Overcoming the Power Wall by Exploiting Inexactness and Emerging COTS Architectural Features

Trading Precision for Improving Application Quality

Mike Fagan[†], Jeremy Schlachter[‡], Kazutomo Yoshii[*],
Sven Leyffer[*], Krishna Palem[†], Marc Snir[*], Stefan M. Wild[*], and Christian Enz[‡]

[*]Mathematics and Computer Science Division
Argonne National Laboratory, Lemont, IL 60439, USA

[†]Department of Computer Science
Rice University, Houston, TX, USA

[‡]Integrated Circuits Laboratory (ICLAB)
Ecole Polytechnique Fédérale de Lausanne (EPFL), Neuchâtel, Switzerland

***Abstract*—Energy and power consumption are major limitations to continued scaling of computing systems. Inexactness where the quality of the solution can be traded for energy savings has been proposed as a counterintuitive approach to overcoming those limitation. However, in the past, inexactness has been necessitated the need for highly customized or specialized hardware. In order to move away from customization, in earlier work [1], it was shown that by interpreting precision in the computation to be the parameter to trade to achieve inexactness, weather prediction and page rank could both benefit in terms of yielding energy savings through reduced precision, while *preserving* the quality of the application. However, this required representations of numbers that were not readily available on *commercial off-the-shelf* (COTS) processors. In this paper, we provide opportunities for extending the notion of trading precision for energy savings into the world COTS. We provide a model and analyze the opportunities and behavior of all three IEEE compliant precision values available on COTS processors: (i) double (ii) single, and (iii) half. Through measurements, we show through a limit study energy savings in going from double precision to half precision are a factor of 3.98.**

I. Introduction

It is widely believed that *energy* and *power* consumption are major limitations to continued scaling of computing systems. Often referred to as the "power wall" (or "energy wall"), this limitation now ranges from the obvious battery-constrained context of embedded computing, to general-purpose computing settings, namely supercomputers and data-centers.

Following the presidential executive order on creating a national strategic computing initiative [2], US agencies are engaged in a project aimed at leading to the deployment of an exascale computer by 2023. Power consumption is a major obstacle to the timely deployment of an exascale platform. The current top supercomputer, Sunway, consumes 15.3MW and achieves 93 petaflop/s on Linpack. With no technology improvements, an increase in performance would require a proportional increase in power consumption: An exascale system would consume more than 160MW. New technology improvements will occur, but it seems unlikely that an exascale system will achieve the target of 20MW.

There is a lot of interest in ameliorating these hurdles through customization—notably through highly stylized and dedicated architectures. These efforts at customization are ubiquitous in the embedded computing space where energy, speed, and size/weight have always played a critical role. The solution in this traditional context is to consider *system-on-a-chip* (SOC) architectures that meld varying amounts of general-purpose embedded computing with custom processing hardware and communication hardware. The problem with SOC architectures is that customization comes with a significant "one-time" *nonrecurring engineering cost* (NRE and *time-to-market* delays [3]. On the other hand, with general-purpose *commercial, off-the-shelf* (COTS) microprocessors, a single highly optimized design is used to amortize the NRE and time-to-market overheads through volume sales.

In addition, customized hardware requires customized software: A specialized system is generally harder to program than a general purpose microprocessor and does not take as much advantage of the large software ecosystem that is available for general purpose processors. For example, Sunway uses a specialized SOC with no caches, but only a small 64K scratchpad for each core. This requires significant changes in software.

This material is based upon work supported by: *i.*at Argonne National Laboratory: the US Dept. of Energy, Office of Science, ASCR, under contract DE-AC02-06CH11357; *ii.*at EPFL: the Swiss National Science Foundation project IneSoC under grant N^{o} 200021-144418; *iii.*at Rice University: DARPA Grant FA8750-16-2-0004.

A general technique for reducing energy/power consumption goes by the name of *inexact design*. The philosophy of inexact design espouses the idea of actively trading application accuracy for disproportionately large energy gains; see [4] for an overview. Recently, this approach was considered in the context of two ubiquitous applications drawn from the weather and climate modeling domains through the IGCM model, and in the context of "big data" through the *PageRank* algorithm [1]. This work established the following thesis:

> For applications that occur quite naturally, e.g. IGCM, PageRank), trading the precision at which the algorithms are implemented garners energy savings without compromising the quality of the application.

The thesis specializes the inexact design methodology by interpreting inexactness as lowering the precision of the computation. This application of inexact design opens the door to an entirely new approach for coping with the power- or energy-wall facing computing.

The work from [1], however, needed architectures where the floating-point numbers did not conform to IEEE standards. Therefore, while being a harbinger of how one could leverage COTS platforms to overcome energy hurdles, the methods could not be supported on currently-available microprocessors. Nevertheless, the insight from [1] is potentially useful if we apply the same methodology in the context of commercially available precision variants. This goal paves the way to the following questions addressed in this paper.

1) What are the most frequently occurring precision modes in COTS microprocessors?
2) What are the costs of various (e.g., integer, floating-point, load, store) instruction types as we vary precision?
3) What are realistic limits to *measured* energy gains we can hope to achieve across these classes?
4) What is a good model through which these measured gains can be explained and the architectural contributions from elements such as data movement from main memory, various levels of cache, and the data-path inferred?

A. Overview of the rest of the paper

The IEEE standards have provided single- and double-precision floating-point for some time and, more recently, a half-precision mode supporting 5 exponent bits, 10 mantissa bits, and a sign bit. We consider all three of these modes here. To start with, in section II, we outline a methodology through which we can reliably measure energy consumption based on novel hardware counters. Section III describes the three modes of arithmetic under consideration. In section IV, we construct (by hand) a family of three *microbenchmarks* to exercise each of the three modes in turn and measure the energy consumed. We show that by lowering precision, we obtain gains that are achievable with current commercially available microprocessors.

We based this study on an Intel core i7 4770 (3.40 GHz) processor. We are also interested, however, in being able to project the benefits of reduced precision in the context of other architectures as well. To do this, in section V, following [1], we consider energy to be relative or normalized by the cheapest operation and use the dimensionless quantity *virtual joules*. We use this model to reason about the gains reported in section IV. The model also serves as a tool to estimate energy savings on platforms that do not support direct physical energy measurement.

B. Related work

Early work on trading the quality or accuracy of a computation for energy savings based on physical mechanisms including CMOS devices can be found in [5]–[7]. A historical perspective on this approach to energy savings with a partial survey of the field can be found in [4]. In building the model used in this paper, we use the results from [8] both to validate our work as well as in determining costs of individual instruction classes. The concept of trading precision for cycles is well known in the supercomputing community [9, 10]. Furthermore, automating the precision/cycles tradeoff is being actively researched — see [11] for example. In contrast, we apply the concept of trading precision The reference list associated with each of the papers we cite provides a more complete citation record.

II. Our Experimental Methodology

Being interested in energy effects means that we must have some way of measuring the energy consumption of programs. For the COTS used in this work, we employed a DELL precision T1700 workstation operated by CentOS 7 and equipped with an Intel core i7 4770 (3.40 GHz) and 16 GB of DDR3 RAM. The operating system was Linux, kernel 4.3. All CPU cores were set to the minimal P-State except for one. The distinguished core was set to Turbo Boost.

This Intel architecture supports Running Average Power Limit (RAPL) hardware counters, which work much like any other hardware performance counter:

1) start counter
2) do work
3) stop counter

The difference between start and stop values measures the energy. The pseudocode for a RAPL measurement is shown in Algorithm 1. We employed RAPL to measure the energy per instruction (EPI) of several instruction classes. Our measurements for these instruction classes appear in section V. In most of the cases, RAPL measurements are consistent with physical measurements, but this methodology tends to underestimate the cost of main memory access [10].

III. Precision Modes available on COTS processors

Almost all floating-point units (FPUs) in commercially available processors support both IEEE 754 single-precision and double-precision arithmetic natively. To increase the peak performance, modern FPUs support single instruction, multiple data (SIMD) and fused multiply-add (FMA).

While single- and double-precision arithmetic are standard, there are many different half-precision formats available. There is, surprisingly, a standard half-precision format supported by

Algorithm 1 Generic RAPL energy measurement

```
Allocate and fill memory with random values
Initialize RAPL
for i = 1, i < N, i++ do
    Increment memory pointer by more than
    a cache line size (if need to nullify cache)
    General Work                ▷ Could be inline assembly
end for
Read RAPL
```

IEEE-754-2008. This standard is named `binary16`. It has 5 bits of exponent, 10 bits of mantissa (11 if you count the leading 1-bit), and a sign bit. The IEEE standard, however, treats `binary16` as a storage format *only*. General-purpose processors have, until recently, not supported `binary16`.

Intel recently introduced some nominal support for `binary16` half-precision into the AVX2 instruction subset in the 3rd generation Intel Core processor family [12]. There is, unfortunately, no native support for half-precision arithmetic in AVX2. Rather, Intel introduced instructions to convert half-precision data into single-precision data and convert single-precision data back into half-precision data. Any arithmetic to be done with `binary16` data must first be converted to standard single-precision. At the end of the calculations, the (single-precision) results may be converted back to half precision for storage.

Although actual floating-point arithmetic for half-precision data is carried out by single-precision logic, reducing data traffic size between processors and main memory benefits applications since memory-wall-induced underutilization is still one of the major problems in scientific computing [13]. Even worse, Moore's law has raised the memory wall. In the exascale computing era, data movement is expected to be one of the most dominant factors for performance and energy efficiency [14].

IV. Limit Study

This section gives insight into how precision can be traded for a significant amount of energy saving on COTS (AVX2) hardware. To see the effects, we constructed a microbenchmark consisting of a typical instruction sequence to compute a vectorized fused multiply-add. The generic code, that applies to single or double precision, is shown in algorithm 2. The only execution differences between single and double precision variants of algorithm 2 lie in the value of `NELTSV256`. A double precision number is 64 bits. Therefore, a 256-bit vector holds 4 double precision values. As a result, the value of `NELTSV256` is 4 for the double precision variant. On the other hand, a single precision number is 32 bits. That means a 256-bit vector holds 8 single precision values, making `NELTSV256` 8. The overall effect is that the double precision variant of the loop in algorithm 2 takes twice as many iterations as the single precision variant.

To study the effects of half precision, we could not use algorithm 2 directly — there is no native support for half precision arithmetic. Any arithmetic must be done in single precision. The Intel programming guide advocated treating half precision processing with 128-bit loads/stores [15]. Following the Intel-suggested method, we constructed algorithm 3. Note that the loop stride in algorithm 3 is the same as the loop stride in the single precision variant of algorithm 2. This means that the loop in algorithm 3 has the same number of iterations as the single precision variant of algorithm 2.

Algorithm 2 Generic Benchmark

```
for i = 1, i < N, i+= NELTSV256 do
    Load 3 256-bits vectors
    Fused Multiply and Add
    Store
end for
```

Algorithm 3 Half precision benchmark

```
for i = 1, i < N, i+= 8 do
    Load 3 128-bits vectors
    Convert them to single precision 256-bits vectors
    Fused Multiply and Add
    Convert result to half precision 128-bits vector
    Store
end for
```

The single and double precision variants of algorithm 2 together with algorithm 3 form the 3 building blocks for our limit study.

The independent variable in our limit study was the total number of data elements in the array — the variable `N` in the sample code. For our limit study, we varied `N` from 32 to 2^{28}. Our limit study measured and observed 2 dependent variables:

1) The energy per operation(EPI), measured in nJ/op
2) The performance (speed) measured in cycles

The results of our limit study can be seen in 2-part Figure 2. The horizontal axis is the data array size (independent variable). The vertical axis reflects each of the dependent variables. For each of double, single and half precision, Figure 2a shows the EPI, as it varies with data array size. The performance (measured in cycles) of the benchmark, as it varies with data array size is shown in Figure 2b. Although cycles are important, the focus of this paper is on the EPI dependent variable. Consequently, the remainder of this discussion addresses the EPI results. As can be seen in Figure 2a, the EPI approaches a *limiting*, or *asymptotic* value. The asymptotic region begins somewhere around data array size 10^7, extending to 2^{28}.

To compare results among the 3 precision variants, we selected a data array size of 10^8 as the sample independent variable value in the asymptotic region. Table I shows the EPI for each of the precisions for the selected 10^8 value. As the table shows, using single precision instead of double reduces the energy by a factor of 2.67. Using half precision instead of single precision reduces the energy by a factor of 1.49. Figure 1 graphically shows the normalized energy costs of

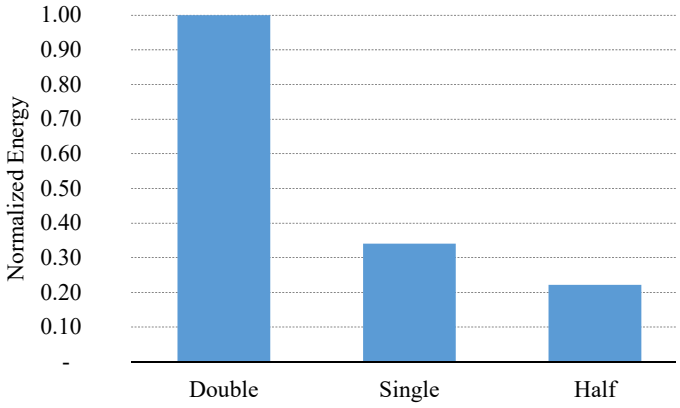


Fig. 1: Normalized energy consumption of the microbenchmarks estimated by the power model for a single-process execution with an array of 67 million elements

TABLE I: Energy for Fused Multiply-Add benchmark

Double-precision	187 nJ/op
Single-precision	70 nJ/op
Half-precision	47 nJ/op

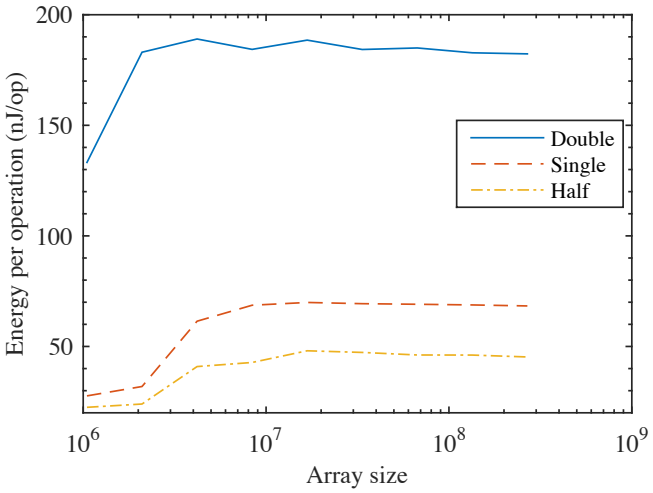
the various precisions using the double precision case as the baseline. Note that the total savings when going from double precision to half precision are a factor of 3.98.

V. Modeling and Explaining the Gains

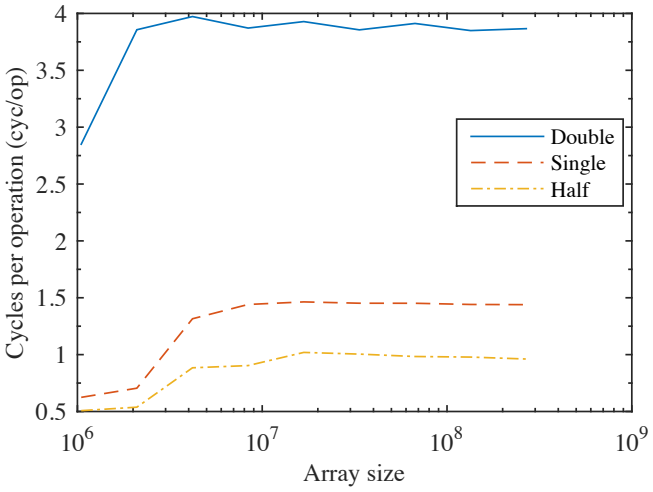
In order to understand the energy effects of various program designs, we must understand the source of the gain (or loss). The classic way to do that is to employ an energy *model*. Our energy modeling approach follows the methodology found in [1]. The inputs to the model are counts of the various instructions. To compute the energy from this count data, we multiply the count by the EPI for the instruction. Instructions naturally fall into certain static categories based on their EPI, e.g. integer, floating point, memory, etc. Memory-referencing instructions, however, behave differently according to whether the reference was in cache or not. So, a dynamic energy model would count the number of cache hits/misses. A further refinement associates the cache hits/misses into cache levels in the memory hierarchy. A reasonably refined energy-modeling tool, then, consists of 2 elements:

1) A method to count the various instruction categories, including all memory-reference refinements.
2) Data on the EPI of the various instruction categories.

For the instruction-counting element of our energy-modeling paradigm, we use *Cachegrind*, a *Valgrind* tool [16]. As previously noted, however, the instruction count should distinguish memory instructions from non-memory instructions, so that cache effects can be accounted for. is as a basis for counting the cache effects. We note that *Cachegrind* is restricted to 2 levels of cache hierarchy. So L2 and L3 cache hits/misses are combined into the Lm category of *Cachegrind*. Another limitation of *Cachegrind* is that it does not distinguish vector



(a)



(b)

Fig. 2: Energy per operation and cycles per operation of the vector fused multiply-add benchmark.

instructions from scalar instructions. So, for the moment, our instruction counting mechanism uses *Cachegrind* primarily to account for the cache behavior of the memory-referencing instructions.

The microbenchmarks we used were sufficiently small that we could simply look at the loop body to count non-memory instruction classes (such as scalar and vector) by hand. By knowing the ratio of various instruction classes to total instructions in the loop body, we could estimate the instruction class counts by simple multiplication. When larger programs are under consideration, we are planning to augment Cachegrind to distinguish instruction class in the counts.

For the second element of our energy-modeling tool, we obtained the EPI of almost all the instruction classes by using the RAPL microbenchmark techniques from section IV. Those results are shown in Table II. We confirmed most of our

measurements with the results in [8]. We were unable to confirm the energy cost of the"half-to-single-convert" in the literature. The value we are using in our analysis is conservative (possibly over-estimated). In addition, the L2 and L3 entries in the table are drawn from the literature, rather than from our measurements.

A. Analyzing the Fused Multiply-Add Microbenchmark

We used *Cachegrind* to count the number of cache hits/misses for two levels of cache (L1 is distinct from L2 and L3). The L2 & L3 cache activity was so small, however, that we ignored it in the energy modeling of the microbenchmark.

Non-data instructions, i.e. vector arithmetic operations, and vector conversions (half precision to single precision and reverse) could easily be counted by hand directly in the source code of each microbenchmark. We extrapolated to whole program counts by simple scaling. For all 3 precisions, the microbenchmark cache hits are effectively in L1. The estimated numerical counts of the instruction classes appear in Table III.

We begin our analysis of results by using our model to analyze the gains enjoyed when switching from double precision to single precision. First, compare the "Total Ins." entries for single and double precision. The comparison shows that single precision executes half the instructions that double precision executes. Therefore, no matter what the energy cost of any given instruction, the overall savings accruing from replacing double precision with single precision must be at least a factor of 2. The savings is a direct effect of vectorization: single-precision SIMD vectors hold twice as many values as double-precision SIMD vectors. There is, however, an additional savings realized by using single precision instead of double precision. Notice that the fraction of cache misses for single precision is smaller by about a factor of 3. The combination of increased cache effectiveness plus overall reduced instruction count yields better than a factor of 2 in energy savings for the double-to-single case. Our energy model for the double-to-single case gives a ratio of 2.92. The as-measured ratio from our RAPL framework is 2.67.

When comparing single to half, note that both the number of memory references as well as the number of fused multiply-add instructions is the same. So there is no "factor-of-2" effect to generate large savings. Furthermore, half precision, must execute the conversion instructions. As can be seen in the "Total Ins." line of the table, the half precision variant executes *more* instructions than the purely single variant. Notice, however, that the cache misses for half-precision are reduced by a little more than a factor of 2. This cache effect is the readily-realized benefit of half precision. This favorable cache behavior is enough to overcome the conversion instructions plus still produce a savings. According to our energy model, the ratio between single precision and half precision is 1.19. The as-measured ratio from our RAPL framework is 1.49.

B. Abstracting Energy Costs through Virtual Joules

All results described so far are specific to the Intel core i7. It would be desirable, however, to consider approaches that are vendor independent. To this end, in [1], the concept of *virtual joules* was introduced. The virtual joule concept enables the comparison of energy savings in a vendor-neutral manner. This model simplifies the energy estimation task by regrouping several instruction having equivalent energy costs, normalized to the least expensive operations. Table II also shows the Virtual Joules, though in this particular case vJ= nJ/ 2. Further details on this method can be found in [1].

TABLE II: EPI for Power Model

Instruction	EPI (nJ)	Virtual Joules(vJ)
Integer ADD (scalar)	2	1
Single ADD (scalar)	4.9	2.45
Double ADD (scalar)	5	2.5
128 Vector ADD (singles)	7.5	3.75
256 Vector ADD (singles)	9	4.5
NOP	1.4	0.7
Half to Single vector convert (8 values)	20	10
Main Memory Integer load (scalar)	183	91.5
Main Memory Single load (scalar)	195	97.5
Main Memory 128 vector load	199	98
Main Memory 256 vector load	204	102
L1 Access (from literature)	3	1.5
L2 Access (from literature)	11	5.5
L3 Access (from literature)	16	8

TABLE III: Instruction count for the three microbenchmark running as single processes with an array of 67 million elements and 44 consecutive runs

	Double	Single	Half
Cache hits	1,845,496,308	1,107,298,372	1,291,849,672
Main Memory	1,107,296,473	369,098,968	184,549,898
Vector FMA	738,197,504	369,098,752	369,098,752
Vector convert	0	0	1,476,395,008
Total Ins.	3,690,990,285	1,845,496,092	3,321,893,330
Modeled Energy	230,686,771,044	78,987,182,476	66,248,922,536

VI. Lessons Learned and Remarks

The important lesson from this study is that the energy gains derived from reduced-precision computation have three sources:

1) Vectorization of arithmetic: Twice as many values are processed per cycle for single precision, as compared to double precision
2) Vectorization of memory accesses: Smaller precision means more elements per load/store. This is the biggest source of gains.
3) Improved cache utilization: Smaller precision means more values can fit in the cache. This increases the number of cache hits.

In our study, the main gain comes from the reduction in memory traffic. As communication is expected to consume an increasing fraction of the total compute energy, the savings are likely to be even more significant in the future. An added advantage of lower precision is the need for less

memory, which is an important consideration, since future high-performance computing systems are likely to have a worse flop to DRAM ratio than do current designs. We note, in passing, that the energy model we used to analyze our results tend to overestimate or underestimate the savings compared to physically measured values. Therefore, we advocate developing additional refinements to improve model accuracy.

While the potential for savings as shown in this paper are interesting, the opportunity that we see here is an ability to use precision to virtually reduce the effects of technology scaling and the power wall citeNCSI. Thus, we believe that it will be very interesting to consider algorithms being *redesigned* so that the saved energy can be *reinvested* and used to actually improve the application's quality. Classical supercomputing workloads that have the flavor of iterative solutions [17] as well as weather and climate models [18, 19] are prime candidates for using inexactness to save in the first instance through lowered precision in the first instance, but then reinvest the saved energy in a different part of the algorithm to improve the overall quality. Consequently, the total energy budget will be the same but however the application could be re-engineered to achieve a higher quality solution through this approach. If successful, this approach can be used to mitigate the many hurdles and concomitant costs associated with deploying the next generation of exascale supercomputers by effectively achieving the quality goals that such scaling would imply from an application standpoint through current COTS systems!

REFERENCES

[1] P. Duben, J. Schlachter, Parishkrati, S. Yenugula, J. Augustine, C. Enz, K. Palem, and T. N. Palmer, "Opportunities for energy efficient computing: A study of inexact general purpose processors for high-performance and big-data applications," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE '15)*, 2015, pp. 764–769.

[2] "National strategic computing initiative strategic plan," July 2016. [Online]. Available: https://www.whitehouse.gov/sites/whitehouse.gov/files/images/NSCI Strategic Plan.pdf

[3] K. V. Palem, "Compilers, architectures and synthesis for embedded computing: retrospect and prospect," in *Record of the IEEE W. Wallace McDowell Award Lecture, Proceedings of the ACM-IEEE Inernational Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, 2010.

[4] K. Palem and A. Lingamneni, "Ten years of building broken chips: The physics and engineering of inexact computing," in *ACM Transactions on Embedded Computing Systems*, vol. 12, no. 2s, May 2013, pp. 87:1–87:23.

[5] K. V. Palem, "Proof as experiment:probabilistic algorithms from a thermodynamic perspective," in *Proceedings of the Intl. Symposium on Verification (Theory and Practice)*, June 29-July 4, 2003.

[6] S. Cheemalavagu, P. Korkmaz, and K. V. Palem, "Ultra low-energy computing via probabilistic algorithms and devices: CMOS device primitives and the energy-probability relationship," in *Proceedings of the 2004 Intl. Conference on Solid State Devices and Materials (SSDM), September 14-17, 2004.*, 2004.

[7] K. V. Palem, "Energy aware computing through probabilistic switching: A study of limits," *IEEE Transactions on Computers*, 2005.

[8] Y. S. Shao and D. Brooks, "Energy characterization and instruction-level energy model of Intel's Xeon Phi processor," in *Proceedings of the 2013 International Symposium on Low Power Electronics and Design*, 2013, pp. 389–394.

[9] A. Buttari, J. Dongarra, J. Kurzak, P. Luszczek, and S. Tomov, "Using mixed precision for sparse matrix computations to enhance the performance while achieving 64-bit accuracy," *ACM Transactions on Mathematical Software (TOMS)*, vol. 34, no. 4, p. 17, 2008.

[10] D. Hackenberg, T. Ilsche, R. Schöne, D. Molka, M. Schmidt, and W. E. Nagel, "Power measurement techniques on standard compute nodes: A quantitative comparison," in *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 194–204.

[11] C. R. Gonzalez, C. Nguyen, H.-D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey, C. Iancu, and D. Hough, "Precimonious: Tuning assistant for floating-point precision," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'13)*, 2013.

[12] C. Lomont, "Introduction to intel advanced vector extensions," *Intel White Paper*, 2011.

[13] W. A. Wulf and S. A. McKee, "Hitting the memory wall: implications of the obvious," *ACM SIGARCH computer architecture news*, vol. 23, no. 1, pp. 20–24, 1995.

[14] P. Kogge and J. Shalf, "Exascale computing trends: Adjusting to the "new normal" for computer architecture," *Computing in Science and Engg.*, vol. 15, no. 6, pp. 16–26, Nov. 2013.

[15] "Intel 64 and IA-32 Architectures Software Developer Manuals." [Online]. Available: http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

[16] "Valgrind Home." [Online]. Available: http://valgrind.org/

[17] J. E. Dennis, Jr and R. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, 1983.

[18] T. Palmer, "Modelling: Build imprecise supercomputers," *Nature*, September 29 2015.

[19] J. Markoff, "A climate-modeling strategy that wont hurt the climate," *The New York Times*, May 11, 2015.