Approximate Computing: Challenges And Opportunities

Ankur Agrawal, Jungwook Choi, Kailash Gopalakrishnan, Suyog Gupta, Ravi Nair, Jinwook Oh,
Daniel A. Prener, Sunil Shukla, Vijayalakshmi Srinivasan, Zehra Sura
IBM T.J. Watson Research Center, Yorktown Heights, NY 10598

***Abstract*—Approximate computing is gaining traction as a computing paradigm for data analytics and cognitive applications that aim to extract deep insight from vast quantities of data. In this paper, we demonstrate that multiple approximation techniques can be applied to applications in these domains and can be further combined together to compound their benefits. In assessing the potential of approximation in these applications, we took the liberty of changing multiple layers of the system stack: architecture, programming model, and algorithms. Across a set of applications spanning the domains of DSP, robotics, and machine learning, we show that hot loops in the applications can be perforated by an average of 50% with proportional reduction in execution time, while still producing acceptable quality of results. In addition, the width of the data used in the computation can be reduced to 10-16 bits from the currently common 32/64 bits with potential for significant performance and energy benefits. For parallel applications we reduced execution time by 50% using relaxed synchronization mechanisms. Finally, our results also demonstrate that benefits compounded when these techniques are applied concurrently.**

Our results across different applications demonstrate that approximate computing is a widely applicable paradigm with potential for compounded benefits from applying multiple techniques across the system stack. In order to exploit these benefits it is essential to re-think multiple layers of the system stack to embrace approximations ground-up and to design tightly integrated approximate accelerators. Doing so will enable moving the applications into a world in which the architecture, programming model, and even the algorithms used to implement the application are all fundamentally designed for approximate computing.

***Index Terms*—Approximate computing, perforation, reduced precision, relaxed synchronization**

I. Introduction

We are at the threshold of an explosion in new data, produced not only by large, powerful scientific and commercial computers, but also by the billions of low-power devices of various kinds. While traditional workloads including transactional and database processing continue to grow modestly, there is an explosion in the computational footprint of a range of applications that aim to extract deep insight from vast quantities of structured and unstructured data. There is an exactness implied by traditional computing that is not needed in the processing of most types of these data. Yet today, these cognitive applications continue to be executed on general-purpose (and accelerator) platforms that are highly precise and designed with reliability from the ground up. Approximate computing aims to relax these constraints with the goal of obtaining significant gains in computational throughput - while still maintaining an acceptable quality of results.

A primary goal of research in approximate computing is to determine what degrees of approximations in the several layers of the system stack (from algorithms down to circuits and semi-conductor devices) are feasible so that the produced results are acceptable, albeit different from those obtained using precise computation. Approximate computing techniques studied by various researchers have focused primarily on optimizing one layer of the system stack and have shown benefits in power or execution time. In this work we set out to investigate if combining multiple approximation techniques spanning more than one layer of the system stack compounded the benefits, and if these compounded benefits are widely applicable across different application domains.

In order to provide a concrete demonstration, we focused on three approximation categories: skipping computations, approximation of arithmetic computations themselves, and approximation of communication between computational elements. As representatives of each category we evaluated loop perforation, reduced arithmetic precision, and relaxation of synchronization. We selected applications that are computationally expensive but have the potential to significantly impact our lives if they became cheap and pervasive. Our applications spanned the domains of digital signal processing, robotics, and machine learning.

Across the set of applications studied, our results show that we were able to perforate hot loops in the studied applications by an average of 50%, with proportional reduction in overall execution time, while still producing acceptable quality of results. In addition, we were able to reduce the width of the data used in the computation to 10-16 bits from the currently common 32 or even 64 bits, with potential for significant performance and energy benefits. In the parallel applications we studied, we were able to reduce execution time by 50% through partial elimination of synchronization overheads.

Finally, our results also demonstrate that the benefits from these techniques are compounded when applied concurrently. That is, combined wisely, the multiple techniques do not significantly lessen the effectiveness of one another. As the benefits of approximate computing are not restricted to a small class of applications these results motivate a re-thinking of the general purpose processor architecture to natively support different kinds of approximation to better realize the potential of approximate computing.

The rest of the paper is organized as follows: Relevant prior work in approximate computing is summarized in Section II. Section III describes the evaluation methodology and the applications used in this study. We present the results in Section IV and conclusions in Section V.

II. Related Work

We summarize the relevant prior work in approximate computing especially focusing on loop perforation, reduced precision computation and relaxed synchronization

A. Loop Perforation

Given a loop with loop variable i that takes discrete values from a finite ordered set S, we define loop perforation as an approximate computing technique that performs the execution of the loop body for $i \in S'$ such that $S' \subset S$. Next, we define a function $P(i)$ for all $i \in S$, such that $P(i)$ denotes the probability that the ith loop iteration is selected for execution. The set S' may be constructed through deterministic or random sampling from S. The loop perforation factor can then be defined as:

$$\eta = 1 - \frac{\sum_{i \in S} P(i)}{|S|}$$

It is often convenient to think about the degree of perforation, $perf = \frac{1}{(1-\eta)}$.

There has been a fair amount of work done on some aspects of loop perforation ([1]–[5]). The focus of these has been on identifying opportunities, developing certain patterns as exemplars, providing compiler support, and studying the accuracy trade-offs for perforation.

B. Reduced Precision Computation

Reduced precision (RP) computation for approximate computing is a technique that represents variables and data structures in a program with fewer bits (compared with conventional integer and floating point numbers). This allows us to utilize less expensive and more energy-efficient hardware to perform the reduced precision computation using ALUs. The area and power of ALUs roughly scale quadratically with the bit width, and therefore, exploiting RP hardware enables packing significantly more ALUs within the same area/power envelope. These benefits are especially noticeable in accelerator architectures, such as GPUs and coarse-grained reconfigurable architectures, where a significant fraction of the area is occupied by these ALUs.

Prior work in exploiting various facets of reduced precision includes application-level analysis and exploitation ([6], [7]), hardware architectures and tradeoffs ([8]–[10]), and language and compiler support ([11]).

C. Relaxed Synchronization

Synchronization overhead is a major bottleneck in scaling parallel applications to a large number of cores. Synchronization, used especially to ensure that threads reach various points in their execution in a predictable manner or to ensure that all threads see consistent values when shared variables are updated, can be systematically relaxed to gain significant performance improvement.

Traditional synchronization techniques such as data privatization ([12], [13]), lock-free synchronization ([12]–[14]) and transactional systems ([15]–[17]) use careful analysis and identification of situations where synchronization is not needed, or use hardware or software structures that are able to detect and correct incorrect behavior due to relaxed synchronization. In [18] we examined the effect of omitting synchronization in situations where such an omission would not lead to catastrophic failure of the system.

III. Evaluation Methodology

Our goal in this work is to move applications into a world in which the algorithms, architecture, and programming model are all fundamentally designed for approximate computing. To do so, we took the liberty of assuming there are no constraints to changing multiple layers of the system stack, including the system and processor architecture, the programming model, and even the algorithm used to implement the application. From such a clean-sheet study we derive an estimate of the compounded potential gains from applying a set of approximate computing techniques concurrently.

We studied applications spanning different domains including image processing, machine learning, deep learning, and robotics. In this paper we present results for the image processing applications: Synthetic Aperture Radar (SAR) and Wide Area Motion Imagery (WAMI), from the PERFECT suite [19], K-means Clustering [20], Deep Neural Networks [21], and Robot Localization [22]. To assess the performance benefits from approximate computing techniques, it is necessary to determine whether the results produced are acceptable. To do so, it is important to understand the quality metrics typically used for these applications, and compare the observed quality with and without using approximate computing techniques.

We relied on a deep understanding of the application domain and usage context to derive the quality metrics, and to control and manage the different approximation techniques. When applying multiple approximate computing techniques simultaneously, we tuned the degree of approximation for each of the techniques to meet the overall quality requirements.

For loop perforation, we studied various flavors including randomized and periodic(strided) perforation. The use of randomized perforation, as contrasted with periodic perforation, has been mentioned in [4], but has not really been pursued. That is explained by the fact that most of the work on loop perforation has been confined to software techniques, lacking any specific hardware support. In our clean-sheet approach, we assume that such hardware support is possible, and study the potential benefits of randomized perforation as well.

In the parallel applications we studied, we omitted relaxing synchronizations that are used to ensure that fundamental data structures, e.g., linked lists, do not break due to simultaneous manipulation of their structure by different threads. We carefully chose only those other synchronizations that are primarily used to ensure that all threads see consistent values when shared variables are updated. For both loop perforation and relaxed synchronization we estimated the benefits based on the observed reduction in the overall execution time of the application and the corresponding trade-offs in the quality of results.

Although there is programming language and hardware support for single precision and double precision floating-point representations, support for custom precision floating-point representation is not straightforward. We used an in-house C++ template class that supports floating-point representation with custom bit-widths for both exponent and mantissa, and custom bit-width integer representation. The template class also supports different rounding modes, including support for a stochastic rounding [23]. We denote the reduced precision representation using $\langle M, E\rangle$ where M is the number of bits in the mantissa and E is the number of bits in the exponent.

Since the underlying hardware does not natively support a non-standard floating-point representation, these custom representations are emulated using the single precision floating-point representation in the template class. Emulation, in general, is several orders of magnitude slower than a design with native hardware support. Hence, to assess the benefits of reduced precision, we focused on the quality of results, and not on the improvement in execution time or reduction in power.

A. Applications

We now present a brief description of each application along with the quality metrics used to determine acceptability of the result.

1) Synthetic Aperture Radar (SAR): Synthetic aperture radar (SAR) [19] is a radar-based imaging modality used to produce high-resolution imagery from an airborne platform. The output of the SAR application is typically a 2-dimensional radar image where regions of high intensity/amplitude correspond to the objects to be identified. Spatial resolution which corresponds to the distance in the azimuth and range directions where the intensity of the signal is $-3dB$ (i.e. $1/2$ the power) of the peak intensity of a target is used as the quality metric as it is a measure of the ability to differentiate proximate targets with similar central-peak intensities.

2) Wide Area Motion Imagery (WAMI): The Wide Area Motion Imagery (WAMI) [19] systems provide a continuous view of expansive areas ($>$ 50 sq. miles) and are used for pattern detection, terrain analysis, and object tracking in a large area. Typically, *precision*, *recall*, and *f-measure* are used to measure the quality of the results. *Precision* is the fraction of retrieved foreground pixels that are relevant and *recall* is the fraction of relevant foreground pixels that are retrieved. *f-measure* is the harmonic mean of *precision* and *recall*.

$$\begin{aligned} precision &= \tfrac{TP}{TP+FP} \\ recall &= \tfrac{TP}{TN+FN} \end{aligned}$$

where TP, FP, TN, and FN refers to the number of true positives, false positives, true negatives, and false negatives, respectively.

3) K-means Clustering: K-means clustering [20] is an important application in signal processing as well as data analytics. An iterative algorithm is used to compute the partitioning of input points into k clusters, for each k=4...13. The iterative partitioning terminates when the number of points moving between clusters is less than a threshold *delta* (set to 0.1% by default) or when the maximum number of iterations has been reached (500 iterations). To estimate the quality of the clusters we used *sum_distance* which computes the sum of the Euclidean distance of each point to its cluster center.

4) Deep Neural Networks (DNN): Among various machine learning techniques, deep learning [21] has emerged as a powerful and versatile class of methods, which have dramatically improved the state of the art in application domains such as computer vision, speech recognition, and natural language classification. In particular, deep neural networks with convolutional layers have proven to be quite effective in image classification, where the input data has significant spatial correlations. Training deep networks involves large network models over a large labeled training dataset, and so these tasks tend to be computationally intensive. Typically, these networks use the back-propagation algorithm [24] to compute how their internal parameters should be fine-tuned in response to new training data.

There are two quality metrics associated with deep learning: (a) test error, and (b) training time. Typically, a portion of the labeled training data is set aside to measure the accuracy of the network after training is complete. The trained network performs classification on this test data subset, and test error is defined as the percentage of test data that is misclassified by the network. Training time measures how quickly a network can be trained. This is important since achieving good test errors usually involves large brute-force search over the space of hyper-parameters of the network. Running training experiments faster therefore equates to better-trained networks.

5) Robot Localization: Simultaneous Localization and Mapping (RGB-D SLAM): Simultaneous Localization and Mapping (SLAM) is an algorithm to recover the camera trajectory and the map from sensor data, in order to autonomously navigate a robot to its destination. One of the most popular SLAM algorithms is RGB-D SLAM [22], which builds an accurate and fast 3D map of surroundings with the help of a novel, low-cost RGB-D camera that provides depth information as well as RGB color information. Absolute trajectory error (ATE) is used as quality metric in the results. ATE measures the global consistency of the trajectory by calculating the maximum translational displacement of the observed trajectory and the ground truth.

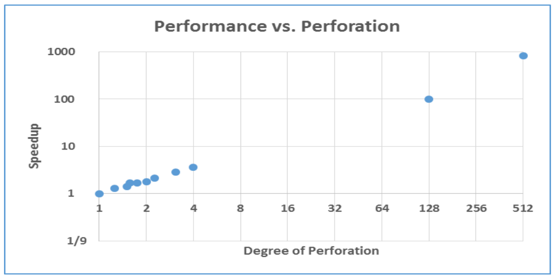

Figure 1. Execution Time Reduction vs. Degree of Perforation.

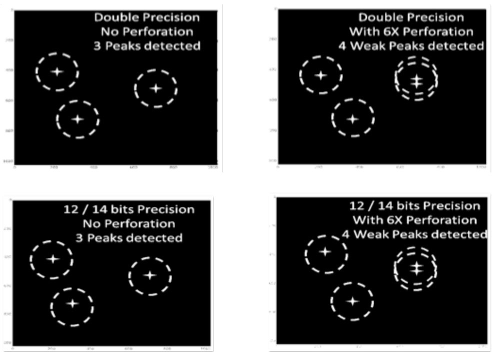

Figure 2. Location of spurious peak with perforation and reduced precision.

IV. Results

We now present individual and compounded performance benefits from loop perforation, reduced precision, and relaxed synchronization (when applicable) along with the quality of the results obtained.

A. SAR

The execution time of the SAR application is dominated by the backprojection kernel which integrates the contribution from each of the P pulses into each of $N_x \times N_y$ pixels. All three dimensions, the $x-$, the $y-$ and the pulse-directions, were chosen for exploitation of perforation. In the $x-$, the $y-$ directions both strided and random perforations were equally beneficial. However, in the pulse direction another form of perforation, the front-loaded perforation was found to be quite effective. In this form of perforation, only the first $(P/perf)$ iterations were performed, and the rest simply dropped.

Fig. 1 shows the reduction in execution time as a function of the degree of perforation. As shown, the total time decreases by a factor even greater than the degree of perforation. We observed that the degradation in the spatial resolution was proportional to the degree of perforation.

In cases where the area of the image of interest is small, a useful technique would be to first perform an approximation, e.g. using perforation, over the whole image, locate the area of interest and then zoom in to get an image in the locality of the area of interest. We observed that such an *approximate-then-validate* strategy leads to significant savings (up to $85\times$) in execution time.

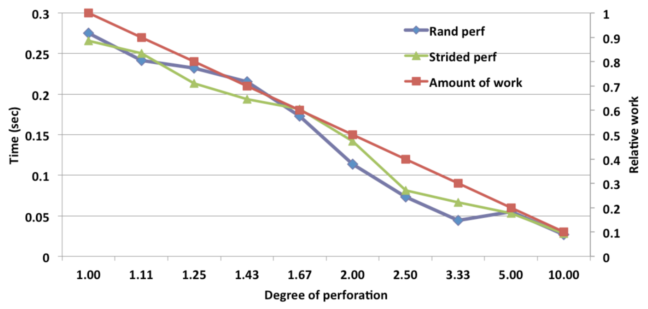

Figure 3. Exectuion time of Hessian kernel vs. Degree of Perforation.

When combined with reduced precision computation, we did not observe any additional degradation of the quality of the results. We varied the degree of perforation from $1\times$ to $8\times$, for a number of precision options ranging from double precision down to 14 bits of floating point representation. For every chosen precision, increasing the degree of perforation increased the number of spurious targets. The output of the backprojection kernel is a matrix of complex values of dimension $N_x \times N_y$. Fig. 2 shows a plot of the location of the spurious peaks. There are three easily visible peaks, and with perforation we observe spurious peaks. However all of the spurious peaks are in close proximity to the originally intended peaks – which therefore would not increase the computational burden under an *approximate-then-validate* strategy. Our results show that reducing the precision did not significantly impact how much perforation is permissible indicating that for SAR, the combined benefits of precision and perforation can indeed be fully exploited.

B. WAMI

Lucas Kanade algorithm, one the key functions of WAMI, constitutes nearly 96% of the execution time. The Lucas-Kanade algorithm aligns a template image to an input image by iteratively performing a series of steps, out of which the Hessian matrix computation consumes 73% of the total execution time. This kernel consists of a quadruply-nested loop, where the outer two loops correspond to the 6×6 Hessian matrix and the inner two loops ($x-$, and $y-$loops respectively) correspond to the number of pixels in the input image (2044x2044). We perforated the $x-$ loop by varying the degree of perforation from 1.0 to 10.0. Fig. 3 shows that both strided and random perforations achieve reduction in execution time proportional to the degree of perforation.

Fig. 4 shows that the quality metrics remain within 2% of the desired value of 1.0, for both random and strided techniques for degree of perforation less than 2.5. After that the quality starts to deteriorate rapidly falling more than 40% as the degree of perforation is increased to 10.0 for both techniques.

We studied a number of precision options, and concluded that with 7 bits of exponents and 8 bits of mantissa (a total of 16 bits for data representation, including the sign

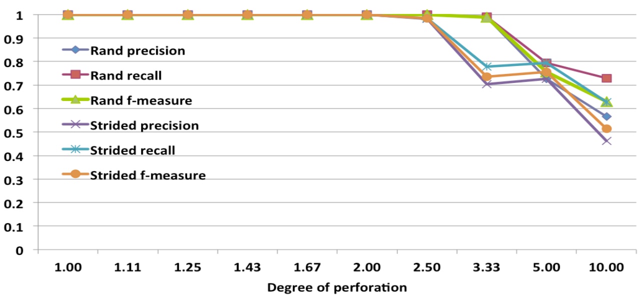

Figure 4. Quality of Results vs. Degree of Perforation.

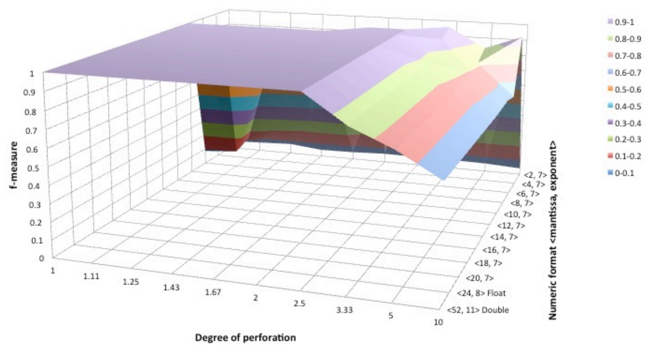

Figure 5. Quality of results with perforation and reduced precision.

bit) little or no degradation of quality was observed. Fig. 5 shows a 3D surface plot of *f-measure* (Y-axis), degree of perforation (X-axis) and numeric precision (Z-axis). On the Z-axis at degree of perforation of 1.0, reducing the precision from $\langle 52, 11\rangle$ (double) to $\langle 8, 7\rangle$ the *f-measure* remains within 0.8% of the desired value of 1.0. However, after $\langle 8, 7\rangle$ the *f-measure* suddenly drops to 0.086. As the degree of perforation is increased, *f-measure* starts to decline. It stays within 20% of the acceptable limit until a degree of perforation of 3.33 for all precision formats between $\langle 52, 11\rangle$ and $\langle 8, 7\rangle$. Below that the results become unpredictable because at lower precision the Hessian matrix consists of very small values, resulting in unpredictable results from the convergence algorithm.

In summary, with nearly no degradation in the quality of results, loop perforation achieves a 6× reduction in the execution time of the Hessian matrix multiplication, and an overall 3x reduction for the entire WAMI application and reduced precision with 7 bits of exponents and 8 bits of mantissa does not degrade the quality of results.

C. K-means Clustering

We perforated the iterative partitioning algorithm and observed that up to perforation degree of 4, the *sum_distance* metric did not deteriorate. As shown in Fig. 6, up to a perforation factor of 0.5 the effect of skipping iterations does not impact the convergence rate and so the execution time decreases steadily. When the perforation factor increases above 0.5, the effect of skipping the iterations slows down the convergence rate, and, with this increase in the number

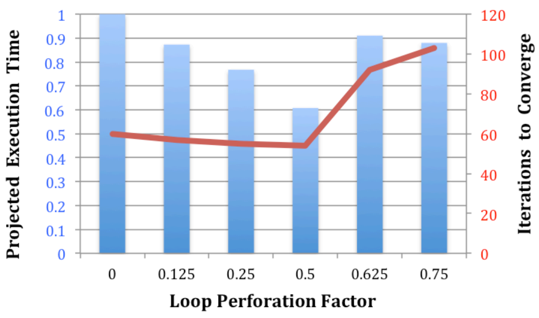

Figure 6. Execution time with perforation.

of iterations executed, the execution time improvement is reduced.

Based on experimenting with a variety of precision options, we concluded that using at least 6 bits of exponent and a total of 16-bits of floating point representation the *sum_distance* metric deteriorated by only 13%. Similar to the effect of perforation, using reduced precision computation slows down the convergence rate.

By increasing the number of parallel threads from 8, 16, 32, 64, to 96, we explored opportunities to relax synchronization in the main computation loop. In all cases, we observed a negligible change in the quality of results, with the *sum_distance* value varying less than 0.1% from the base case without approximation. Further, there was no change in the number of iterations required for the partitioning algorithm to converge when relaxing 1 in 5 iterations, regardless of the number of threads used. However, when synchronization was relaxed in every iteration and the number of threads used was 16 or greater, the partitioning loop executed for the maximum number of iterations and did not meet the convergence criterion of less than 0.1% of points moving between clusters. Our prior work [18] also shows that while the potential for significant speedup (up to 13×) exists, the speedup is limited when using more parallel threads.

To assess the compounded benefits from all the three approximation techniques we picked the best case configuration for each individual approximation technique, i.e. $\eta = 0.5$ for randomized loop perforation, $\langle mantissa, exponent\rangle = \langle 9, 6\rangle$ for reduced precision, and using 8 threads with relaxed synchronization. We ran experiments applying all pairwise combinations of these techniques, as well as applying all three techniques at the same time. We observed that adding relaxed synchronization to any other technique(s) had no impact on quality of results or number of iterations to converge. For the combination of loop perforation and reduced precision, the result was the same as the behavior of reduced precision, both in terms of quality of results and number of iterations to converge.

D. DNN

We present results of experiments performed on a network trained to classify images in one of ten categories using the

CIFAR-10 dataset. This dataset consists of 50000 labeled training images and 10000 test images. Each image belongs to one of the 10 classes, with 6000 images per class. The network is composed of 3 convolutional layers followed by one fully-connected soft-max layer. The baseline training experiments (single node, single-precision floating-point number representation and no perforation) yield a network that has about 18% test error. Training deep convolution networks is computationally demanding, with most of the computations concentrated in computing the 2D convolutions and matrix multiplications.

As there is significant redundancy in the convolution operation, we randomly skipped the computation of some of the locations during every convolution operation. For these points, we substituted data from the nearest valid computation. Randomly scrambling the maps for the convolution in each forward and backward convolution ensures that there is little degradation in accuracy over training performed with unperforated convolution layers. We do not perform perforations in the update step where we compute the new kernels, since we observed that it impacts the test errors of the trained network significantly.

We experimented with 16-, 12-, and 10-bit representation for floating-point numbers, where we fixed the exponent to be 6 bits, and used 9, 5, and 3 bits, respectively, to represent the mantissa. In the training experiments, only the matrix multiplication (GEMM) computations were performed in reduced precision. Since convolution operations are also cast as GEMM operations, this covers over 90% of all computations. We observed little to no degradation in the training error (cross-entropy) or test error when we reduce the precision to 12b.

When training a deep convolutional network in parallel over a cluster of computing nodes, a parameter server assumes the responsibility of collecting the weight updates from each of the learner nodes, and computing the weights for the learners to use in the next iteration of the stochastic gradient descent algorithm. In traditional distributed stochastic gradient descent, the parameter server waits for an update from each of the learning nodes before computing the next set of weights, commonly known as hard synchronization (hardsync). We relax the policy at the parameter server: instead of waiting from an update from each learner, the parameter server waits for n updates if there are n learner nodes, without requiring that each of the n updates came from distinct learners. If different learners are processing mini-batches with different speeds, this relaxed policy allows the faster learners to have higher throughput since they no longer have to wait for the slow learners to complete their computations. We call this policy "soft-sync" [25].

Fig. 7 shows the benefits of performing perforations in convolution for two cases: (a) Hardsync protocol at the parameter server, and (b) Softsync protocol at the parameter server. For 50% perforation, with Hardsync, we obtain about 1.5× speed-up, while Softsync yields 1.32× speedup. We observe less than the ideal speed-up of 2x for 50% perforations, suggesting that there are fixed overheads in performing the perforations.

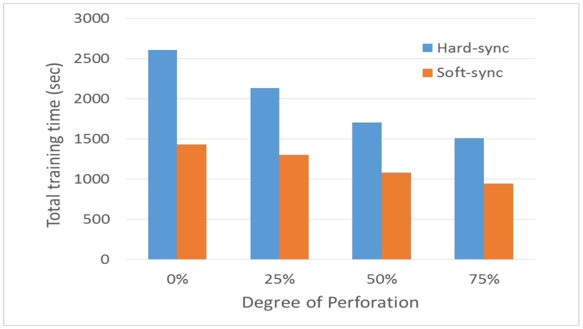

Figure 7. Training time with perforation and relaxed synchronization.

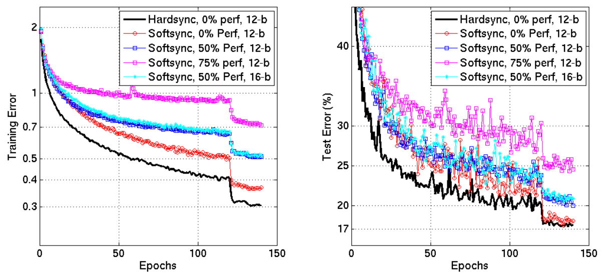

Figure 8. Training and Test Error for the CIFAR10 with varying degree of perforation in the convolution layers.

Fig. 8 shows the training and test error curves for experiments where we combine all three forms of approximations. As baseline we plot curves for the experiment with 0% perforation, Hardsync and 12b computations. When we modify the synchronization protocol to Softsync (red- curves), we observe a slight degradation in test error. As the degree of perforation is now increased to 50% and 75% (blue and magenta curves), we see that test errors degrade to about 20% and 25%, respectively. Finally, we plot the cyan curve that plots results for the 16b experiment with 50% perforation and Softsync. We observe that there is no improvement in test error with the slightly higher precision. This suggests that the quality of the trained network does not change much as the precision is reduced, before abruptly deteriorating at a certain threshold. The abrupt reduction in the test error occurs due to a reduction in the learning rate (step size) used during Stochastic Gradient Descent (SGD). A typical prescription of SGD mandates the learning rate to reduce as the learning proceeds [26]. In these experiments, we have adopted the step-learning rate schedule in which the learning rate is held constant for a certain number of epochs, followed by a step reduction.

Overall, the degradation in test error is dominated by the non-idealities introduced by the perforation. Relaxing synchronization and reducing the precision of computation have limited impact of the quality of the trained network.

E. RGB-D SLAM

The trajectory estimation of RGB-D SLAM can be divided into Frontend and Backend. Frontend builds spatial relations

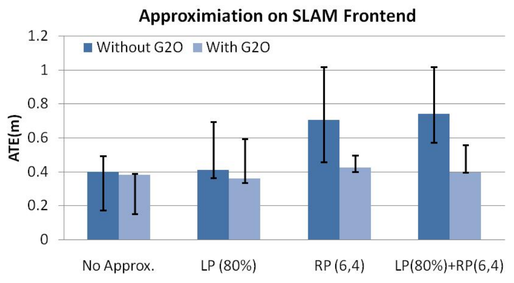

Figure 9. ATE variation with redureduced precision and perforation in Frontend.

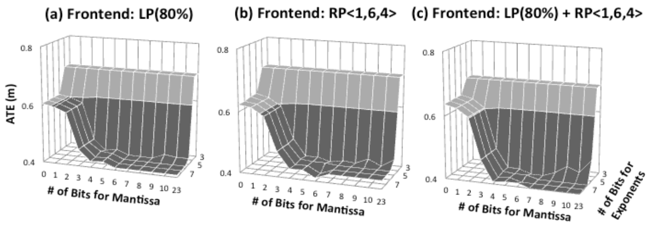

Figure 10. ATE impact with reduced precision in G2O and frontEnd with (a) 80% loop perforation, (b) reduced precision ¡1,6,4¿, (c) both perforation (80%) and reduced precision ¡1,6,4¿.

based on the extracted features of the individual observations between the input RGB-D frames. Backend optimizes the poses of these observations with wider frame windows built as a pose graph.

Feature detection and extraction consumes more than 75% of Fronend execution time which is dominated by Gaussian filtering implemented with 2D convolution. Gaussian 2D convolution in SIFT [27] feature detector/extractor requires at most 40 convolution operations (using five blur levels and eight octaves) for every input image, and is well-suited for loop perforation. We also experimented with a variety of reduced precision computations for the 2D convolution.

Backend constitutes global pose optimization functions using G2O [28] which iteratively construct and solve a linear system to find the robot trajectory given the frame-to-frame correspondence information from Frontend. The execution time of Backend is dominated by preconditioned conjugate gradient (PCG) function used for solving linear systems. As the solution of PCG is iteratively refined, numerical errors caused by approximation such as reduced precision can be repaired over the iterations. Thus, we experimented with a variety of reduced precision representations in all the vector/matrix operation in PCG.

We present results for one of the representative RGB-D benchmarks (fr2-pioneers-slams3) [29]. Fig. 9 shows the ATE variations when Frontend functions were approximated using loop perforation (80%) and reduced precision computations with 6 bits for mantissa and 4 bits for exponent. The error bars show the minimum and maximum ATE observed across a set of runs. Although loop perforation and reduced precision increase ATE variation significantly, we observe that when G2O is used in Backend both the ATE variation and the average ATE decreased significantly, thereby, providing more robust result quality. In particular, the reduced precision is quite disruptive for the case without G2O, but with G2O, the ATE variation is decreased by a factor of 1.73. The importance of global pose optimization with G2O becomes more critical in the presence of approximation on the SLAM Frontend.

We present the results using reduced precision computations in the G2O while using approximate computing techniques in Frontend as well. We used reduced precision of with 6 bits for mantissa and 4 bits for exponent and/or 80% loop perforation in Frontend, and used its estimated trajectory and the frame-to-frame correspondence information as input for G2O with various reduced-precision representations. As shown in Figure 10, for all the cases of the approximated Frontend, G2O demonstrates remarkable tolerance to reduced precision even down to 4 bits of mantissa and 6 bits of exponent.

F. Summary

Across different applications our results show that, without incurring loss of quality, the precision of the critical variables within hot functions of the program can be reduced to 16 bits (and in some cases even down to 10 bits). Furthermore, our results demonstrate that hot loops in the studied applications can be perforated by an average of 50%, with proportional reduction in overall execution time. In the parallel applications we studied we were able to reduce execution time by 50% using relaxed synchronization mechanisms. Finally, our results also demonstrate that the benefits from these techniques are compounded when applied concurrently.

V. Conclusions

Approximate computing techniques studied by various researchers have focused primarily on optimizing one layer of the system stack and have shown benefits in power or execution time. A fundamental goal of this work was to study if combining multiple approximation techniques spanning more than one layer of the system stack compounded the benefits, and if these compounded benefits are widely applicable across different application domains. We focused on three approximation categories: skipping computations, approximation of arithmetic computations themselves, and approximation of communication between computational elements. As representatives of each category we evaluated loop perforation, reduced precision computation, and relaxation of synchronization. We selected applications spanning the domains of digital signal processing, robotics, and machine learning.

Across the set of applications studied, our results show that, without incurring loss of quality, we were able to perforate hot loops in the studied applications by an average of 50%, with proportional reduction in overall execution time. In addition, we were able to reduce the width of the data used in the computation to 10 to 16 bits from the currently common 32 or even 64 bits, thus enabling significant performance and energy benefits. In the parallel applications we studied we were able to reduce execution time by 50% using relaxed synchronization mechanisms. More importantly, our results demonstrate that the benefits from these techniques are compounded when applied concurrently. Based on this effort, we infer that dramatic improvements in throughput and/or energy efficiency requires exploring solutions spanning the entire system stack starting from novel algorithms, hardware architectures, programming model all the way to approximate circuits and devices designed with approximate computing in mind.

Our results across different applications demonstrate that approximate computing is a widely applicable paradigm with potential for compounded benefits from applying multiple techniques across the system stack. In order to exploit these benefits it is essential to re-think multiple layers of the system stack to embrace approximations ground-up and to design tightly integrated approximate accelerators. For instance, the success of applying reduced precision computations in these application domains lays the groundwork for new accelerator microarchitectures that could be designed from the ground up with reduced precision units. Such architectures could provide significant improvements in throughput, area and power for image processing and machine learning applications over the 64-bit CPUs and GPUs that are used today. As an example, such a microarchitecture could incorporate an array of 16-bit (or smaller) reduced precision units in a SIMD or a dataflow fashion. Due to the quadratic dependency of the area and power of these reduced precision units on bit-width, such an accelerator could provide much higher throughput for machine-learning and image-processing applications. In addition, the accelerator could have architecture and hardware support for different flavors of loop perforation, programming model support for relaxed synchronization, and relaxed cache coherence and memory consistency models.

While the results of our experiments are extremely promising, there is still work to be done in delivering on the promise of approximate computing. For example, in this work, we manually estimated the trade-offs in the quality when applying multiple approximate computing techniques simultaneously, and tuned the degree of approximation for each of the techniques to meet the quality requirements. Further, we relied on a deep understanding of the application domain and usage context to derive the quality metrics, and to control and manage the different approximation techniques. In some respects, this is not different from what is needed in tuning any system to utilize the available resources efficiently. Nevertheless, the adoption of this paradigm could be accelerated by the development of systematic methods to reason about the scope, control, validation, and management of approximate techniques.

Acknowledgment

This work was funded in part by DARPA (FA8750-15-C-0125).

References

[1] H. Hoffmann *et al.*, "Using code perforation to improve performance, reduce energy consumption, and respond to failures," *MIT-CSAIL-TR-2009-042*, 2009.

[2] M. Rinard, H. Hoffmann, S. Misailovic, and S. Sidiroglou, "Patterns and statistical analysis for understanding reduced resource computing," in *ACM Sigplan Notices*, vol. 45, no. 10. ACM, 2010, pp. 806–821.

[3] H. Hoffmann *et al.*, "Dynamic knobs for responsive power-aware computing," in *ACM SIGPLAN Notices*, vol. 46, no. 3. ACM, 2011, pp. 199–212.

[4] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. accuracy trade-offs with loop perforation," in *FSE*. ACM, 2011, pp. 124–134.

[5] S. Misailovic, D. M. Roy, and M. C. Rinard, "Probabilistically accurate program transformations," in *Static Analysis*. Springer, 2011, pp. 316–333.

[6] A. W. Brown, P. H. Kelly, and W. Luk, "Profiling floating point value ranges for reconfigurable implementation," in *Proc. 1st HiPEAC Workshop on Reconfigurable Computing*, 2007, pp. 6–16.

[7] D. H. Bailey, "Resolving numerical anomalies in scientific computation." [Online]. Available: http://www.davidhbailey.com/dhbpapers/numerical-bugs.pdf

[8] J. Y. F. Tong, D. Nagle, and R. A. Rutenbar, "Reducing power by optimizing the necessary precision/range of floating-point arithmetic," *IEEE Trans. VLSI Syst.*, vol. 8, no. 3, pp. 273–286, 2000.

[9] S. R. Amant *et al.*, "General-purpose code acceleration with limited-precision analog computation," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 505–516, 2014.

[10] D. F. Zucker and R. B. Lee, "Reuse of high precision arithmetic hardware to perform multiple concurrent low precision calculations," *CSL-TR-94-616*, 1994.

[11] A. Sampson *et al.*, "Enerj: Approximate data types for safe and general low-power computation," in *ACM SIGPLAN Notices*, vol. 46, no. 6. ACM, 2011, pp. 164–174.

[12] F. Allen *et al.*, "A framework for determining useful parallelism," in *ICS*. ACM, 1988, pp. 207–215.

[13] Z. Li, "Array privatization for parallel execution of loops," in *ICS*. ACM, 1992, pp. 313–322.

[14] L. Lamport, "Concurrent reading and writing," *Communications of the ACM*, vol. 20, no. 11, pp. 806–811, 1977.

[15] M. Herlihy and J. E. B. Moss, "Transactional memory: Architectural support for lock-free data structures," *SIGARCH Comput. Archit. News*, vol. 21, no. 2, pp. 289–300, May 1993.

[16] J. Alemany and E. W. Felten, "Performance issues in non-blocking synchronization on shared-memory multiprocessors," in *PODC*. ACM, 1992, pp. 125–134.

[17] N. Shavit and D. Touitou, "Software transactional memory," *Distributed Computing*, vol. 10, no. 2, pp. 99–116, 1997.

[18] L. Renganarayana, V. Srinivasan, R. Nair, and D. Prener, "Programming with relaxed synchronization," in *RACES*. ACM, 2012, pp. 41–50.

[19] K. Barker *et al.*, "Perfect (power efficiency revolution for embedded computing technologies) benchmark suite manual," *Pacific Northwest National Laboratory and Georgia Tech Research Institute*, 2013.

[20] Nu-minebench. [Online]. Available: http://cucis.ece.northwestern.edu/projects/DMS/MineBench.html

[21] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[22] F. Endres *et al.*, "3-d mapping with an rgb-d camera," *IEEE Trans. Robot.*, vol. 30, no. 1, pp. 177–187, 2014.

[23] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," *arXiv preprint arXiv:1502.02551*, 2015.

[24] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[25] W. Zhang, S. Gupta, X. Lian, and J. Liu, "Staleness-aware async-sgd for distributed deep learning," *arXiv preprint arXiv:1511.05950*, 2015.

[26] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[27] D. G. Lowe, "Object recognition from local scale-invariant features," in *ICCV*, vol. 2. Ieee, 1999, pp. 1150–1157.

[28] R. Kümmerle *et al.*, "g2o: A general framework for graph optimization," in *ICRA*. IEEE, 2011, pp. 3607–3613.

[29] J. Sturm *et al.*, "A benchmark for the evaluation of rgb-d slam systems," in *IROS*. IEEE, 2012, pp. 573–580.