2017 IEEE International Conference on Cluster Computing

Thoughtful Precision in Mini-apps

Shane Fogerty*, Siddhartha Bishnu*, Yuliana Zamora*†, Laura Monroe*
Steve Poole*, Michael Lam‡, Joe Schoonover§, and Robert Robey*
*Los Alamos National Laboratory
†University of Chicago
‡James Madison University
§Cooperative Institute for Research in Environmental Sciences (CIRES)
Emails:{sfogerty,siddhartha.bishnu,lmonroe,swpoole,brobey}@lanl.gov,
yzamora@uchicago.edu, lam2mo@jmu.edu, Joseph.Schoonover@colorado.edu

***Abstract*—Approximate computing addresses many of the identified challenges for exascale computing, leading to performance improvements that may include changes in fidelity of calculation. In this paper, we examine approximate approaches for a range of DOE-relevant computational problems run on a variety of architectures as a proxy for the wider set of exascale-class applications.**

We show anticipated improvements in computational and memory performance and in power savings. We also assess application correctness when operating under conditions of reduced precision, and show that this is within acceptable bounds. Finally, we discuss the trade space between performance, power, precision and resolution for these mini-apps, and optimized solutions attained within given constraints, with positive implications for application of approximate computing to exascale-class problems.

Keywords*—*Inexact computing, reduced precision, reduced precision arithmetic, mini-apps. Beyond Moore's Law, hardware/software codesign

I. Introduction

High-performance computing is reaching a point at which simply adding nodes and hoping to scale is not a sustainable model, because of power, reliability and other concerns. In addition, the impending post-Moore's-Law era and the anticipated end of CMOS scaling necessitate new and novel approaches to computation. Inexact computing applies to a wider range of these novel computing paradigms, as well as to pre- and post-exascale conventional computing.

Inexact computing can be either probabilistic or deterministic. Unlike probabilistic computing, deterministic computing produces the same result for each run, which may or may not be precise, but differs from the exact result by the same amount. In this paper, we study reduced precision computation, a deterministic form of inexact computing.

Inexact computing addresses many, if not most, of the ten challenges discussed in the DOE Advanced Scientific Computing Advisory Committee Subcommittee Report on the Top Ten Challenges for Exascale [1], including energy efficiency, data size (impacting bandwidth and storage requirements), memory use, computational algorithms, resilience and correctness and scientific productivity.

For this paper, we ran DOE-relevant scientific mini-applications approximately using reduced precisions, thereby saving power and enhancing performance with little impact to computational fidelity. Our work in this area is part of the DOE's Beyond Moore's Law program as part of the Presidential Executive Order for the National Strategic Computing Initiative (NSCI) Strategic Plan [2], intended to sustain and enhance U.S. leadership in high-performance computing (HPC) with particular reference to exascale and post-Moore's era goals.

II. Approximate Computing for HPC Applications

Our early investigations into inexact computing for HPC applications concentrate on power and performance gains and algorithmic changes, and at the same time, look at changes in computational fidelity and correctness that may be induced. Power is of especial interest as it is such a major obstacle: according to [1], the energy efficiency of current systems must improve by at least a factor of 8-10 to reach this stated goal of 20 MW for an exascale system.

We focus here on approximate computing rather than probabilistic computing, at least as a first attempt at inexact for HPC using DOE-relevant applications. We made this choice for several reasons: 1) because approximate computing is more compatible with current developer expectations, 2) because we suspected that it would be possible to perform scientific calculations at lower precision than is currently used, at least in parts of the calculations, and thought it possible to effect such changes with fairly minimal disruption to the code base, and 3) because we surmised that energy costs could be reduced in this manner without significantly impacting the results. All these reasons made approximate computing an attractive first target.

We note that scientific computing is already approximate, since all simulations are approximate. In fact, all floating point computing is approximate, since only finite precision is available on a digital machine. However, many scientific calculations use the entire precision furnished by the architecture just because it is available.

In contrast to this, the point of our work is to use limited precision not just to the extent imposed by the machine, but to reduce precision as far as one can. We attempt to run at the different precisions actually required, at the algorithmic and architectural component levels, but without impairing correctness to the point that the simulation results are unusable.

2168-9253/17 $31.00 © 2017 IEEE
DOI 10.1109/CLUSTER.2017.120

III. Prior Work

A. Inexact Computing

Many effective techniques and methods for approximate computing have emerged in the last decade that selectively approximate computation to obtain disproportionate gains in performance and storage cost reduction. These are described in detail in a 2016 review of the field by Mittal in [3]. These methods promise great benefit to a wide range of scientific computing and other applications.

However, to take full advantage of these methods, careful selection of application-specific approximation tactics is required. In addition, the results of applications using approximate computing must be carefully examined to ensure that the results are "close enough", meeting scientific standards as per an application-appropriate distance metric.

B. Reduced Precision

There have been many previous mixed-precision versions of numeric codes [4–15]. These efforts demonstrate that in some cases, mixed-precision code can achieve similar accuracy to the original double-precision code while being significantly faster and reducing memory pressure. These efforts span many problem domains, such as dense linear algebra routines [4, 6], finite element methods [5, 11], fluid dynamics [8], lattice quantum chromodynamics [7], atmospheric models [9], Monte Carlo simulations for financial market analysis [10], and semi-Lagrangian methods [12]. Many of these efforts extend their work to accelerators such as GPUs or other many-core architectures like Intel Phi as well as re-configurable architectures such as FPGAs. There is also at least one effort that has started with a mixed-precision code and improved it further by enhancing the numerics [13]. Duben et al. [14] showed potential energy savings by the use of reduced precision techniques on PageRank and on the Intermediate General Circulation Model (IGCM) climate modeling application run on a simulated architecture, and in [15], ran a simple fused-multiply-add benchmark on a currently available Intel core i7 4770 (3.40 GHz) with 16 GB of DDR3 RAM.

More recently, some groups have built tools for analyzing numerical codes with the goal of automatically building mixed-precision versions or in some way informing their development. Dynamic approaches include instruction-based analysis tools such as CRAFT [16, 17], variable-based analysis tools such as Precimonious [18] and Blame Analysis [19], and sampling-based approaches such as Herbie [20]. Static approaches include Taylor-expansion-based tools such as FP-Tuner [21] and SMT-based tools such as Rosa [22]. All of these tools provide recommendations for reducing precision while maintaining a specified amount of accuracy. In some cases, such configurations must be realized using manual code transformations for performance testing.

Again, approximate approaches vary among computational problems and computer architectures. In this paper, we target DOE-specific and DOE-relevant exascale-related mini-apps, and we run on an assortment of commercial off-the-shelf (COTS) platforms that are available now at DOE laboratories while looking towards future trends in hardware in support of codesign efforts.

C. Reproducible Global Sums

Precision reduction by itself may not be the best and most accurate approach. Instead, we focus on choosing the level of precision according to the needs of the calculation, and even increasing precision, if needed. Increasing precision in well-chosen sub-calculations can then enable the rest of the calculation to be done at lower precision.

Work starting in about 2010 by several research groups indicated that the most sensitive parts of numerical calculations involved global sums across the entire computational domain. Studies by Robey [23], Demmel and Nguyen [24], Chapp [25], and Iakymchuk [26] show that the typical error in global sums can be reduced from about 7 digits of precision to 15 digits, within a few bits of perfect reproducibility.

IV. Methodology

Throughout this work we seek to understand the impact of inexact methods, in this case by varying the precision of calculations for scientific applications that reflect the workloads on DOE computing systems. We would like to address the effects that reducing precision has on the power consumption and cost of the computation as well as the validity and correctness of the computational solution.

We approach this as a true codesign problem: we are developing the algorithmic approaches to DOE Advanced Simulation and Computing (ASC) problems at the same time that we are exploring emerging inexact architectures that will natively give the promised power and performance benefits. Two fundamental questions are:

- Which applications and algorithms benefit from inexact methods and the power savings and resilience they afford?
- Which architectures will support these applications most efficiently?

These applications and architectures are the ones that will gain from the power and resilience benefits inexact computing will provide.

We have selected a DOE mini-app and another relevant mini-app that are amenable to varying precision to get a real-world assessment of potential advantages and disadvantages of lower precision. Other applications may show different effects and may even need a different implementation methodology.

We start by using existing standard precision formats commonly available in current computing hardware as given in the IEEE 754 standard [27]. We analyze results for performance and storage benefits and for correctness, and will then partition the application for more targeted benefits. This is sufficient for a preliminary assessment of impact on a small set of mini-apps.

A. CLAMR

CLAMR is a hydrodynamic, cell-based adaptive mesh refinement (AMR) mini-app developed at Los Alamos National Laboratory (LANL). The CLAMR code simulates fluid motion using the Shallow Water equations, and represents common computational methods used in many important scientific computing efforts [28]. CLAMR is open-sourced and used as a test-bed for hybrid algorithm development using MPI and OpenCL

GPU code [29]. Having different compile options for different precision levels make CLAMR an ideal mini-app to run further tests.

B. SELF

The Spectral Element Libraries in Fortran (SELF) is a set of Fortran modules that define data structures and procedures that facilitate rapid implementation of Spectral Element Methods for solving a variety of scientific problems [30]. This framework includes tools for specifying an isoparametric mesh, computing derivatives in curvilinear coordinates, spectral filtering, and time integration. Following the generalized formulation of a conservation law, as in Kopriva (2009), the SELF provides sufficient tools for quickly implementing approximate solvers of partial differential equations. In this paper, we focus on an implementation of the SELF that solves the 3-D Compressible Navier-Stokes equations.

C. Experimental Process

The IEEE 754 standard defines finite binary formats, called floating-point numbers, for representing real numbers. The basic formats for floating-point numbers have varying widths corresponding to different levels of precision - 16 bits (half precision), 32 bits (single precision), 64 bits (double precision), and more.

Compile options are available in CLAMR for setting precision specifications (minimum, mixed, or full precision), which were produced by the precision analysis of Lam and Hollingsworth [17]. Minimum precision sets single precision throughout the code. Mixed precision sets the large physical state arrays to single precision, but promotes all local calculations to doubles. The goal is to save storage space while keeping as much precision as possible elsewhere. Full precision sets everything to double precision in the numerical calculations. In all of these, graphics and plotting calculations are kept at single precision since the resolution of screens and plotters cannot benefit from higher precision.

D. Vectorization

In the past 15 years, Streaming SIMD Extensions (SSE) have increased registers from support of operations on just two double-precision floating-point numbers, to Advanced Vector eXtensions (AVX) supporting operations on eight double-precision floating-point numbers on current architectures like the Xeon Phi x200 (Knight's Landing). This trend highlights the importance of vectorization in optimizing applications.

For the CLAMR runs, it was observed that the majority of CPU time spent on floating-point arithmetic lies within the finite-difference algorithm loop. This was not originally vectorized, so we generated optimization reports using Intel compiler flags and added OpenMP SIMD pragma statements to enable vectorization of the *finite_diff* loop, which performs many floating point operations.

E. Architectures

The mini-apps were tested on both Intel processors and Nvidia GPUs. More specifically, Intel Haswell Xeon Processor E5-2660 v3, Intel Broadwell Xeon Processor E5-2695 v4, Nvidia GPU Tesla K40m, Nvidia Quadro K6000, Nvidia Tesla P100 SXM2-16GB and GeForce GTX TITAN X were tested.

TABLE I. SINGLE PRECISION IMPROVES CLAMR RUNTIMES AND REDUCES MEMORY USE

Arch.	Memory Usage (GB)			Runtime			Speedup
	Min	Mixed	Full	Min	Mixed	Full	
Haswell	1.59	1.60	1.66	26.3	29.9	31.3	19%
Broadwell	1.59	1.59	1.66	25.3	31.0	31.4	24%
Tesla K40m	0.50	0.50	0.52	4.9	12.8	12.8	261%
Quadro K6000	0.50	0.50	0.50	4.2	10.6	10.6	252%
GTX TITAN X	0.50	0.52	0.58	2.8	12.5	12.7	453%

Many architectures were tested to assess the effect of hardware choice on reduced-precision computation, toward a goal of true codesign. Variations in the relative efficiency of single and double precision floating-point computation on a given choice of hardware should influence performance of reduced-precision computing, especially on GPUs where this variation can be large. Insights into hardware design preferences for reduced-precision computing may help inform codesign of future hardware to meet the challenges of exascale computing.

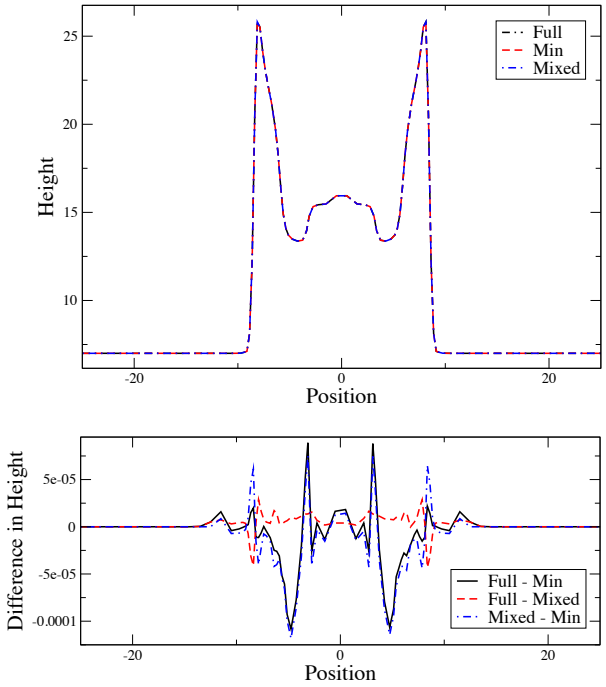

Fig. 1. Comparison shows slices of CLAMR simulation results are nearly identical for each precision level (top) with 64 grid points and 2 levels of AMR. Differences between full and mixed precision results are smallest (bottom)

V. RESULTS

A. CLAMR

We ran a cylindrical dam break problem in CLAMR on the Haswell CPU with both a unvectorized and vectorized version for a 64×64 and 128×128 grid with 2 levels of AMR. The result for the first run is depicted in Figure 1, where we plot the solution after 1000 iterations and study the difference

TABLE II. ESTIMATED CLAMR ENERGY USE ON DIFFERENT ARCHITECTURES

Architecture	CLAMR Energy Use (Joules)		
	Min	Mixed	Full
Haswell	2762	3140	3287
Broadwell	3033	3725	3762
Tesla K40m	1054	2752	2752
Quadro K6000	945	2385	2385
GTX TITANX	700	3125	3175

among the solutions obtained along a cut-line passing through the center of the domain for different levels of precision.

The performance gains and memory usage are shown in Table III for varying levels of precision. The simulations used an initial coarse grid with 1920×1920 cells, with a maximum of 2 levels of AMR, and were run for 200 iterations. All were run on Intel Haswell E5-2660_v3 processors. Initial, unvectorized runs using minimum precision show some modest performance gains (about 12% speedup) over runs using full precision for both the total CPU time and the *finite_diff* function. When computing scalar floating point operations, the reduction in precision has a limited impact on performance gain. SIMD functions were implemented to vectorize *finite_diff* in CLAMR, improving performance greatly. This change highlights the performance gains of single precision calculations, with a speedup of 1.9x in *finite_diff* compared to full precision.

Although computation time when using mixed precision was not significantly improved over using full precision, reduction in memory use and output file size was notable. For both minimum and mixed precision, checkpoint output file sizes were about 2/3 of those when using full precision. Memory usage during the simulations was also reduced when using less than full precision.

With a view to comparing the CLAMR results for three different levels of precision, we study the numerical solution on a vertical line-cut passing through the center of the domain at the same iteration for each run. Figure 1 plots the fluid height, as a function of the position along the line-cut, for different resolution runs. It is observed that the plots for all precision levels are visually indistinguishable. This figure also depicts the differences in height among the runs at different precision levels, which are typically at least five to six orders of magnitude less than the magnitude of the height. From this observation, we can conclude that a lower precision run does not change the numerical solution to an appreciable extent to be detected by the naked eye. The peak differences between computed heights for full and minimum precisions are the largest, whereas mixed precision produces remarkably similar results to those of full precision. This makes mixed precision a compelling choice for CLAMR simulations, as it can not only save memory and storage, but can also result in a numerical solution almost identical to its full precision counterpart. In order to study the symmetric nature of the (ideally symmetric) solution, we consider the numerical solution on each half of the line-out. Extending from the left end all the way to the center of the line-out, we plot the difference in the numerical solution at every point, from that on the other half of the line-out, equidistant from the center. This plot depicted in Figure 2 illustrates that a reduced precision run amplifies the asymmetry

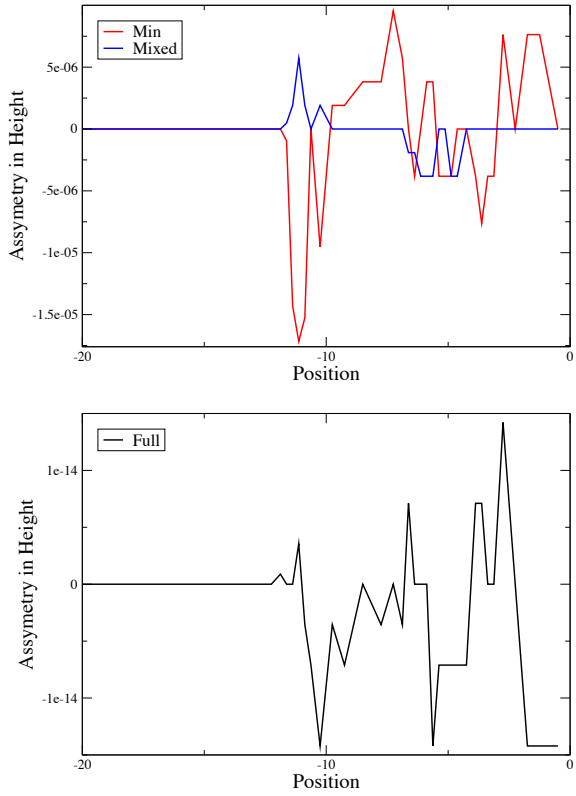

Fig. 2. Height Asymmetry for the CLAMR simulations

of the numerical solution. But even in minimum precision, the magnitude of the differences are at least a factor of 10^{-6} less than that of the solution.

Gains made in performance when using lowered precision can be reinvested in other (often more precious) resources. For example, a minimum-precision run could employ a greater number of cells or levels of AMR, while keeping performance near that of a full-precision run with less resolution. To test this concept, we run a full-precision-low-resolution (Full-LoRes) version as well as a minimum-precision-high-resolution (Min-HiRes) version of CLAMR and plot the simulation slices in Figure 3. We note that these two runs are not exactly equivalent to each other due to complications introduced by AMR and different time steps determined by using the same Courant number. However, we compute the solutions at almost the same instant of simulation time, scale the Min-HiRes run and plot them in the same graph for ease in visualizing their differences. Even though the solutions do not exactly align themselves for the above-mentioned reasons, it is clear that the Min-HiRes solution has a more detailed structure than the Full-LoRes one. In this way, we can combine lower precision with higher degrees of freedom, resulting in a better solution.

Different architectures were tested with CLAMR at all three precision levels. Table I shows CLAMR runtimes improved on all of these architectures, and better performance on CPUs with reduced precision, as well as increased speedup on GPUs. Memory use was consistently decreased with reduced precision on all architectures. CLAMR energy use was estimated by multiplying nominal power specifications by

TABLE III. CLAMR PRECISION COMPARISONS AND VECTORIZATION

	Min. Precision	Mixed Precision	Full Precision
finite_diff time $unvectorized$	11.4	12.3	12.7
finite_diff time $vectorized$	4.8	8.9	9.2
Checkpoint file size	86M	86M	128M

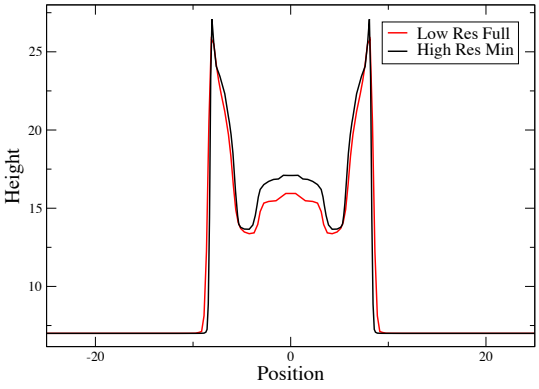

Fig. 3. Comparison of CLAMR simulation slices between a minimum precision, high resolution run and a similar full precision, low resolution run

runtimes, and show significant energy savings when using reduced-precision modes, as shown in Table II.

B. SELF

In this study, the SELF simulates an anomalous warm blob that rises in an otherwise neutrally buoyant fluid, similar to the initial condition in [31]. The domain consists of $20 \times 20 \times 20$ elements, each with $8 \times 8 \times 8$ quadrature points giving roughly 24 million degrees of freedom. When comparing run-time between single and double precision, the intrinsic `CPU_TIME` routine is placed around the main time integration loop, which calls a 3rd-order Runge-Kutta time integrator 100 times.

1) CPU: SELF was tested on two nodes with different architectures. The first node was an Intel Haswell E5-2660_v3 CPU and the second, an Intel Broadwell E5-2695_v4 CPU. We initially tested a non-vectorized version of SELF with the example thermal bubble problem using both Intel (v.17.0.0) and GNU (v.4.9.3) compilers. Interestingly, the total run-time was less for double precision than for single precision with the GNU compiler on Haswell and Broadwell. This result, though unexpected, indicates that reduced precision does not always reduce run-time. Researching this issue is beyond the scope of this paper but it definitely merits further investigation. With the Intel compiler, however, the single precision run was less time-consuming than the double precision one as expected. These runtimes are shown in Table IV.

TABLE IV. NONVECTORIZED SELF CONSUMES LESS RUNTIME FOR DOUBLE PRECISION THAN FOR SINGLE PRECISION WITH GNU COMPILER

	Single Precision Runtime (s)	Double Precision Runtime (s)
GNU	304.09	261.65
Intel	185.89	252.85

Adopting the same approach as with CLAMR, we took a horizontal line-out of the numerical solution passing through the center of the domain and plotted the density anomaly

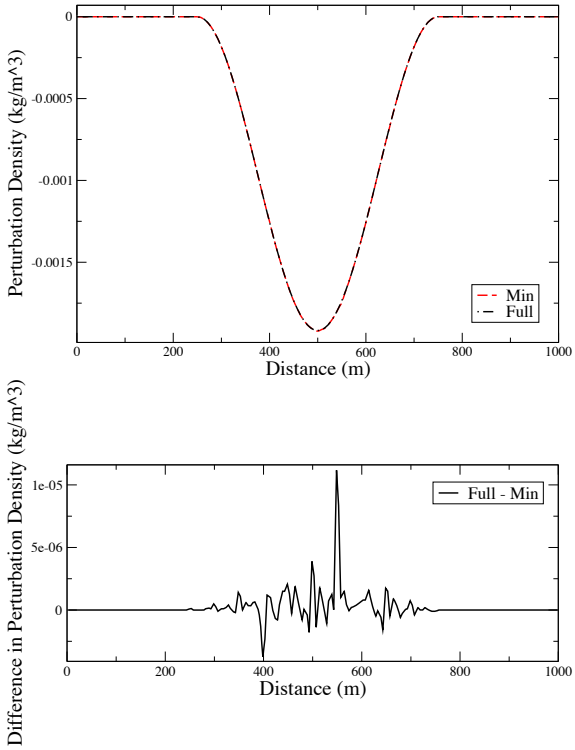

Fig. 4. Slice of SELF simulations for single and double precision levels with 20 elements and polynomial basis functions of order 7 in each direction (above) and the difference between them (below)

in Figure 4 for the single and double precision runs. As in CLAMR, it is observed that the solutions for the two precision levels are visually identical. The absolute difference $(\sim \mathrm{O}(10^{-5}))$ between the solutions for these precision levels is two orders of magnitude less than the solution. The difference between the density anomalies on either side of the center of the line-out for each precision level are plotted in Figure 5, demonstrating that for double precision, the asymmetry oscillates frequently about the x-axis and assumes almost equal number of positive and negative values with similar magnitude. However, for the single precision run, the asymmetry is mostly positive, implying that the solution on the right half of the line-out is larger than the corresponding left-half solution. As of now, we do not have a clear explanation behind the nature of these plots. We suspect interpolation errors may have played a role, which needs to be further investigated.

2) GPU: SELF performance gains with double and single precision on GPUs are presented in Table V. For each run, SELF was uses OpenACC on the GPUs with PGI 16.10 and CUDA 8.0, and GCC 6.3.0 on the CPUs. Single precision reduced run-times by roughly 30-35% on the Tesla K40m, the Quadro K6000, and the Tesla P100 SXM2. However, the speedup on the GeForce GTX TITAN X when using single precision was 3x when compared to double precision. In fact, SELF with single precision on the TITAN X outperformed SELF using double precision on the P100. The TITAN X is unique in our testing suite because its specifications have a ratio of single-precision Gflops to double-precision Gflops of 32:1, while this ratio can be up to 2:1 for GPUs specifically

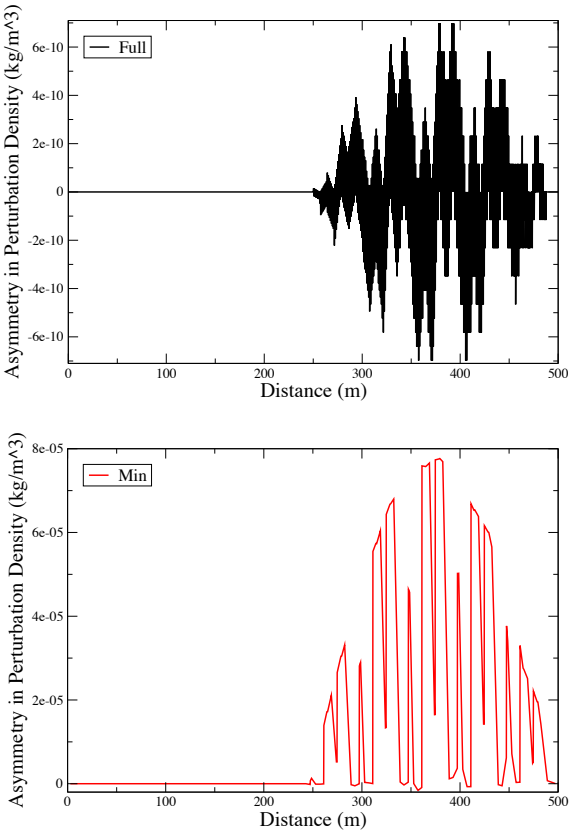

Fig. 5. Asymmetry in Perturbation Density for the SELF simulations

TABLE V. SINGLE PRECISION IMPROVES SELF RUNTIMES AND REDUCES MEMORY USE

Arch.	Memory Usage (GB)		Runtime		Speedup
	Single Precision	Double Precision	Single Precision	Double Precision	
Haswell	2.7	2.7	179.5	270.4	51%
Broadwell	3.0	5.4	184.1	224.2	22%
Tesla K40m	2.3	4.4	40.1	53.7	34%
Quadro K6000	2.3	4.4	32.6	42.6	31%
Tesla P100	2.5	4.7	13.5	17.3	28%
GTX TITANX	2.3	4.5	16.1	49.7	309%

TABLE VI. SELF ON DIFFERENT ARCHITECTURES

Architecture	SELF Energy Use (Joules)	
	Single Precision	Double Precision
Haswell	18795	28350
Broadwell	22080	26880
Tesla K40m	8617	11546
Quadro K6000	7335	9585
Tesla P100	3375	4325
GTX TITANX	4025	12425

TABLE VII. COST MODEL

	Minimum Precision	Mixed Precision	Full Precision
CLAMR Compute Cost	$223.22	$257.10	$267.07
CLAMR Storage Cost	$121.66	$121.66	$181.56
CLAMR Total Cost	$344.88	$378.76	$448.63
SELF Compute Cost	$763.32	-	$1157.94
SELF Storage Cost	$792.59	-	$792.59
SELF Total Cost	$1555.91	-	$1950.53

targeted to the scientific computing market.

Performance results with SELF on GPUs shows the benefits of single-precision floating-point. Since the scaling of many modern applications is limited not by flops, but by memory bandwidth, it makes sense to run applications with single-precision calculations to save on memory use. This is emphasized in Table V, where for our SELF runs with single-precision, a TITAN X overcomes the generational divide and competes well with a Tesla P100. Given the similar performance of these GPUs and much higher costs of purchasing the newer P100s, resources can be conserved in HPC centers by investing in at least a partition with cheaper GPUs like the TITAN X for high-performance approximate computing.

Power consumption was estimated for different architectures for CLAMR and SELF with nominal specifications. By multiplying this rough estimate of power by the runtime, we estimated the approximate energy used for similar runs at different precision levels, as shown in Table VI.

VI. COST ANALYSIS

Cost analyses for high-performance computing must account for the energy use of running a calculation on a typical HPC installation, and it may also be desirable to include a share of the hardware purchase cost and operational staffing cost in the analysis. We start with the latter approach by using the rates charged by commercial services.

Conveniently, Amazon Web Services has a "Monthly Cost Calculator [32]." We chose EC2 and S3 services as the most applicable. We ignore data retrieval and transfer since we are assuming on-site usage of our own HPC installation [33]. Applying all these factors, the minimum/mixed precision data is about 2/3 the size of the full-precision values and the cost is shown in Table VII.

The EC2 "c4.8xlarge" instance with Haswell family processors and 60 GB RAM was selected as most similar to other runs on more traditional HPC machines. Instances were chosen with utilization estimated using our tested run-times on the Haswell architecture, scaled up from seconds to hours per week. Storage costs are an important component of budgeting computing needs. In this AWS estimate, we used the S3 service and scaled the normal storage and infrequent access storage with the same factor as the compute time, but then reduced by a factor of five to account for longer runs with fewer output files. SELF costs were much more expensive, and so we scaled the compute time down by 50%, as well as assuming the data would be stored less frequently, reducing the storage amount by a factor of ten. Data retrieval and transfer were not included for simplicity, and support costs were ignored. Although floating point compression can produce impressive storage savings [34], additional compute costs for the compression algorithm would complicate this simple cost model and so compression is not considered. Though the choices for cost models may change with each user, those presented here are chosen to be representative of a typical application user's needs.

A cost model was obtained for both applications using the Amazon Web Services cost modeling tool. The costs shown in Table VII give a clear picture of the savings expected when employing approximate computing. CLAMR shows a savings of up to 23% of costs using minimum precision instead of full precision and a 15% in savings using mixed precision instead of full precision. SELF shows a savings of up to 20% using a minimum precision implementation instead of full precision. SELF does not have a mixed-precision option currently.

VII. Conclusion

We have demonstrated using real compute systems that are now coming available that in two different DOE-relevant mini-applications, reduced precision implementations can reduce storage costs and increase performance by a factor of 2-3 times, with only modest changes to the application code base. For implementations that use double precision and have large percentages of floating point operations the results will likely be similar.

The strong coupling of gains from improved data motion through vectorization and faster computation speed has been shown. Though this coupling is difficult to separate, we believe memory bandwidth strongly limits representative applications, so speedups shown are primarily due to improved data motion.

As a result of these precision changes, these codes can run on a smaller number of nodes, use less storage hardware and, most importantly, reduce power consumption for a given simulation.

The benefits of having selectable precision levels within a code base has also been shown. Users can choose the precision suitable for the particular simulation they are running.

When the cost of a double-precision floating point operation was the same as a single-precision operation, there was only a small motivation to use anything but double precision. Making smart decisions on precision is predicated on the availability of different precisions in the hardware. Architectures are now emerging that afford reduced precision with commensurate power and performance savings.

It is time for application developers to jump on this disruptive trend in computing capabilities.

VIII. Future Work

The goal of this project is an understanding of the algorithms and architectures that are relevant to DOE mission space. The DOE mini-apps provide a rich opportunity to explore impacts of reduced precision in representative applications across a broad array of algorithm types.

This includes the derivation of heuristics for precision choice, at the algorithm and sub-algorithm levels. The selection of the correct precision to use in each step of an algorithm requires a strong understanding of the method and also benefits from years of experience. We will extend this work to look at a broad range of mini-apps with different classes of algorithms. The approaches to reducing precision will certainly vary, although common patterns will be encountered.

We also need to understand the architectural advances anticipated in coming years that will allow us to leverage these techniques in the most efficient manner. We note that new hardware with many more precision choices will be available in the future. These may be driven by other application domains such as machine learning. We plan to explore how these hardware advances may be utilized in our community.

Acknowledgment

We would like to thank the Information Science and Technology Institute (ISTI), the Parallel Computing Summer Research Internship (PCSRI) program, the Darwin Cluster and the Beyond Moore's Law project at LANL for providing us with resources and support. Also we want to acknowledge Ronald W. Green's expertise and Vectorization presentation.

This work was supported by the Los Alamos National Laboratory under contract DE-AC52-06NA25396.

References

[1] ASCAC Subcommittee. Top ten exascale research challenges. *US Department Of Energy Report*, 2014.

[2] Barack Obama. Exec. Order No. 13,702, 80 Fed. Reg. 46177 (July 29, 2015) - Creating a National Strategic Computing Initiative. Technical report, Federal Register, US Govt, 2015.

[3] Sparsh Mittal. A survey of techniques for approximate computing. *ACM Comput. Surv.*, 48(4):62:1–62:33, March 2016.

[4] Xiaoye S. Li, James W. Demmel, David H. Bailey, Greg Henry, Yozo Hida, Jimmy Iskandar, William Kahan, Suh Y. Kang, Anil Kapur, Michael C. Martin, Brandon J. Thompson, Teresa Tung, and Daniel J. Yoo. Design, implementation and testing of extended and mixed precision BLAS. *ACM Transactions on Mathematical Software*, 28(2):152–205, jun 2002.

[5] Dominik Göddeke, Robert Strzodka, and Stefan Turek. Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in FEM simulations. *International Journal of Parallel, Emergent and Distributed Systems*, 22(4):221–256, aug 2007.

[6] Alfredo Buttari, Jack Dongarra, Jakub Kurzak, Piotr Luszczek, and Stanimire Tomov. Using mixed precision for sparse matrix computations to enhance the performance while achieving 64-bit accuracy. *ACM Transactions on Mathematical Software*, 34(4):1–22, 2008.

[7] M. A. Clark, R. Babich, K. Barros, R. C. Brower, and C. Rebbi. Solving Lattice QCD systems of equations using mixed precision solvers on GPUs. *Computer Physics Communications*, 181(9), 2010.

[8] Hartwig Anzt, Björn Rocker, and Vincent Heuveline. Energy efficiency of mixed precision iterative refinement methods using hybrid hardware platforms. *Computer Science Research and Development*, 25(3-4):141–148, 2010.

[9] L. Gan, H. Fu, W. Luk, C. Yang, W. Xue, X. Huang, Y. Zhang, and G. Yang. Accelerating solvers for global atmospheric equations through mixed-precision data flow engine. In *2013 23rd International Conference on Field programmable Logic and Applications*, pages 1–6, Sept 2013.

[10] C. Brugger, C. de Schryver, N. Wehn, S. Omland, M. Hefter, K. Ritter, A. Kostiuk, and R. Korn. Mixed

precision multilevel Monte Carlo on hybrid computing systems. In *2014 IEEE Conference on Computational Intelligence for Financial Engineering Economics (CIFEr)*, pages 215–222, March 2014.

[11] C. Richter, S. Schps, and M. Clemens. GPU-accelerated mixed precision algebraic multigrid preconditioners for discrete elliptic field problems. In *9th IET International Conference on Computation in Electromagnetics (CEM 2014)*, pages 1–2, March 2014.

[12] L. Einkemmer. A mixed precision semi-Lagrangian algorithm and its performance on accelerators. In *2016 International Conference on High Performance Computing Simulation (HPCS)*, pages 74–80, July 2016.

[13] Ichitaro Yamazaki, Stanimire Tomov, Jakub Kurzak, Jack Dongarra, and Jesse Barlow. Mixed-precision block gram schmidt orthogonalization. In *Proceedings of the 6th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, ScalA '15, pages 2:1–2:8, New York, NY, USA, 2015. ACM.

[14] Peter Düben, Jeremy Schlachter, Parishkrati, Sreelatha Yenugula, John Augustine, Christian 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, pages 764–769, San Jose, CA, USA, 2015. EDA Consortium.

[15] Mike Fagan, Jeremy Schlachter, Kazutomo Yoshii, Sven Leyffer, Krishna V. Palem, Marc Snir, Stefan M. Wild, and Christian C. Enz. Overcoming the power wall by exploiting inexactness and emerging COTS architectural features: Trading precision for improving application quality. In *29th IEEE International System-on-Chip Conference, SOCC 2016, Seattle, WA, USA, September 6-9, 2016*, pages 241–246, 2016.

[16] Michael O. Lam, Jeffrey K. Hollingsworth, Bronis R. de Supinski, and Matthew P. Legendre. Automatically Adapting Programs for Mixed-Precision Floating-Point Computation. In *Proceedings of the 27th International ACM Conference on Supercomputing (ICS '13)*, page 369, New York, New York, USA, jun 2013. ACM Press.

[17] Michael O Lam and Jeffrey K Hollingsworth. Fine-grained floating-point precision analysis. *The International Journal of High Performance Computing Applications*, pages 1–15, 2016.

[18] Cindy Rubio-González, Cuong Nguyen, Hong Diep Nguyen, James Demmel, William Kahan, Koushik Sen, David H. Bailey, Costin Iancu, and David Hough. Precimonious: Tuning Assistant for Floating-Point Precision. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on (SC'13)*, pages 1–12, New York, New York, USA, nov 2013. ACM Press.

[19] Cindy Rubio-González, David Hough, Cuong Nguyen, Benjamin Mehne, Koushik Sen, James Demmel, William Kahan, Costin Iancu, Wim Lavrijsen, and David H. Bailey. Floating-point precision tuning using blame analysis. In *Proceedings of the 38th International Conference on Software Engineering - ICSE '16*, pages 1074–1085, New York, New York, USA, 2016. ACM Press.

[20] Pavel Panchekha, Alex Sanchez-Stern, James R. Wilcox, and Zachary Tatlock. Automatically improving accuracy for floating point expressions. *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation - PLDI 2015*, pages 1–11, 2015.

[21] Wei-Fan Chiang, Mark Baranowski, Ian Briggs, Alexey Solovyev, Ganesh Gopalakrishnan, and Zvonimir Rakamari. Rigorous Floating-Point Mixed-Precision Tuning. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL'17)*, pages 300—-315, New York, NY, USA, 2017. ACM.

[22] Eva Darulova and Viktor Kuncak. Towards a Compiler for Reals. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 39(2):8:1—-8:28, 2017.

[23] Robert W Robey, Jonathan M Robey, and Rob Aulwes. In search of numerical consistency in parallel programming. *Parallel Computing*, 37(4):217–229, 2011.

[24] James Demmel and Hong Diep Nguyen. Parallel reproducible summation. *IEEE Transactions on Computers*, 64(7):2060–2070, 2015.

[25] Dylan Chapp, Travis Johnston, and Michela Taufer. On the need for reproducible numerical accuracy through intelligent runtime selection of reduction algorithms at the extreme scale. In *2015 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 166–175. IEEE, 2015.

[26] Roman Iakymchuk, Sylvain Collange, David Defour, and Stef Graillat. ExBLAS: Reproducible and accurate BLAS library. In *NRE: Numerical Reproducibility at Exascale*, 2015.

[27] Dan Zuras, Mike Cowlishaw, Alex Aiken, Matthew Applegate, David Bailey, Steve Bass, Dileep Bhandarkar, Mahesh Bhat, David Bindel, Sylvie Boldo, et al. IEEE standard for floating-point arithmetic. *IEEE Std 754-2008*, pages 1–70, 2008.

[28] D. Nicholaeff, N. Davis, D. Trujillo, and R. W. Robey. Cell-based adaptive mesh refinement implemented with general purpose graphics processing units. Technical Report LA-UR-11-07127, Los Alamos National Laboratory, 2011.

[29] D. Nicholaeff, N. Davis, D. Trujillo, and R. W. Robey. CLAMR – cell-based adaptive mesh refinement mini-app. http://www.github.com/losalamos/CLAMR, 2011.

[30] Joe Schoonover. Spectral Element Libraries in Fortran (SELF). https://schoonovernumerics.github.io/SELF/. Accessed: 2017-07-07.

[31] D.S. Abdi, L.C. Wilcox, T.C. Warburton, and F.X. Giraldo. A GPU-accelerated continuous and discontinuous Galerkin non-hydrostatic atmospheric model. *Int. J. of High Perf. Comp. Appl.*, 2017.

[32] Amazon Web Services. http://aws.amazon.com. Accessed: 2017-07-07.

[33] Ewa Deelman, Gurmeet Singh, Miron Livny, Bruce Berriman, and John Good. The cost of doing science on the cloud: the montage example. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, page 50. IEEE Press, 2008.

[34] Peter Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE transactions on visualization and computer graphics*, 20(12):2674–2683, 2014.