC) systems have grown in size and the scope of problems being solved, reducing their power consumption has become a first-class problem. Indeed, many argue that power consumption is one of the primary constraints on the size of upcoming HPC systems [4][5][20][27][30]. We see this impacting industry, academia, and government, where substantial effort and resources are being marshaled to improve energy efficiency in HPC centers. On the other hand, the problems being solved on HPC systems, ranging from basic research to solving day-to-day problems in defense and industry, have HPC users demanding more and more performance out of their systems.

In response to these forces, HPC system architects have sought out designs that deliver higher performance with lower power budgets. One of the design alternatives that has gathered much attention along these lines is to use a large number of small, low-power cores in place of a smaller number of large, power-hungry cores. In particular, ARM processors, the dominant platform in the embedded and mobile computing domains, are being considered. The argument for using a large number of ARM cores is twofold. First, low-power cores are often more energy efficient than high-end cores [17]. Second, having come from domains which have always been power constrained, ARM designs in particular have been engineered to be frugal with power; careful attention having been given to include only those features that are worth the extra power they consume [7]. However, the question remains: *are those features well-suited to HPC applications?*

Current 32-bit ARM platforms such as ARMv7 have limitations that preclude their immediate use in modern HPC systems: only 4GB of memory are supported per process [15], and the ISA and hardware support for vector math is limited [8]. Ameliorating these limitations is one of the purposes of ARMv8, a 64-bit version of the ARM architecture, which is set to be released in early to mid 2014. Among other improvements, ARMv8 includes the ability to natively address significantly more than 4GB of memory, along with support for IEEE754 double-precision (DP) math and vectorized DP operations [14]. Still, it remains unclear whether these improvements will impact the ability of ARMv8 to deliver satisfactory performance to broad classes of HPC applications, and to what extent they will improve upon existing ARMv7 processors.

In this work, we characterize the performance and energy of ARM and x86 platforms by drawing compute kernels and applications from a number of HPC problem domains. These benchmarks are methodically characterized in terms of their performance and power on several ARMv7 (32-bit) and x86 processors. We examine performance, energy and energy-delay product (EDP), finding that these metrics vary by least an order-of-magnitude on a given implementation, and that they depend on the specific features of the application being run. We employ static program analysis on the benchmark kernels to characterize their behavior in terms of memory and floating point operations. From these characteristics, we develop simple regression models for performance, energy, and EDP disparities across applications, finding that these are largely explainable as functions of the memory and floating point characteristics of the compiled application. Building upon this insight, we present a model for estimating how performance is likely to change with improvements in the CPU and memory of upcoming 64-bit ARMv8 systems, finding that both have significant impacts on the performance of a broad class of applications.

The rest of this paper is structured as follows. Section 2 discusses work in the literature related to this paper. Section 3 explains the experimental methodology used in this work to assess the performance and power characteristics of HPC applications. Section 4 presents a methodical evaluation of two ARM platforms on a number of compute kernels and application benchmarks, followed by a discussion of the factors underlying the performance and energy characteristics of the applications and how these characteristics are likely to be impacted by the introduction of 64-bit ARM platforms. Finally, Section 5 concludes.

## 2 Related Work

This section describes the related literature in two areas that intersect with our work: using ARM cores in HPC and HPC Performance Modeling.

### 2.1 ARM in High Performance Computing

Rajovic et al. [26] evaluate the performance and energy efficiency of the Tegra 2, Tegra 3, and Quadro 1000M on a set of HPC microkernels. The Tegra 2 and 3 contain two and four core ARM Cortex-A9 processors respectively, and the Quadro 1000M is a discrete mobile

GPU. Padoin et al. [24] compare the scalability and energy efficiency of a PandaBoard, Snowball, and Tegra 2 when running High Performance Linpack. Ou et al. [23] compare energy and cost efficiency of a PandaBoard containing an ARM Cortex-A9 with an Intel Core2 Q9400 on three applications: web server throughput, an in-memory database, and video transcoding. They find that the PandaBoard achieved the greatest energy efficiency gains in less computationally intensive applications (the in-memory database in their study). Fürlinger et al. build a cluster of second-generation Apple TV devices which utilize an ARM Cortex-A8 [13]. They evaluate CPU and memory performance compared to a BeagleBoard and system performance per watt running High Performance Linpack compared to systems on the Green500 list.

Blem et al. [7] focus on the specific microarchitectural implementations of ARM and x86 processors, comparing an ARM Cortex-A8, ARM Cortex-A9, Intel Sandybridge, and an Intel Atom. By showing that the Atom could achieve similar energy consumption to the Cortex-A9 when controlling for microarchitectural features, they conclude that ISA is not major determinant of energy efficiency, instead finding that ARM and x86 implementations are simply different engineering design points.

Our work complements this existing body of literature. Our contribution is to document the performance and energy impact ARM cores have on a wide range of HPC computational benchmarks, as well as to show that the variability in performance and energy can largely be attributed to FP/SIMD computation and interactions with the memory subsystem.

### 2.2 HPC Application Performance Modeling

Kerbyson et al. propose some of the seminal ideas in predictive application performance and scalability modeling, showing that it is possible to accurately model the performance for a single application and that the model depends on specifics of the implementation of that application [18][19]. Several other works show how to use an application-independent approach to modeling performance, using a variety of application characteristics collected as traces of the running application, then mixing those with the results of measurement microkernels that are deployed on the system to predict performance for the application/system pair [10][28]. Snavely et al. [29] show that while a cycle accurate simulator could be very accurate, it was infeasible for a full-scale HPC application. Instead, they show that it is possible to tractably predict performance using a few important features.

Carrington et al. [9] show that simple combinations of metrics are infeasible to use for precisely predicting HPC application performance. In this work we show that even simple, static features of HPC applications can be employed to provide useful insights into the direction and magnitude of their performance and energy characteristics, even while precise performance prediction with those features may not be feasible.

## 3 Analysis and Measurement Methodology

The aim of this work is to characterize an extensive set of HPC application benchmarks in terms of their performance, energy and energy-delay product (EDP) on a several ARM processor configurations. This section discusses the methodological considerations made to develop these characterizations. We begin by discussing the performance measurement methodology, followed by a discussion of a methodology for attributing the wall-level power draw to the workload running on a system. Last, we describe a set of program analysis tools and methodologies that are deployed in the evaluation to develop energy models.

### 3.1 Performance Measurement

This work evaluates a number of HPC application kernels and benchmarks for performance and power. Our approach to measuring performance on application kernels is to manually insert timing instrumentation around the key computational loops, avoiding measurement of initialization and finalization code such as parsing arguments, reading files, allocating/freeing memory and output validation. The performance of these activities is important, yet in benchmark kernels they tend to be greatly over represented as a fraction of runtime relative to their runtime in full application codes. Many HPC benchmarking packages such as the NAS Parallel Benchmarks [6], pcubed [21] and polybench [25] adopt a similar rationale, providing (sometimes multiple) timers around important phases of computational work.

### 3.2 Attributing Power to a Workload

The goal of our power measurement methodology is to isolate the power draw consumed only by the CPUs running the application. To isolate the power draw in this fashion we measure system-wide power draw during long-running computational kernels at several core counts, with the purpose of deriving the power contribution only of the cores actively involved in the computation. We begin with the formulation of system-wide power shown in Equation 1.

$$W^i_{system} = i * W_{active} + (N - i) * W_{idle} + W_{other} \tag{1}$$

The elements of Equation 1 are $i$, the number of active cores, $W^i_{system}$, the measured power using $i$ active cores and $N$, the total number of available cores on the system. The goal of producing an equation in this form is to derive $W_{active}$, the power draw of a single active core, $W_{idle}$, the power draw of a single idle core and $W_{other}$, the power draw of all other system components. Because there are three unknowns ($W_{active}$, $W_{idle}$ and $W_{other}$), measurements at three core counts ($i = c_1, c_2, c_3$) is sufficient to produce system of equations, shown in Equation 2, to which we can apply any of a number of numerical techniques to approximate the unknowns. In this work we use Gaussian elimination.

$$\begin{bmatrix} W^{c_1}_{system} \\ W^{c_2}_{system} \\ W^{c_3}_{system} \end{bmatrix} = W_{active} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} + W_{idle} \begin{bmatrix} N - c_1 \\ N - c_2 \\ N - c_3 \end{bmatrix} + W_{other} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \tag{2}$$

This framing of the problem makes several assumptions. First, it assumes that $W_{active}$, $W_{idle}$ and $W_{other}$ do not depend on the number of cores that are active. For this assumption to hold, the workload must be carefully selected so that each additional running instance of the kernel produces a similar additional power draw increase. This means ensuring that running instances do not compete with one another for processor resources like cache and interconnect, which would introduce execution stalls and reduce circuit-level switching activity. Second, this formulation resolves $W_{active}$, $W_{idle}$ and $W_{other}$ only for a particular benchmark. Empirically, however, we found that $W_{idle}$ and $W_{other}$ for a particular system are stable across a range of computational kernels, indicating that these values are relatively independent of the workload running on the system. Therefore, we utilize this methodology for only a few kernels on each system to estimate $W_{idle}$ and $W_{other}$ for the system, allowing us to isolate the power per active core for **any workload** by plugging the full system power measurement for that workload $W^i_{system}$, along with $W_{idle}$ and $W_{other}$, into Equation 1.

## 3.3 Program Static Analysis Tools

In this work we employ two binary analysis tools to analyze application codes. In particular, we use the EPAX toolkit [12] to analyze the static properties of ARM binaries and the PEBIL toolkit [22] on x86 binaries. Static binary analysis is the act of examining a compiled binary program to extract information about the properties of the code and data that reside within that program. EPAX and PEBIL accomplish this by reading the executable from disk, parsing and disassembling its contents, then writing out a file containing a number of details about the machine-level instructions in the program as well the relationship between those instructions such as their membership in high-level structures such as basic blocks, loops, and functions. In this work, we use EPAX on ARM binaries and PEBIL on x86 binaries to extract a number of features we expect to be salient to HPC applications, including counts of floating point and vector (SIMD) operations, along with the counts and properties of memory operations. When possible to gather at compile-time, we augment the information gathered by EPAX and PEBIL with information about the sizes of key data structures within the important computational loops. As we show in Section 4.3, this array of static properties is enough to make informative predictions about the direction and magnitude of the relative amount of energy consumed when running the application on ARM and x86 systems.

| | **Intel Sandy Bridge** | **ARM Cortex-A9** | **ARM Cortex-A15** |
|---|---|---|---|
| **Name** | Dell Poweredge T620 | Dell Copper | nCore BrownDwarf Y-class |
| **Platform** | x86_64 64-bit | ARMv7 32-bit | ARMv7 32-bit |
| **Processor** | 8-core 2.6GHz Xeon E5-2670 | 4-core 1.6GHz Marvell MV78460 | 4-core 1.4GHz TI 66AK2E05 |
| **D-Cache** | Shared 20MB L3, Priv. 256KB L2, Priv. 32KB L1 | Shared 2MB L2, Priv. 32KB L1 | Shared 4MB L2, Priv. 32KB L1 |
| **Memory** | 32GB 1333MHz DDR3 | 4GB 1333MHz DDR3 | 2GB 1600MHz DDR3 |
| **FP/SIMD** | SSE, AVX | VFPv3-D32, no SIMD | VFPv4, NEON |
| **Notes** | *Turbo and HT disabled* | - | *c66x DSP cores disabled* |

**Table 1.** Platform configurations

| **Type** | **Programs** | | **Summary** |
|---|---|---|---|
| **Compute Kernels** | **PolyBench[25]** | adi, atax, bicg, cholesky, doitgen, dynprog, fdtd-2d, fdtd-ampl, gemver, gesummv, grammschmidt, jacobi-2d, mvt, seidel, symm, trisolv, trmm | linear algebra, data mining, stencils |
| | **Other** | covcol, dct, dsyr2k, dsyrk, matmulinit, mm, stencil-3d, strmm, strsm, swim, tce | |
| **Application Benchmarks** | **Mantevo[16]** | miniMD, CoMD, miniGhost | molecular dynamics, finite element, finite difference, quantum chromodynamics, plasma physics |
| | **CORAL[1]** | AMGmk, MILCmk | |
| | **Trinity[11]** | miniFE, GTC | |

**Table 2.** Benchmarks and applications

## 4 Evaluation

### 4.1 Experimental Setup

We utilize three distinct platforms throughout this evaluation, summarized in Table 1. These test platforms consist of a high-end Intel Sandy Bridge E5-2670, a popular configuration among the largest modern supercomputers [2]. We also use two energy-efficient ARM server platforms: a Cortex-A9 based Dell Copper server and a Cortex-A15 based nCore BrownDwarf Y-class supercomputer. For power measurement, we use a Yokogawa WT310 digital power meter [3] to measure AC power draw of the entire system at the wall. Power measurements for each benchmark run are then isolated using the approach described in Section 3.2.

On our test platforms we deploy 28 compute kernels and 7 application benchmarks, summarized in Table 2. Many of the compute kernels are drawn directly from the Polyhedral Benchmark Suite [25], while others are augmented versions thereof or hand-written compute kernels of our devising. For each compute kernel we generate a total of eight configurations, consisting of the cross product of double- and single-precision (DP and SP) versions of the benchmarks and data set sizes that are large enough that they fit into each of the four levels of the memory hierarchy on all systems (L1 , L2 and L3 Cache[1] as well as main memory). This yields a total of 224 compute kernels. The sizes of the four data sets were chosen carefully so that both the DP and SP versions fit into the same level of the memory hierarchy on all systems (SP data types generally consume half the memory of their DP counterpart). For our particular test platforms, we use 10-15KB of SP data for L1, 80-100KB of SP data for L2, 700-900KB of SP data for L3 and 50-70MB of SP data for main memory. The seven application benchmarks are also described in Table 2, which are drawn from the Mantevo [16], CORAL [1] and NERSC-8 Trinity [11] benchmark suites and represent applications from among a number of unique computational domains. For most applications we use both DP and SP versions. The exception to this is miniMD, for which we were unable to compile the DP version on either of the ARM platforms. Benchmarks and applications are compiled with `gcc`, using optimization level `-O3` and vectorization support flags: `-funsafe-math-optimizations -mavx` on the Sandy Bridge and `-funsafe-math-optimizations -mfpu=neon`[2] on both ARM systems. We pin threads to cores to ensure that no thread migration occurs during any experimental runs. All performance, power, energy and EDP numbers presented are the average of three runs.

### 4.2 Performance and Energy Characterization

We begin the evaluation by presenting performance and energy characterizations of the compute kernels and benchmark applications on all systems. Figure 1 shows distributions of the performance 1(a)-1(b), energy 1(c)-1(d) and EDP 1(e)-1(f) for the compute kernels, grouped according to floating point precision (SP/DP) and which memory level the kernel exercises (L1/L2/L3/MM) and normalized to the Intel Sandy Bridge system, where values greater than one for runtime indicate ARM performance suffers relative to the Sandy Bridge system, and values less than one for energy and EDP identify benchmarks that are more energy efficient when executed on the ARM systems. Three

[1] Neither the Cortex-A9 nor the Cortex-A15 have L3 cache, and thus they have two sizes that fit into main memory.

[2] Without `-funsafe-math-optimizations`, SIMD NEON instructions will fail to materialize on the ARM systems because those instructions do not adhere to the IEEE754 standard.

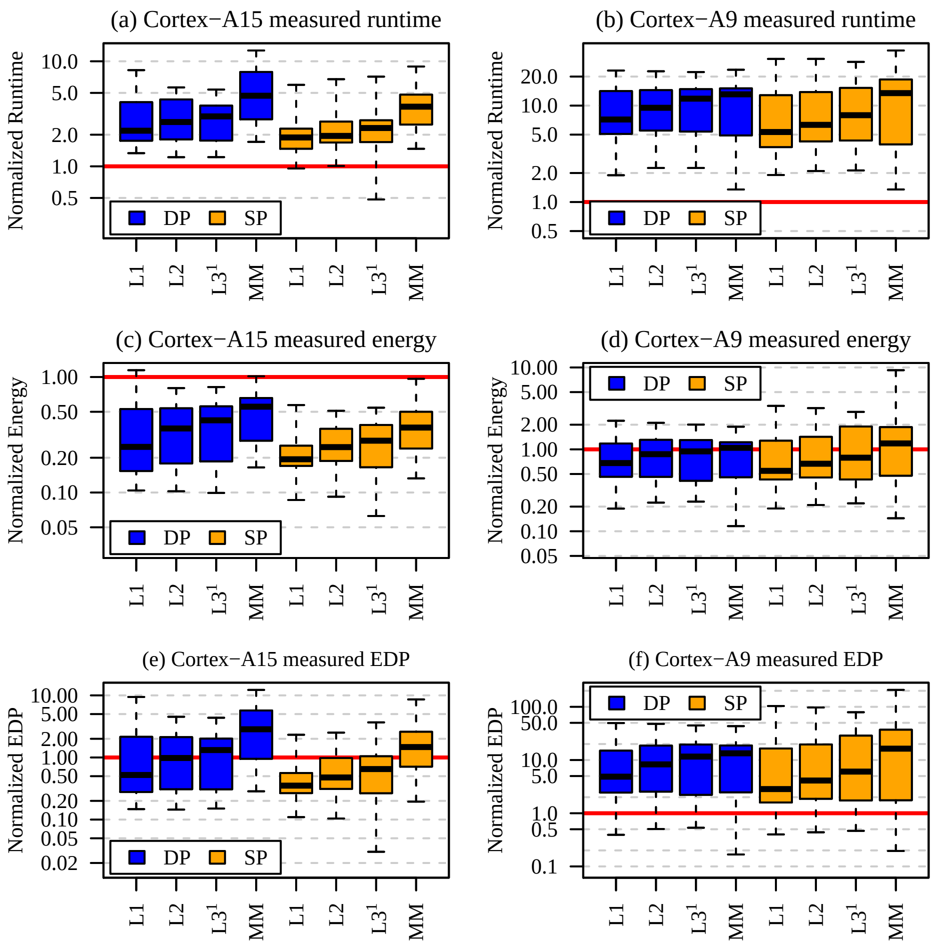

**Fig. 1.** Distributions of the runtime (a)-(b), energy (c)-(d) and energy-delay product (e)-(f) for single-core compute kernels on ARM Cortex-A15 and Cortex-A9, relative to Intel Sandy Bridge. Distributions are shown as box plots, which highlight the the maximum (upper tail), 75th percentile (box upper-bound), median (line within box), 25th percentile (box lower-bound) and minimum (lower tail). Interested readers can find more detailed charts at `http://epanalytics.com/data/euro-par2014/`.

interesting trends can be observed. First, in almost all cases the SP versions of the kernels show better characteristics on the ARM systems over their DP counterparts, an issue that should be resolved on future 64-bit ARM systems. Second, there is substantial variation in runtime even within a particular grouping of kernels, suggesting that performance, energy and EDP have a substantial software component, rather than being a simple property of the hardware. Third, the larger the working set, the worse the efficiency is on the ARM systems. For example, the Cortex-A15 energy results show that median L1-Cache energy improvement is more than double that of main memory energy improvement. This suggests that there is room to improve the efficiency of HPC applications by improving the cache and memory architecture of the ARM platforms. We refer the interested reader to `http://epanalytics.com/data/euro-par2014/` to find a more detailed treatment of these charts.

In Figure 2, we present similar findings on the performance 2(a), the energy 2(b) and the energy-delay product 2(c) for the application benchmarks.

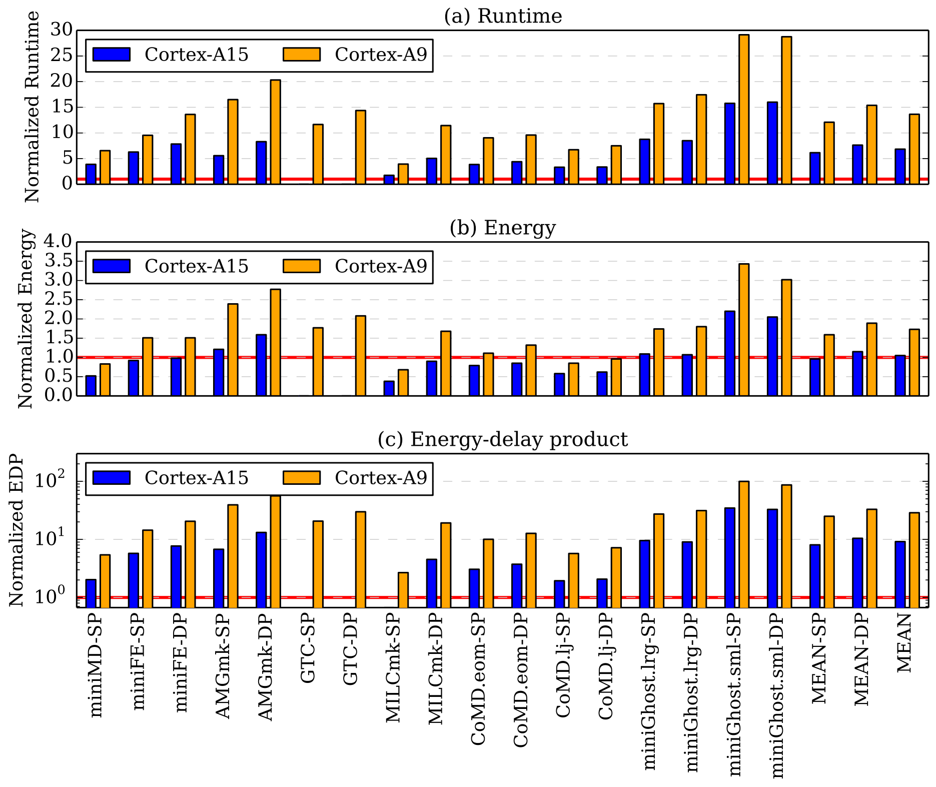

**Fig. 2.** Runtime (a), energy (b) and energy-delay product (c) for quad-core application benchmarks on an ARM Cortex-A15 and Cortex-A9, relative to an Intel Sandy Bridge. Note that (c) is plotted on a log scale.

### 4.3 Attributing Energy Characteristics to Static Program Features

In Section 3.3, we described two static binary analysis tools, PEBIL for x86 and EPAX for ARM, which were employed to collect information about the the memory/cache

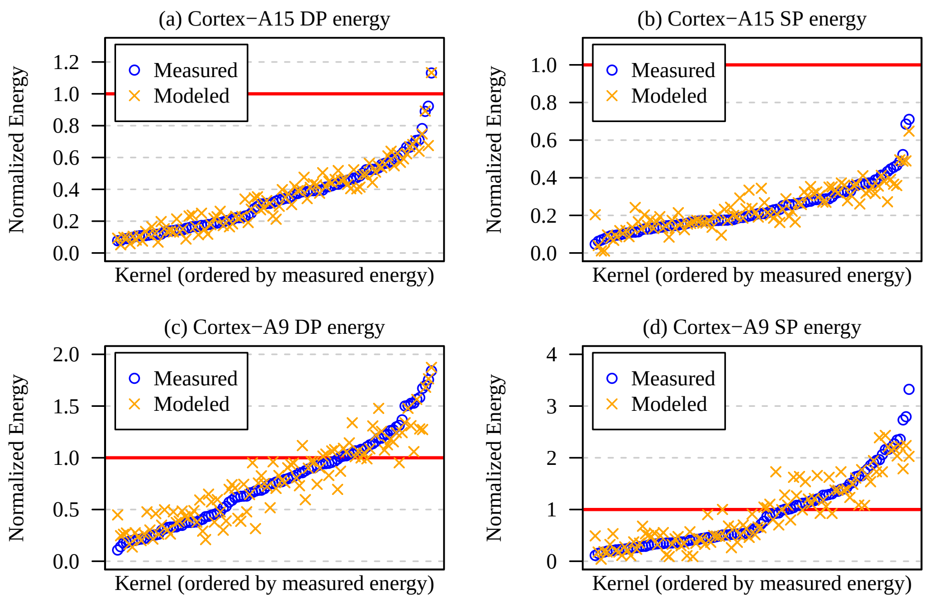

**Fig. 3.** Measured and modeled energy for Cortex-A15 (a)-(b) and Cortex-A9 (c)-(d). A statistical measure of the variation in kernel energy that is explained by the models (adjusted R-squared) is (a) 90%, (b) 64%, (c) 80% and (d) 76%.

and floating point/SIMD operations that reside within the key loops of the compute kernels. Specifically, we collect the counts of instructions, memory operations, floating point operations, the number of bytes moved per memory operation, and the size of the key data structure(s) in the loop. We then use multivariate linear regression to build models of the energy consumption (normalized to Sandy Bridge) of the compute kernels as a function only of these terms and some of their simple variants (e.g., floating point ops per instruction), along with 10-fold cross validation on the models. Figure 3 shows the measured and modeled energy consumption for the Cortex-A15 3(a)-3(b) and the Cortex-A9 3(c)-3(d), again normalized to the Intel Sandy Bridge.

Two interesting features are apparent from Figure 3. First, we observe that the models capture a significant fraction of the variation in energy across the compute kernels. Visually, this can be seen where the shape of the modeled energy points follows the shape of the measured energy points. A statistical measure of this property is given by the *adjusted R-squared* of the model [31]. Adjusted R-squared is the percentage of variation captured by the model, where a perfect model would capture 100%. The models shown in Figures 3(a), 3(b), 3(c) and 3(d) have adjusted R-squared measures of 90%, 64%, 80% and 76% respectively. Qualitatively, the models account for the majority of the energy variation across benchmarks. Second, the models are able to correctly predict which system uses the least energy to run a particular compute kernel in 210 of the 224 kernels. We take care to note that these models are imprecise, lacking exactness in the energy prediction of any particular compute kernel. Nevertheless, they are surprisingly useful for estimating the direction and magnitude of the energy difference between the ARM and x86 systems.

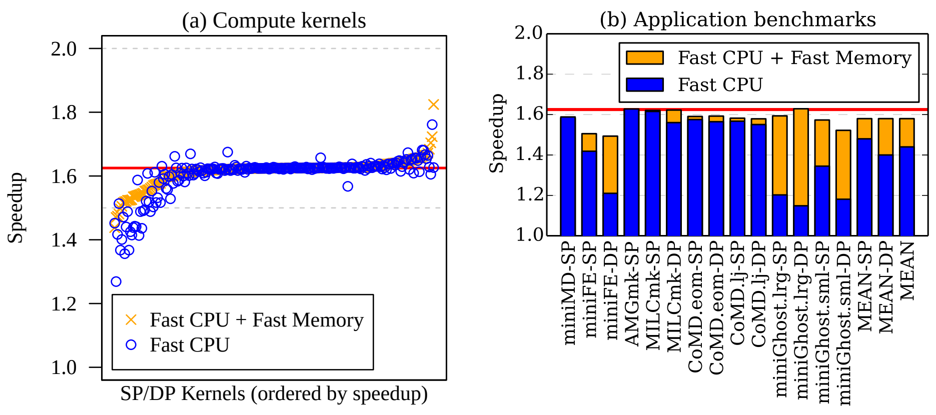

**Fig. 4.** Estimated speedup conferred by CPU and memory speed improvements in 64-bit ARM systems for (a) compute kernels (b) and application benchmarks. The thick red line shows the theoretical speedup that would be achieved if scaling by the CPU clock rate increase ($2.6/1.6 = 1.625$).

### 4.4 Implications for 64-bit ARM

Implementations of 64-bit ARM platforms are expected to arrive in early to mid 2014. It is widely anticipated that 64-bit ARM will improve upon the current 32-bit implementations by offering higher clock rates, improvements in the memory architecture, and more complete vector math support, for example by supporting 2-wide DP SIMD operations and fully adhering to the IEEE754 standard. We estimate the impact of these factors on performance by examining the relationship those factors have to performance on the Sandy Bridge system. In particular, we dial down the memory and processor clock frequencies on the Sandy Bridge system to 800MHz and 1.6GHz respectively to measure the speedup between the low and high clock rate runs, which represents how much benefit is conferred to the application by running on hardware which has faster compute and memory resources. Similarly, we estimate the impact of faster CPU only by dialing down only the memory. The estimated speedups produced by this approach are presented in Figure 4, showing in 4(a) that increasing a slow clock rate by a factor of 1.625 confers a speedup of at least 1.625x for a majority (81%) of compute kernels. This suggests that clock rate increases in 64-bit ARM systems are likely to show substantial improvements for the performance of many HPC applications. In 4(b), we present the application benchmarks speedups when speeding up only the CPU clock rate (blue/dark), and both the CPU and memory clock rates (orange/light). From these results and the results in 4(a), we can infer that increases in the speed of the cores, as opposed to the memory, account for the largest share of the speedups in the applications. We conclude from these insights that improvements in the clock rates of 64-bit ARM implementations are likely to have a substantial benefit to HPC applications, while memory speed plays a significant but quantitatively less important role.

## 5 Conclusion

Using a large number of small, low-power cores has been gaining ground as a design strategy to improve the energy efficiency of upcoming HPC systems. As ARM is the dominant platform in the mobile and embedded computing segments, many believe that

ARM is a viable competitor to the high-end x86 systems that make up a substantial fraction of large-scale HPC systems today. In this work, we methodically documented the performance and energy characteristics of a number of HPC computations on several current ARM platforms. We found that performance and energy efficiency of the ARM systems varies by up to an order-of-magnitude and depends on the computational and memory characteristics of the application. Moreover, we showed that this variability can be described as a function of two important processor subsystems: the floating point/SIMD unit and the cache/memory hierarchy. Finally, we investigated the performance implications that 64-bit ARM systems will have, finding that HPC applications stand to benefit substantially from changes in the CPU and memory subsystems.

## Acknowledgments

This work was supported in part by the U.S. Department of Defense HPCMP PETTT program (Contract No: GS04T09DBC0017 though DRC) and by the U.S. Air Force Office of Scientific Research under AFOSR Award No. FA9550-12-1-0476. We also wish to thank Mr. Tim Carroll and Dr. Mark Fernandez of Dell for providing early access to the Cortex-A9 based Dell Copper ARM server.

## References

1. *CORAL Benchmark Codes*, 2013. `https://asc.llnl.gov/CORAL-benchmarks/`.
2. *The Top 500 list*, November 2013. `http://www.top500.org`.
3. *Yokogawa: WT300 Series Digital Power Meters*, 2014. http://tmi.yokogawa.com/us/products/digital-power-analyzers/digital-power-analyzers/wt300-series-digital-power-meters/.
4. K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, et al. The landscape of parallel computing research: A view from berkeley. Technical report, Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, 2006.
5. N. Attig, P. Gibbon, and T. Lippert. Trends in supercomputing: The european path to exascale. *Computer Physics Communications*, 182(9):2041–2046, 2011.
6. 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 summary and preliminary results. In *Supercomputing, 1991. Supercomputing'91. Proceedings of the 1991 ACM/IEEE Conference on*, pages 158–165. IEEE, 1991.
7. E. R. Blem, J. Menon, and K. Sankaralingam. Power struggles: Revisiting the risc vs. cisc debate on contemporary arm and x86 architectures. In *HPCA*, pages 1–12, 2013.
8. A. Buttari, J. Dongarra, J. Langou, J. Langou, P. Luszczek, and J. Kurzak. Mixed precision iterative refinement techniques for the solution of dense linear systems. *International Journal of High Performance Computing Applications*, 21(4):457–466, 2007.
9. L. Carrington, M. Laurenzano, A. Snavely, R. L. Campbell, and L. P. Davis. How well can simple metrics represent the performance of hpc applications? In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, SC '05, pages 48–, Washington, DC, USA, 2005. IEEE Computer Society.
10. L. Carrington, A. Snavely, X. Gao, and N. Wolter. A performance prediction framework for scientific applications. In *ICCS Workshop on Performance Modeling and Analysis (PMA03*, pages 926–935, 2003.
11. 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. 2013.
12. EP Analytics. *EPAX Toolkit: Binary Analysis for ARM*, 2014. `http://epaxtoolkit.com/`.

13. K. Fürlinger, C. Klausecker, and D. Kranzlmüller. Towards energy efficient parallel computing on consumer electronic devices. In *Information and Communication on Technology for the Fight against Global Warming*, pages 1–9. Springer, 2011.
14. J. Goodacre. Technology preview: The armv8 architecture. *White Paper, Nov*, 2011.
15. J. Goodacre and A. Cambridge. The evolution of the arm architecture towards big data and the data-centre. In *Proceedings of the 8th Workshop on Virtualization in High-Performance Cloud Computing*, page 4. ACM, 2013.
16. 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, Tech. Rep*, 2009.
17. U. Hölzle. Brawny cores still beat wimpy cores, most of the time. *IEEE Micro*, 30(4), 2010.
18. D. J. Kerbyson, H. J. Alme, A. Hoisie, F. Petrini, H. J. Wasserman, and M. Gittings. Predictive performance and scalability modeling of a large-scale application. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '01, pages 37–37, New York, NY, USA, 2001. ACM.
19. D. J. Kerbyson and P. W. Jones. A performance model of the parallel ocean program. *Int. J. High Perform. Comput. Appl.*, 19(3):261–276, Aug. 2005.
20. P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, et al. Exascale computing study: Technology challenges in achieving exascale systems. 2008.
21. M. A. Laurenzano, M. Meswani, L. Carrington, A. Snavely, M. M. Tikir, and S. Poole. Reducing energy usage with memory and computation-aware dynamic frequency scaling. In *Euro-Par 2011 Parallel Processing*, pages 79–90. Springer, 2011.
22. M. A. Laurenzano, M. M. Tikir, L. Carrington, and A. Snavely. Pebil: Efficient static binary instrumentation for linux. In *Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium on*, pages 175–183. IEEE, 2010.
23. Z. Ou, B. Pang, Y. Deng, J. K. Nurminen, A. Yla-Jaaski, and P. Hui. Energy- and cost-efficiency analysis of arm-based clusters. *Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2012.
24. E. L. Padoin, D. A. de Oliveira, P. Velho, P. O. Navaux, B. Videau, A. Degomme, and J.-F. Mehaut. Scalability and energy efficiency of hpc cluster with arm mpsoc.
25. L.-N. Pouchet. *PolyBench: The Polyhedral Benchmark suite*, 2012. `http://www.cse.ohio-state.edu/~pouchet/software/polybench/`.
26. N. Rajovic, A. Rico, J. Vipond, I. Gelado, N. Puzovik, and A. Ramirez. Experiences with mobile processors for energy efficient hpc. *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2013.
27. J. Shalf, S. Dosanjh, and J. Morrison. Exascale computing technology challenges. In *High Performance Computing for Computational Science*, pages 1–25. Springer, 2011.
28. S. Sharkawi, D. DeSota, R. Panda, S. Stevens, V. Taylor, and X. Wu. Swapp: A framework for performance projections of hpc applications using benchmarks. In *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, IPDPSW '12, pages 1722–1731, Washington, DC, USA, 2012. IEEE Computer Society.
29. A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha. A framework for performance modeling and prediction. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, Supercomputing '02, pages 1–17, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.
30. M. Snir, W. Gropp, and P. Kogge. Exascale research: Preparing for the post–moore era. 2011.
31. W. P. Vogt and R. B. Johnson. *Dictionary of statistics & methodology: A nontechnical guide for the social sciences.* Sage, 2011.