Approximate Communication: Techniques for Reducing Communication Bottlenecks in Large-Scale Parallel Systems

FILIPE BETZEL, KAREN KHATAMIFARD, HARINI SURESH, DAVID J. LILJA, JOHN SARTORI, and ULYA KARPUZCU, University of Minnesota

Approximate computing has gained research attention recently as a way to increase energy efficiency and/or performance by exploiting some applications' intrinsic error resiliency. However, little attention has been given to its potential for tackling the communication bottleneck that remains one of the looming challenges to be tackled for efficient parallelism. This article explores the potential benefits of approximate computing for communication reduction by surveying three promising techniques for approximate communication: compression, relaxed synchronization, and value prediction. The techniques are compared based on an evaluation framework composed of communication cost reduction, performance, energy reduction, applicability, overheads, and output degradation. Comparison results demonstrate that lossy link compression and approximate value prediction show great promise for reducing the communication bottleneck in bandwidth-constrained applications. Meanwhile, relaxed synchronization is found to provide large speedups for select error-tolerant applications, but suffers from limited general applicability and unreliable output degradation guarantees. Finally, this article concludes with several suggestions for future research on approximate communication techniques.

CCS Concepts: • **Computer systems organization → Multicore architectures**; *Interconnection architectures*; *Multiple instruction, multiple data*; • **Computing methodologies** → Massively parallel algorithms;

Additional Key Words and Phrases: Approximate communication, approximate computing, communication reduction, scalability

ACM Reference format:
Filipe Betzel, Karen Khatamifard, Harini Suresh, David J. Lilja, John Sartori, and Ulya Karpuzcu. 2018. Approximate Communication: Techniques for Reducing Communication Bottlenecks in Large-Scale Parallel Systems. *ACM Comput. Surv.* 51, 1, Article 1 (January 2018), 32 pages.
https://doi.org/10.1145/3145812

1 INTRODUCTION

As semiconductor technology scaling approaches fundamental physical limits, the performance and energy efficiency of single-processor computing systems has reached a plateau. In light of diminishing marginal returns in processor efficiency, technological, economical, and practical considerations have converged to dictate that improved performance and energy efficiency in computing systems will increasingly be sought through parallelism. As the traditional hardware–software stack is primarily built around single-processor systems, the nearly universal shift to

Authors' addresses: F. Betzel, K. Khatamifard, H. Suresh, D. J. Lilja, J. Sartori, and U. Karpuzcu, University of Minnesota, Department Of Electrical and Computer Engineering, 200 Union St SE, Minneapolis, MN 55455; emails: {betze006, khata006, sures013, lilja, jsartori, ukarpuzc}@umn.edu.

https://doi.org/10.1145/3145812

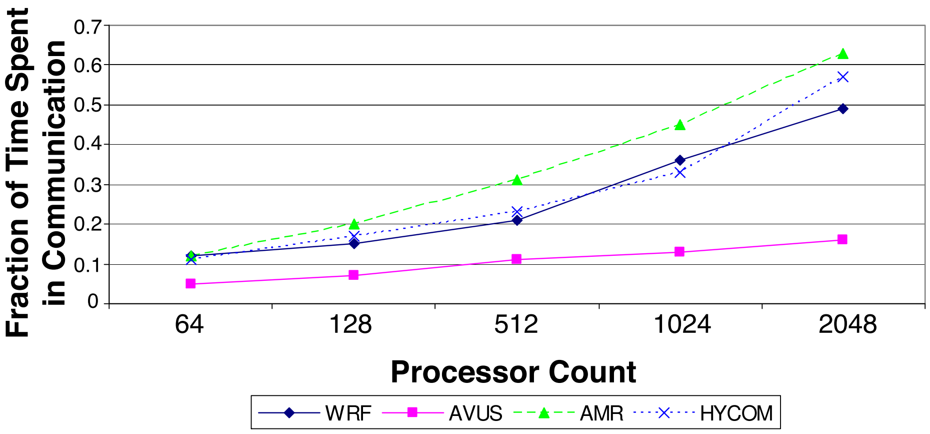

Fig. 1. Increase in communication overhead with processor count for representative exascale applications: WRF (weather prediction), AVUS (computational fluid dynamics), AMR (adaptive mesh refinement), HYCOM (ocean modeling) [17].

multiprocessor systems means that we may want to rethink, and subsequently redesign, several elements in the traditional hardware–software stack.

As we increasingly seek to improve performance and energy efficiency through exploiting parallelism and scalability, one of the looming challenges that remains to be tackled is that scalable and efficient parallelism appears to be incompatible with any nonnegligible amount of communication or synchronization between processing elements. Even for modest parallel scaling (e.g., on the order of tens of cores), all but the most embarrassingly parallel applications achieve only meager improvements in performance and energy efficiency [17]. This is somewhat alarming, given that current petascale machines house on the order of tens of thousands of processors and tens of millions of cores. Indeed, highly parallel machines only reach their peak performance ratings when running benchmarks with extremely high computation-to-communication ratios [44]. The push toward exascale computing will stress parallelism bottlenecks, such as communication and synchronization, even further.

For example, a recent study on exascale computing [17] provides results showing the fraction of time spent in communication for several parallel applications as the number of processing elements varies (Figure 1). While the fraction of time spent in communication is approximately 10% or less for all applications when running on 64 or fewer processors, the fraction of time spent in communication balloons to consume the majority of runtime (over 50%) for most of the applications as the number of processors increases by a factor of 32. As dictated by Amdahl's law [9], even a small fraction of runtime spent in communication (e.g., 1%) can easily become the primary bottleneck for an application that targets extreme (e.g., 100-1000X) parallelism. Clearly, even a relatively small dependence on communication and synchronization can seriously inhibit efficiency as we seek greater levels of parallelism. However, many (perhaps most) real parallel applications require nontrivial amounts of communication and synchronization.

Communication is also costly in terms of power. It is estimated that interconnection networks consume 10% to 20% of the power in current High-Performance Computing (HPC) systems [177] and that the majority of this power is used in the network's links [178]. As such, the same exascale study mentioned above [17] conveyed "a real need for the development of architectures and matching programming systems solutions that focus on reducing communications, and thus communication power," admitting that "the toughest of the power problems is not … computation, but communication." Additionally, similar power constraints can also be found in the design of Networks-on-Chip (NoCs) [80], where even low-power designs are found to account for as much as one-third of a chip's power consumption [74]. Finally, as increased parallel integration stresses power and thermal budgets and we run up against the power wall [49, 155], every bit of power spent on communication is power that cannot be spent to perform computation.

Clearly, efficient exploitation of parallelism and scalability requires solutions that target both the latency and power of communication. Several works have proposed hardware and software techniques for improving the efficiency of communication and synchronization operations in large-scale multiprocessor systems [3, 7, 16, 27, 38, 54, 63, 78, 94, 109, 123, 147] and chip multiprocessors [136, 138, 142, 164, 183]. The most efficient of these techniques allow communication and synchronization costs to scale proportional to $log(cpu\ count)$. However, as demonstrated in [17], communication latencies must improve at a rate better than $log(cpu\ count)$ in order to achieve more than one order of magnitude performance improvement from scaling out one of today's machines by 1000X. Thus, even the *efficient* communication and synchronization techniques proposed in the past may not be suitable for exploiting parallelism and scalability in next-generation computing systems.

Approximate computing is a recent research topic that is growing in popularity among researchers and industry. It aims to increase power and/or performance by allowing computations to be performed "approximately." In other words, in approximate computing, inaccuracies are deliberately allowed or even introduced in order to reap efficiency gains. In a typical system, computer architects must balance the trade-offs between energy efficiency and performance under certain constraints, such as chip area and power. Approximate computing introduces a new trade-off between output quality and energy efficiency. As such, it allows the flexibility to introduce some acceptable degradation in the output in order to improve the efficiency of the system. The main motivation for approximate computing comes from the observation that many recent compute and data-intensive applications are fairly tolerant to small inaccuracies in the results. Recognition, Mining, and Synthesis (RMS) [47] applications are an example of computing tasks that do not aim to reach an exact numerical answer; rather, they work with models from real-world data that are intrinsically inaccurate. Applications in pattern-recognition, machine-learning, and physical-simulation domains are a few examples of such applications with inherent output error resilience. There are a variety of approaches to exploiting this application error resilience for approximate computing. Recent papers have explored the design of approximate circuits, architectural modifications, and approximate software, as shown in Table 1.

We define *approximate communication* as the application of approximate computing techniques to parallel systems with the goal of reducing the amount of communication between processing elements. Previous research [95, 96, 99] has shown that scalability in parallel applications can be improved significantly by relaxing the application's communication and synchronization patterns. With that in mind, this article presents a survey of three algorithmic and architecturally based techniques found in the literature that, if adapted to approximate communication, would have the potential to reduce communication overhead and enable improved scalability in future exascale systems. While the main focus is on parallel applications running on massively parallel architectures, some ideas can also be applied to reducing communication overhead on distributed systems and chip multiprocessors. We aim to answer the following set of questions:

(1) Can we apply the ideas of approximate computing to target the communication scalability challenge in massively parallel systems?

Table 1. Approximate Computing Techniques Classified in Terms of Their Potential for Communication Reduction

	Potential for Communication Reduction	Target Communication Cost	References
Software			
Loop Perforation	Indirect	Reduces N	[14, 64, 100, 149]
Task Dropping / Skipping	Indirect	Reduces N	[21, 26, 57, 95, 133, 158]
Memory Access Skipping	Direct	Reduces N	[133]
Data Sampling	Direct	Reduces Q	[13, 57]
Thread Fusion	Indirect	Reduces N	[133]
Pattern Reduction (Paraprox)	Direct	Reduces N and Q	[132]
Lossy Compression	Direct	Reduces Q	[133]
Precision Scaling	Direct	Reduces Q	[1, 11, 66, 129, 130, 157]
Relaxed Synchronization	Direct	Reduces Sync	[21, 96, 101, 126, 127]
Algorithm Selection	Indirect	Reduces N or Q	[12, 14]
Parameter Adjustment	Indirect	Reduces N or Q	[65]
Architecture			
Approximate Storage	None	N/A	[50, 125, 135, 172]
ISA Extensions	Indirect	Reduces N or Q	[50, 134]
Approximate Accelerators	Indirect	Reduces N or Q	[45, 104, 151, 180]
Neural Accelerators	Indirect	Reduces N or Q	[51, 59, 105, 161]
Approximate General Purpose Processors	None	N/A	[29–31, 46, 84, 160, 175, 176]
Stochastic Processors	None	N/A	[32, 108]
Approximate Value Prediction	Direct	Reduces N	[97, 154, 156]
Fuzzy Memoization	Direct	Reduces N	[8, 72]
Reducing Divergence in GPU	Direct	Reduces N	[59, 139]
Circuit			
Imprecise Logic	None	N/A	[25, 50, 98, 162, 174]
Voltage Overscaling	Indirect	Reduces Power	[50, 103, 122]
DRAM Refresh Rate Reduction	Indirect	Reduces Power	[33, 90, 134]
Analog Computation	None	N/A	[151]
HW Precision Scaling	None	N/A	[157]
Soft Fault Tolerance	None	N/A	[73, 114, 153]
Reduced Precision FPU	None	N/A	[179]
Approximate Adder	None	N/A	[60, 70]
Approximate Multiplier	None	N/A	[76]
Unreliable Memory	Indirect	Reduces Power	[53, 55, 71, 125, 148]

Note: N = Number of communication messages. Q = Size of communication messages.

(2) What are some existing approximate techniques that could be used for communication reduction?
(3) How do these techniques compare and how can they be applied to approximate communication?

This article is organized as follows. Section 2 has an overview of the existing research on three promising communication reduction techniques: compression, relaxed synchronization, and value prediction. Section 3 provides a comparative analysis of the techniques using a common evaluation framework. Then, based on the analysis results, Section 4 presents several suggestions for future research. Section 5 contains our conclusions.

2 AN OVERVIEW OF PROMISING TECHNIQUES FOR APPROXIMATE COMMUNICATION

Approximate communication is a subsection of the larger approximate computing field. Table 1 shows a list of current approximate computing techniques outlined in recent survey papers [102, 159, 171]. We classify the techniques in terms of their target computing stack layer (software, architecture, or circuit) and in terms of their potential to reduce communication overhead either directly or indirectly. For those techniques with communication reduction potential, we also classify them in terms of the main target communication cost that is likely to be reduced by the technique. The communication cost, as further explained in Section 3.1, is divided into two main components: the number of messages transmitted (N) and the size of each message (Q). We also identify those techniques that target synchronization and power overheads directly. The table shows that several current approximate computing techniques have the potential to improve communication efficiency, either directly or indirectly. In this article, however, we concentrate on those techniques that most directly target communication and synchronization overheads in parallel systems. For synchronization overheads, we chose relaxed synchronization as the most promising technique. For communication overhead, on the other hand, we chose load value prediction and lossy compression as the most promising techniques to target the number and size of communication messages, respectively. Other techniques in the list are likely to also yield significant communication efficiency improvements and are worth exploring in further research. However, by concentrating on only the selected techniques, this article is able to explore each technique in greater depth and present a comparative analysis of their potential, as found in Section 3.2. In this section, we present a survey on each technique individually. First, we provide a broader background on each technique and then discuss its benefits and challenges for approximate communication.

2.1 Compression

Compression is a commonly used technique to reduce bandwidth utilization and improve energy efficiency when transmitting data over a communication channel. The premise of data compression is the removal of redundant data, effectively increasing the density of data to be communicated. An approach to achieving compression of data is the representation of data using codes, which capture the information to be transmitted at a shorter length. Compression techniques can be divided into two categories based on the output fidelity: lossless and lossy compression. Lossless compression enables a faithful reconstruction of the data at the receiver. Lossy compression achieves better compression by compromising on the accuracy of the information [24]. Lelewer and Hirschberg [85] present a survey of data compression techniques, classifying them into domain-specific techniques and general-purpose techniques. Domain-specific techniques are coding techniques specific to the data of interest, whereas general-purpose techniques do not require a prior knowledge of the data to be compressed.

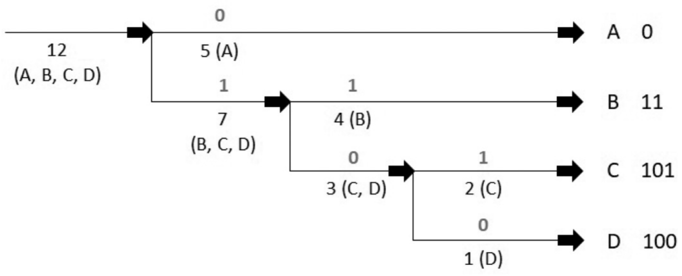

Fig. 2. Simple example of a Huffman encoding tree for the character string "AAAAABBBBCCD."

2.1.1 Lossless Compression. General-purpose lossless data compression techniques can be classified into statistical data compression and dictionary-based compression techniques. Statistical data compression methods make use of the probability of occurrence of the symbols to assign codes. Entropy encoding techniques, such as Huffman and Arithmetic encoding, assign shorter codes to frequently occurring symbols and longer codes to rarely occurring symbols. Entropy encoding is preceded by a modeling element to model the probability of occurrences of the symbols from the source. In the case of Huffman coding, the encoding can be represented as a Huffman tree. Figure 2 shows a Huffman tree encoding of four characters with different probabilities. In this example, the average bit length is reduced from two with a simple encoding (e.g., A = 00, B = 01, C = 10, D = 11) to 1.83 with Huffman coding. Huffman coding has found many applications, including commercial ones, such as JPEG image compression [165]. Another example of statistical data compression is Predictive Coding [48], which emerged in 1955 as a solution for efficient coding for systems that have an inherent predictability of data. In this compression method, the knowledge of the past symbols transmitted by the source is used to predict the current symbol. The coding schemes transmit the error between the original and the predicted symbol when the predicted symbol is within a threshold of the original symbol. With the same prediction made at the receiver based on the incoming stream and the error information in the encoded stream, the original symbol is recovered.

Dictionary-based compression [131] uses a dictionary that holds strings of symbols. They read strings of symbols from the input and try to match the strings with those present in the dictionary. If there is a match, the string is replaced with a reference to the dictionary, called a token, instead of a code for the symbol. Longer repeated strings provide a better compression ratio. If a match with the dictionary is not found, the uncompressed data are written. For better compression, the stream of tokens and uncompressed data may be further compressed using entropy encoders, such as Huffman or Arithmetic encoders. The Lempel Ziv Welch (LZW) [168] compression algorithm is good example of a simple and efficient implementation of dictionary-based compression found in commercial products, such as the GIF file format.

2.1.2 Lossy Compression. The working of lossy compression algorithms is based on the approximation of the input stream to remove redundant and irrelevant data. Lossy compression algorithms are domain specific, since the improvement in the compression over lossless algorithms arises due to their ability to weed out irrelevant data. Lossy compression methods are applicable only for applications in which perceptual inaccuracies in the output are permissible. Typical applications of lossy compression include image, video, and audio compression [77, 110, 116]. The loss of output fidelity is either imperceptible to the user or has been accepted by the user for the application and, hence, can be traded for a higher compression ratio. Lossy compression of scientific datasets has also been attempted from earth-observing system instruments, ocean science acoustic sonar data, synthetic aperture radar, and so on [58, 68, 92].

Determining the extent to which quality can be traded for improvement in compression ratio may be challenging, as the decision may often depend on user preferences and may differ based on the application. Different metrics exist for evaluating the output quality depending on the application: Peak Signal-to-Noise Ratio (PSNR), Structural Similarity (SSIM), Multiscale SSIM (MS-SSIM) and so on [86, 167]. Lossy compression algorithms allow fluid trade-offs between quality and compression ratio, enabling a user to select a desired point in the trade-off space depending on desired quality and/or compression. The difference between the input stream to be compressed and the reconstructed stream after decompression is called distortion. The goal of lossy compression is to maximize the compression, i.e., minimize the average number of bits to encode the input stream, within the given distortion limits.

Lossy compression algorithms may be categorized as either predictive coding or transform coding. Predictive coding provides a prediction of the current sample from the previous decoded sample. For example, in Linear Predictive Coding (LPC), a linear combination of past samples is used to approximate or *predict* the current time domain sample [43]. One of its primary uses is in digital audio compression, such as in the GSM Codecs [106]. Transform coding provides a better representation of the dataset in a different domain, which helps portray information as it will be perceived. This helps to identify information that can be quantized with little impact on perceived quality. Transforms such as Discrete Fourier Transform (DFT), Discrete Cosine Transform (DCT), and Discrete Wavelet Transform (DWT) [131] represent the source signal in terms of its spectral information, which can be approximated with minimal effect on output quality. In the domain of image compression, DFT and DCT represent the pixels of an image by the transform coefficients that correspond to different spatial frequencies. A segment of the image with small details and sharp edges contains high spatial frequencies. The principle behind quantization of the coefficients to obtain compression is that the human eye is more sensitive to variations in the lower-frequency regions than in the higher-frequency regions. DWT captures both spatial and temporal information. There has been significant research on introducing approximations to these transforms to reduce the complexity of implementation, thus trading accuracy for complexity [36, 113]. Integer approximations to Fourier Transforms have also been attempted; the approximated transformation has been shown to reconstruct the input perfectly [35]. A methodology to approximate Discrete Sinusoidal transforms resulting in integer coefficients has been applied to Discrete Fourier, Hartley, and Cosine transforms [82]. The benefits of the integer approximations to transforms are in the lower multiplicative complexity of the computations compared to traditional approaches.

2.1.3 Applications of Compression. One of the most common applications of compression is in reducing the storage requirements of data. File compression applications—such as Gzip, Bzip2, and LZMA—are well known and widely used for reducing the size of files in a file system. Memory compression [2] seeks to improve the effective system memory capacity by compressing the data in main memory. Compressor and decompressor hardware engines help limit the additional latency incurred from compression and decompression. It has also been shown [145] that memory compression not only improves storage capacity but also reduces memory link bandwidth utilization and power. Other research efforts have also investigated the design of cache compression [5, 61, 79] for reducing the miss rate in on-chip caches at the cost of an increase in hit latency.

2.1.4 Link Compression. Compression may also be applied on-the-fly on data that is to be transferred over a bandwidth-constrained network. Compression of data to be communicated practically increases the bandwidth available for communication [23, 37, 42] and can be optimized by exploitation of knowledge about the data. Link compression is commonly used in many fields, from telephony to video streaming. However, many computer networks have high variability in network bandwidth and congestion. If compression is online and makes use of a CPU, its utilization decides the efficiency of compression. Adaptive online compression techniques [75, 121, 170] allow compression to be selectively applied based on network and system usage as well as the type of data being transmitted. Since different compression algorithms have different computation overheads and compression ratios, adaptive online compression techniques also dynamically change the compression algorithm that is applied. This allows for the most efficient compression algorithm to be used given the current system conditions and predicted compression ratios.

Link compression has also been applied to parallel architectures in order to reduce the communication overheads in parallel systems. Bui et al. [20] showed how the parallel performance of the BlueGene/Q supercomputer can be improved with a combination of compression, topology-aware data aggregation, and subfiling. Their results showed that compression added an extra 40% performance improvement over data aggregation and subfiling alone. Welton et al. [169] presented a design of a data compression service that is implemented in the I/O forwarding layer of supercomputers, and Bicer et al. [18] demonstrated the use of a domain-specific lossy compression algorithm to reduce I/O time and improve performance of parallel scientific applications.

In addition to supercomputers, some studies [10, 41, 69, 112, 150] have investigated compressing the data traffic in an NoC. Just as in supercomputers, interconnect network traffic is becoming a major bottleneck and the main power drain in chip multiprocessors [15, 19, 40, 74, 80]. These studies show that it is possible to design efficient compression algorithms and architectures that reduce packet latency and energy, even in high-performance interconnects such as those found in NoCs. The studies also demonstrate that compression can be applied to a wide range of architectures to significantly reduce the communication overheads in parallel systems.

2.1.5 Compression and GPUs. Several research papers have also investigated the benefits of compression for graphics processing units (GPUs). Sathish et al. [140] investigated the use of link compression to improve the effective bandwidth between the GPU and its off-chip memory. Their research is based on the observation that "state-of-the-art graphic processing units (GPUs) provide very high memory bandwidth, but the performance of many general-purpose GPU (GPGPU) workloads is still bounded by memory bandwidth." They propose a new memory controller (MC) architecture capable of performing lossy and lossless compression on blocks of data written to main memory. Vijaykumar et al. [163] propose using *assist warps* that make use of idle GPU computing resources to alleviate the memory bandwidth bottleneck in memory-bound applications. The assist warps may be used to perform data compression on cached data before being written to main memory and decompression after reading from main memory. Thus, their framework is able to reduce the compute versus memory resource imbalances in memory-bound applications. It was shown to provide an average of 41.7% performance improvement on a set of bandwidth-sensitive GPU applications. Modern GPUs are, in fact, already making use of compression to increase the effective memory bandwidth. One example is the Delta Color Compression used by the GeForce GTX 980 GPU [111].

In addition to performance and bandwidth improvements, compression has also been shown to be useful in reducing GPU power consumption. For example, Warped-compression [83] compresses the register values within a warp, allowing fewer register file bank accesses and thus saving power. Pekhimenko et al. [117] proposes two toggle-aware compression techniques (Energy Control and Metadata Consolidation), which are shown to reduce the GPU's DRAM bus energy consumption by 8.3%, on average, despite the fact that in some cases GPU data compression may increase memory energy consumption by increasing the number of bit toggles on the interconnects.

2.1.6 Compression and Energy Consumption. Compression has a direct impact on the energy consumption of the system. While compression and decompression incur additional computation and thus additional energy consumption, it also decreases the amount of data that needs to be transmitted over the communication channel. If the energy required to perform compression and decompression is lower than the energy required to transmit the extra bits, then compression will save energy. Thus, analysis of energy consumption of compression is dependent on the compression ratio and the transmission link energy. One notable example in which compression is particularly effective in reducing energy consumption is in a wireless environment. The study performed in [82] on Intel i7 quad-core processors in a wireless environment supporting 11Mbps with 11W and wired environment with 15Mbps with 15W shows that sending uncompressed data can consume more energy than performing compression on a multicore processor. For example, JPEG and H.264 in a wireless environment consumes an average of 1/5th of the energy and JPEG2000 in wired environment consumes half of the energy compared to transmitting uncompressed data, on average.

The use of approximate hardware to trade off accuracy and power consumption has been investigated through the design of approximate circuit blocks. For example, the authors in [67] propose the design of imprecise adders by impacting the number of transistors in an adder block and their load capacitance. Using approximate addition for the least significant bits for the computations involved in DCT and inverse DCT blocks for image/audio/video compression reveals savings in power and area without an appreciable loss of quality.

2.1.7 Compression and Parallel Systems. The emergence and ubiquity of mobile computing, sensor networks, parallel computing, cloud computing, and so on have increased the stress on the communication bandwidth. While compression is fairly common in systems that transmit data over long distances (e.g., the Internet, telephony, and so forth) it may begin to see widespread use in large-scale datacenters and other parallel computing domains as systems move toward increasing levels of parallelism, and communication becomes increasingly expensive relative to computation. Even though there is a lot of existing research in the applications of compression, there is still a need for additional research in the application of compression to parallel systems. In particular, aggressive lossy compression algorithms could be applied to recognition, mining, and synthesis applications by exploiting these applications' resilience to data inaccuracies. Approximating communication in interconnect networks through lossy compression may provide improved scalability for these parallel applications.

2.1.8 Benefits and Challenges of Compression for Approximate Communication. Several papers have shown the benefits of link compression in reducing the amount of data that needs to be transferred over a communication channel. For example, in [6], link compression alone was shown to reduce bandwidth between 23% and 41%. Das et al. [41] showed that compression in the network interface controller of an NoC is able to reduce network latency by 20%, on average. Link compression is a technique that has been used for many years not only in computer networks but also in telephony and video broadcasting. Moreover, link compression techniques borrow from years of research on compression from industry and academia. Compression techniques have a long history of successful implementations in commercial products and a diverse set of ongoing research. Thus, compression is a technique that is well understood, widely used in practice, and trusted by system designers. Another benefit from compression is that it can be optimized for certain applications and implemented in either software or hardware. Thus, it provides ample design choices to fit any system and application.

While link compression alone may deliver improved bandwidth and latency, lossy link compression is likely to result in even greater benefits. Lossy compression can be thought of as an approximation technique allowing the fluid trade-off of output accuracy and improved bandwidth. Most current research on lossy compression has focused on domain-specific implementations, such as lossy compression algorithms for video or images, or for specific scientific datasets. However, if applied to recognition, mining, and synthesis applications, lossy link compression could take advantage of these applications' error resilience for even greater bandwidth, scalability, and energy efficiency gains.

One of the greatest challenges of link compression is that it does not always result in improved performance or energy efficiency. Since link compression introduces the additional computational overhead of compression and decompression of the data, its benefits are tightly coupled to the CPU and network state, the type of communication channel, and the compressibility of the data. Link compression is likely to be beneficial in cases in which there is a high communication-to-computation ratio, the data needs to be transferred over a long or high power channel (e.g., wirelesss), and the data is sparse or with high value locality and not already compressed. As discussed in the compression overview (Section 2.1), adaptive compression techniques are able to selectively apply compression only when it is beneficial. Thus, any application of compression for approximate communication is likely to require such adaptive techniques.

2.2 Relaxed Synchronization

Accesses to shared data should be synchronized to guarantee correct execution of parallel programs. Both the type and the frequency of synchronization strongly depend on the underlying algorithm. Synchronization mainly serves to establish producer–consumer semantics and mutual exclusion. While producer–consumer synchronization enforces dependencies among parallel tasks of execution, mutual exclusion restricts accesses to data objects to one parallel task at a time. A typical parallel program consists of multiple synchronization points (possibly each conforming to a different category) which can be interleaved in numerous ways. Independent of the frequency, category, or interleaving, synchronization dictates a total or partial order on parallel tasks. Hence, each synchronization point represents a point of serialization; accordingly, synchronization-incurred overheads can easily hurt scalability of parallel programs.

To improve scalability in the face of inevitable synchronization, recent studies [101, 126, 128] proposed to relax a subset of the synchronization points and to exploit the implicit noise tolerance of an important class of future parallel applications—(R)ecognition, (M)ining, and (S)ynthesis [28] in mitigating relaxation-induced atomicity/ordering violations or data races. By relaxing synchronization points contributing to data flow (not control flow), relaxation-induced errors manifest as degradation in the application output quality. Hence, relaxed synchronization can be used to trade off output quality for improved scalability until the point at which output degradation becomes unacceptable.

2.2.1 Relaxed Speculative Synchronization. Relaxation idea is not limited to classic synchronization. Under speculative synchronization [62], parallel tasks of execution (threads or transactions) proceed speculatively without waiting at synchronization—i.e., serialization—points. Until speculation gets resolved, the speculative state should not become visible, i.e., committed; hence, the speculative state should be buffered. These systems require not only extra storage for the speculative state but also a framework to detect conflicts among parallel tasks and to orchestrate rollback to a safe state in the case of misspeculation. Once speculation is deemed safe, coordination in committing speculative state per task is also necessary. Relaxation techniques can be regarded as lightweight speculative synchronization techniques in which the overhead of storage, conflict detection, rollback, or commitment is notably reduced (if not eliminated) by exploiting inherent noise tolerance of RMS applications. Aggressive relaxation can cut off these overheads as long as the degradation in the output quality remains at acceptable levels. Alternatively, depending on preset thresholds of criticality, only a critical subset of tasks can be tracked for speculative buffering, conflict detection, recovery, and commitment.

2.2.2 *Relaxed Shared Data Access.* Recent studies [126, 128] investigated the idea of eliminating the synchronization points of error-tolerant applications to gain speed-up, while keeping output accuracy at an acceptable level. Rinard [128] focuses on a common case in which different threads want to access (update) the same data structure in parallel. Usually, mutual exclusion is deployed in such cases to protect conflicting accesses to the shared data. Otherwise, it is very likely to observe out-of-bounds accesses, resulting in early application termination without generating the output. This study introduces a data structure in which unprotected conflicting insert of elements would not cause any such catastrophic scenarios. In the proposed array structure, conflicting accesses may lead to dropped elements, which degrade output quality rather than causing address space violation. Using this data structure, eliminating mutual exclusion generates performance benefits, while output quality may be degraded due to elements being dropped. In addition, Rinard looks at another case–barrier relaxation. Barrier synchronization is used to make sure that all threads are at the same computation phase. Due to workload imbalance, some threads may need to wait for other threads (stalled), reducing parallelism. In the relaxed scenario, a thread simply passes the barrier without waiting for other threads to get to the barrier. As a result, performance is improved, although data dependencies may cause lower output quality.

2.2.3 *Relaxed Protected Code Sections.* Renganarayana et al. [126] analyze how relaxing mutual exclusion may affect different applications' performance and output quality. They consider both classic (blocking) mutual exclusion and speculative (nonblocking) locks as case studies. In the study, synchronization points are eliminated completely, leading to conflicting accesses to shared resources. Hence, output accuracy may be affected. They also propose a mechanism called Relax and Check (RaC) that allows control over the quality of the output. In RaC, both the original version of the algorithm and the relaxed version are compiled with the application. At the end of the relaxed portion, a check is performed to determine the quality of the output. If the quality is not acceptable, then the original version with full synchronization is executed. This is a simple scheme that can guarantee that the final output meets a certain quality goal. However, it may degrade performance in some cases since it may require more than twice as much work when the relaxed version produces an unacceptable result. Moreover, some applications may not have a method to verify the results' acceptability.

2.2.4 *Benefits and Challenges of Relaxed Synchronization for Approximate Communication.* Relaxed synchronization is an attempt to extend approximate computing concepts to tackle the high impact that synchronization has on the scalability of parallel applications. While the idea of optimizing synchronization points has always been a major concern of parallel application developers, relaxed synchronization takes a more aggressive approach by removing as many synchronization points as possible while keeping output degradation within acceptable limits. Relaxed synchronization was included as a possible approximate communication technique due to the possibility of reducing synchronization overheads.

The main benefit of relaxed synchronization is in reducing serialization points. Renganarayana et al. [126] showed that synchronization overhead is a significant portion of the execution time in a parallel k-means benchmark. The fraction of the execution time spent on synchronization increases with thread count, which severely limits scalability. This may happen even in applications with a high level of data parallelism, if, for example, these applications have a high degree of interthread dependencies and poor load balancing. In the k-means benchmark example, it was shown that synchronization overhead increases from approximately 70% for 2 threads to about 90% for 8 threads. Thus, relaxed synchronization can be a potential solution for the effect of synchronization in diminishing returns of higher thread counts. Another benefit of relaxed synchronization is that it is completely implemented in software by modifying existing applications, making it easier to implement and control by the application developer. As mentioned in [126], this can also be automated using autotuners or iterative compilers.

However, relaxed synchronization has many challenges that make it difficult to apply beyond a very restricted set of applications. Since relaxing synchronization allows the possibility of many different interweavings of thread executions, it introduces nondeterminism in the solutions. Another issue is the choice of which synchronization points to relax. Some synchronization points are essential for proper program execution while others are safe to approximate. Moreover, the safe-to-approximate synchronization points may have different impacts on output degradation. Good candidates for approximation are synchronization points with high impact on execution time or power and low impact on output quality. However, the selection of this ideal set of synchronization points may be complicated by the fact that the ideal set may be dependent on the input.

2.3 Value Prediction

Value prediction is based on the observation that many applications exhibit significant data value locality [88]. Value prediction seeks to exploit this predictability of data values to break true data dependencies in a processor's pipeline. A value predictor, perhaps similar in design to the pervasive branch predictors, may be used to *predict* instruction inputs that are dependent on long-latency operations. The processor may then use the predicted value to continue execution speculatively while waiting for the long-latency operation to finish.

Sazeides and Smith [141] demonstrated one of the first designs of value predictors that could achieve between 56% and 92% value prediction accuracy. They divided value predictors into two types: computational predictors, such as last-value predictors and stride predictors, and context-based predictors. Lipasti et al. [88] showed the benefits of load value prediction for collapsing true data dependencies and reducing average memory latency and bandwidth requirements. It was shown that many load instructions have high value locality—as much as 80%, on average—in some applications. Their work also included a constant verification unit that identifies load instructions that always retrieve constant data and skips memory accesses for these loads. Calder et al. [22] analyzed value prediction in light of predictor capacity constraints and misprediction penalties. It was shown that accurate value confidence prediction is key to obtaining speedups from value prediction. Follow-up work aimed at improving predictor accuracy includes global context predictors [107], difference value predictors [56], and several other hybrid designs [166].

2.3.1 Challenges in Value Predictor Design. Most of these early research efforts in value prediction focused primarily on the accuracy and coverage of their predictor designs. However, they mostly ignored the high performance loss associated with misprediction recovery. A real implementation of value prediction must include not only a high-accuracy value predictor but also a rollback mechanism that can recover the processor's execution state in case of a misprediction. Implementing a rollback mechanism requires modifications to the out-of-order pipeline to support data checkpointing, data validation, and recovery logic. This leads to increased complexity and power consumption. Moreover, the average performance benefit per accurate prediction is quite small, on the order of half a cycle per successfully predicted instruction [118, 119]. Meanwhile, the misprediction penalty may be quite high, typically on the order of tens of cycles depending on the recovery mechanism used. One common mechanism is called pipeline squashing, which is the same mechanism used to recover from branch mispredictions. While it is relatively simple to implement, especially if data validation is done at the commit stage, it is very costly in terms of wasted cycles on misprediction. Selective reissue, on the other hand, allows only the mispredicted instruction and its dependency chain to be replayed. While this allows for a smaller, albeit still significant, misprediction recovery penalty, it comes at a much higher cost in design complexity.

Finally, in addition to the rollback overhead, other implementation issues with value prediction include the high cost of predictor tables, prediction latency, complexity of data checkpoint logic, increased power consumption, and difficulties predicting back-to-back loads and floating-point numbers [120].

Despite all these challenges, recent work is revisiting value prediction as single-core performance regains research attention. Perais and Seznec [119] have shown that by using a high-accuracy confidence predictor, they can offset the high misprediction costs associated with pipeline squashing recovery. Their design achieves high prediction accuracy (>99.5%) with low coverage by using probabilistic saturating confidence counters. It also reduces misprediction resolution complexity by employing pipeline squashing at the commit stage and adds a latency-tolerant predictor based on global branch history. Follow-up work also addressed issues with Physical Register File access [118] and multiple predictions per cycle [120]. Reported speedups ranged from 5% to 65% on select benchmarks.

2.3.2 Value Prediction and Parallel Systems. In the context of parallel computer architectures, value prediction may be particularly attractive as a method for reducing communication overheads. Conventionally, data values produced in one core but consumed at a different core require that the data be transferred to the consumer core, typically over an interconnection network, before they can be used. With value prediction, however, the consumer core may proceed with execution while it waits for the requested value to arrive. It may also allow the *parallelization* of certain sequences of code by breaking data dependencies or allowing cores to speculatively move ahead of synchronization points. The potential benefits of value prediction for parallel computing were shown in [89]. This paper presents a reformulation of Amdahl's law, which factors in the parallelization effect from value prediction. Their formula provides an upper bound on the maximum speedup attainable based on prediction accuracy. They also developed a theoretical model based on information theory for the maximum achievable prediction accuracy. Their experimental results showed that value prediction can improve performance by as much as 267% based on simulations with the PARSEC and SPLASH-2 benchmarks.

Another potential use of value prediction is in improving the performance of thread-level speculation (TLS). The basic premise of TLS is to implement either compiler or hardware mechanisms that automatically parallelize sequences of code that may not actually be independent. The selected sequences of code (epochs) are executed speculatively in parallel. At the end of an epoch, if data dependence violations are detected, a recovery mechanism is run; otherwise, the epoch's data are committed. One key design issue for TLS is handling these data dependencies between concurrent epochs. In principle, value prediction could be used to reduce data dependency violations by predicting the dependent values. Steffan et al. [152] compared three main value communication mechanisms for TLS: speculation, synchronization, and prediction. However, their results showed that while value prediction can be effective, other techniques—such as silent stores—yield similar results at a lower cost. Cintra and Torrellas [34] also evaluated using value prediction as part of their learning-based TLS framework but also found no real benefits to value prediction. Nevertheless, software-only value predictors [124], which avoid the high costs associated with hardware value predictors, have been shown to be an effective technique, suggesting a promising direction for value prediction in the context of TLS.

It is worth noting that additional challenges exist for value prediction when applied in a multithreaded environment. Most previous works on value predictors have assumed a single-threaded uniprocessor environment and have mostly neglected issues related to value prediction in multithreaded architectures, such as simultaneous multithreading (SMT), coarse-grained multithreading (CMT), single-chip multiprocessing (CMP), and traditional multiprocessing (MP). Martin et al. [93] demonstrated that naïve implementations of value prediction may inadvertently violate a processor's memory consistency model by allowing "a processor to relax the ordering between data dependent operations." They have shown that this issue may be remedied by borrowing techniques from aggressive implementations of sequential consistency, but these come at a performance cost. Finally, another little-researched topic is the effect of context switches on a predictor's accuracy. This issue is not unique to value prediction, though. Lee et al. [81], for example, showed the effects of context switching on branch predictors. Hybrid predictors [52, 144], which combine fast-learning predictors with slower but more accurate predictors, is a common mitigation technique for branch predictors that may also be applied to value predictors.

2.3.3 Value Prediction and Approximate Computing. As the previous sections have shown, despite considerable research, there are many challenges to traditional value prediction that have prevented its widespread adoption. However, new research has looked at applying approximate computing techniques to value prediction as a way to reap the benefits of value prediction without the associated costs. Load Value Approximation (LVA) [97] and Rollback Free Value Prediction (RFVP) [173] are similar works that explore the idea of using predicted load values as approximate data values. Thus, these approximate load value predictors do not verify the predicted values and do not require rollbacks. The main idea is to trade the performance benefit of an RFVP for the introduction of errors in the output.

Below are a few of the benefits of approximate value prediction:

(1) Approximate value prediction does not require a recovery mechanism for mispredictions, thus avoiding the need for complex rollback hardware, one of the main drawbacks of traditional value prediction.
(2) Removing the requirement of precise prediction helps to overcome the difficulty in floating-point prediction.
(3) While traditional predictors avoid costly rollbacks by making a prediction only if confidence is high, an approximate predictor can relax the confidence requirement, leading to higher predictor usage and better performance gains.
(4) Previously mentioned memory consistency issues with traditional value predictors may be ignored with approximate value prediction.
(5) Most important, in an approximate load value predictor, it is not necessary to always fetch a block from memory on a cache miss. Unlike traditional value predictors that must fetch blocks to validate the predicted values, in this approximate design, blocks need only be fetched in order to update the predictors. Thus, in a shared memory system, we may achieve communication reduction by predicting the shared values that would need to be transmitted between cores. Hence, the frequency by which the value predictors are updated becomes a tunable parameter for the trade-off between the amount of communication reduction and the output error.

2.3.4 Trade-offs of Approximate Value Prediction. While LVA and RFVP have several potential benefits over traditional value prediction, they do come with some trade-offs. For example, as mentioned earlier, one benefit of LVA and RFVP over traditional value prediction is that they require no mechanism to rollback in case of a misprediction. Rollback hardware incurs a high overhead due to all the logic required to keep track of the speculative state changes, the value verification logic, and the rollback logic itself. This is not only a high hardware overhead, but also a high development cost since all the complex logic needs to be thoroughly validated. Approximate value prediction does not require any recovery hardware, making its implementation simpler and cheaper. Moreover, since it does not modify the main execution pipeline, it can be designed as an independent performance module that can be programmatically enabled or disabled. However, while traditional value prediction is completely transparent to the programmer, since the correctness of execution is guaranteed by the recovery hardware, approximate value prediction requires significant programmer involvement in identifying load values to approximate. This can be done using code annotations, as in LVA, or with the help of profiling tools, as in RFVP. In either case, assuming that value prediction is an optional CPU performance feature, programmers may be reluctant to use it if it is too difficult or time-consuming.

Perhaps approximate value prediction techniques such as LVA and RFVP may find the greatest benefits in the context of high-performance computing, which utilizes massively parallel architectures with long-latency interconnects. As [89] showed, value prediction has great potential to increase an application's data parallelism and its potential maximum speedup from processor scaling, as per Amdahl's law. Moreover, several value prediction papers [87, 88, 97, 137, 156, 173, 181] have shown that the main benefit of value prediction comes from its memory latency overhead reduction. Since communication latencies in massively parallel architectures are presumably much higher, it is expected to yield greater benefits from value prediction. Finally, as the RFVP paper [173] showed, approximate value prediction not only reduces communication latency but also communication bandwidth, which again can be a bottleneck in many massively parallel architectures.

2.3.5 Benefits and Challenges of Value Prediction for Approximate Communication. In this section, we present two papers that explored the concept of approximate value prediction: LVA [97] and RFVP [173]. Both papers show the benefits of approximate value prediction over traditional value prediction. Instead of requiring data to be fetched from memory on every cache miss in order to validate the value prediction, both papers implement mechanisms that allow memory fetches to be performed less often, as they are used only for updating the value predictors. The result is that, in addition to the reduced average memory latency from traditional value prediction, approximation also reduces the memory bandwidth requirement. The same technique could also be easily applied to reduce communication in parallel computers since the value predictors could also be used for predicting remote data, thus requiring fewer communication messages.

Approximate value prediction helps overcome many of the shortcomings that have kept traditional value prediction from reaching commercial application. However, it is still somewhat of a complex architectural feature to implement and there is a large design space for predictor designs that need to be explored. Another issue is that, if we assume that an approximate value predictor would be implemented as an extra feature in a microprocessor, it is unclear if it would be used often enough to justify the area, power, and design costs associated with it. Nevertheless, it does hold the promise of significantly reducing interconnect and memory bandwidth requirements, as was shown in the LVA and RFVP papers.

3 PUTTING IT ALL TOGETHER

Section 2 provided an overview of three promising techniques for approximate communication. In this section, the three techniques are brought together and compared side by side based on a common evaluation framework. While the comparison is mostly qualitative in nature, we present some evaluation results from relevant papers in order to support conclusions on each technique's potential for reducing communication on massively parallel systems.

3.1 Evaluation Framework

Approximate computing is a relatively new research field; there are still many opportunities for research, especially in the area of approximate communication techniques. However, it is important that new ideas and techniques in this area be evaluated in a consistent manner, taking into consideration the most important aspects and goals of approximate communication. In addition to the speedup and energy consumption normally reported, approximate techniques require careful consideration of output degradation and how it varies based on the application. Many techniques only apply to certain applications or only have benefits when applied to a certain communication and/or computation pattern. Additionally, many techniques give programmers the flexibility to specify the application's output quality goals that, in turn, affect the performance, power, or bandwidth gains achievable from the technique. Therefore, a proper evaluation should carefully consider these trade-offs when drawing conclusions about a technique.

We have selected what we believe are the six most important evaluation parameters that must be considered when determining the feasibility and merits of an approximate communication technique. They include metrics that are commonly reported on system papers (i.e., performance, energy, overheads) and those specific to approximation techniques (i.e., output degradation). The goal is to consider all these metrics together in order to provide a fair evaluation of each technique. Here, we give a background on each evaluation parameter and an explanation of their importance for approximate communication. In the next section, we use this framework for a comparative analysis of the surveyed techniques.

Communication Cost Reduction. The main purpose of approximate communication is to reduce the communication cost of a parallel application. Culler et al. [39] have defined the following communication cost model for a message-passing system:

$$C = Frequency * \left(Overhead + Delay + \frac{Length}{Bandwidth} + Contention - Overlap\right) \tag{1}$$

where

Frequency is the frequency of communication messages in the program.
Overhead is the combined overhead of handling initiation and reception of a message on the sending and receiving processors, assuming no contention with other activities.
Delay is the nonoverhead delay for the first bit of the message to reach the destination processor or memory.
Length is the average length of a communication message, which can be further broken down as the total amount of data communicated by the program divided by the number of messages.
Bandwidth is the point-to-point bandwidth of communication afforded for the transfer by the communication path, excluding the processor overhead.
Contention is the time induced by contention for resources with other activities.
Overlap is the amount of the communication cost that can be overlapped with computation or other communication.

From this model, it becomes clear that there are many different approaches to reducing communication costs in a parallel system. While there is a lot of research looking to improve overhead, delay, contention, and overlap, approximate communication techniques target primarily the number and size of messages. Thus, if we ignore the overhead, delay, contention, and overlap components in the above equation, we can approximate the communication cost to $C = Frequency * (\alpha * \frac{Length}{Bandwidth})$ or $Comm_{Req} = \alpha * N * Q$, where α = *scaling factor*, N = *the number of messages*, and Q = *average size of communication messages.* Techniques that aim to maximize information content, minimize the data footprint, or communicate only approximate values, such as lossy compression, aim to reduce Q. On the other hand, techniques that aim to minimize the frequency of communication or the number of times required to communicate, such as value prediction or relaxed synchronization, aim to reduce N.

Performance Improvement. Performance, as measured by the application's execution time and/or speedup over a baseline, is a common metric for evaluating and comparing the benefits of different techniques. Approximate communication techniques can impact performance either directly or indirectly. Some techniques may improve performance directly by, for example, skipping or approximating certain time-consuming computations. Communication reduction may also impact performance indirectly, however. Generally speaking, communication-bound applications will typically experience the greatest performance benefits from approximate communication techniques, while compute-bound applications will see more modest improvements. Moreover, in some cases, the technique's overhead may overcome its benefits, which may lead to performance degradation. Therefore, in addition to communication cost reduction, we must also evaluate an approximate communication technique's performance impact.

Energy Reduction. This metric is becoming ever more important in modern systems. Previous studies [146] have shown that communication power is already one of the main power draws and its share of the total system power budget is expected to increase as CPU core efficiency improvements outpace improvements in interconnect efficiency. It is important to differentiate power versus energy efficiency, however. Energy is the product of power and time. Thus, any technique that improves the execution time of a benchmark without increasing system power considerably will see an energy consumption improvement. While most research papers report their energy consumption savings, peak power is actually a greater concern for parallel systems. One of the biggest challenges in scaling up massively parallel systems is heat dissipation, and the ability of a cooling system to maintain a safe operating temperature is directly related to the system's peak power. On the other hand, energy efficiency is also important but for a different reason. Energy efficiency translates into savings in electricity consumption for large systems. Energy efficiency is also of great importance for battery-operated devices in order to extend their battery lives. We will consider both power and energy efficiency improvements in our evaluation of the techniques in this article.

Applicability. Not all approximate communication techniques yield the same benefits on all applications. Some techniques target applications with high memory bandwidth requirements or with a lot of interprocessor synchronization and will sometimes have significant slowdowns if applied outside their scope. The benefits of certain techniques can vary drastically depending on the application to which they are applied. Moreover, some techniques are application aware, meaning that they are able to dynamically adapt to the running application. Thus, it is important to understand if a given technique is generally applicable or if it is optimized for a certain application.

Overhead. This blanket evaluation parameter includes a diverse set of implementation specific costs, which include runtime setup overhead, chip area, design complexity, price, and programming effort. For software-based techniques, setup time and programming effort might be the main concerns, while for hardware-based techniques, chip area and power consumption may be the major concerns. It is important to understand if and when a technique's cost may overcome its benefits. Sometimes, techniques may look great in theory and have good speedups on a simulator but fall short when implemented on a real system. It is important for system researchers to carefully consider all different sources of overhead that may hinder the expected performance of a given technique.

Output Degradation. The key aspect of approximate computing is the trade-off between output quality and energy efficiency or performance improvement. Therefore, a proper evaluation of a given approximate computing technique requires a measurement of the output degradation that captures all user-relevant information. Unlike other metrics, such as speedup and energy efficiency, output-degradation metrics are application dependent and not as straightforward to evaluate. Akturk et al. [4] present an overview of some of the challenges and pitfalls in evaluating accuracy loss in approximate computing. Some of the key points made in their work include:

(1) *Accuracy versus validity*: Some approximation techniques may lead to invalid outputs, requiring validity checks and accuracy metrics.
(2) *Accuracy versus acceptability*: Different applications will be more or less resilient to the amount of output degradation that is acceptable. Likewise, even the same application may have different accuracy requirements depending on the usage scenario.
(3) *Relative metrics*: Metrics computed from the absolute values of the output may introduce a bias in the output. For example, if the output values are coordinates, using absolute values may give higher importance to points further from the origin. Thus, good metrics should be relative to the range of the output.
(4) *Averaging effects*: Average metrics may hide differences between small, frequent errors in the output versus large, infrequent ones, which may be more or less important to different applications.
(5) *Input data*: A given approximation technique may have a different impact on output quality for different input sizes or values. Moreover, some inputs may produce invalid results or even crash the application. It is thus essential to cover such cases when evaluating or testing theses techniques.
(6) *Nondeterminism*: The best candidate applications for approximation are typically those of a probabilistic nature, which will have inherent nondeterminism in the output. Moreover, the approximation technique itself may introduce nondeterminism if, for example, the approximate technique relaxes serialization guarantees in a parallel application. Therefore, accuracy metrics must be based on statistic guarantees of the output quality and must be able to isolate inherent and approximation-induced nondeterminism.

A proper evaluation of output degradation from approximation techniques should carefully consider the most appropriate metrics. Typically, it may be necessary to include more than one metric. For example, in order to account for the averaging effects, it may be necessary to capture the average alongside the min-max values or an error distribution. One will typically also have to reason about the validity and acceptability requirements of each benchmarked application. Finally, given the probabilistic and nondeterministic nature of many approximation techniques, evaluation will typically require a large number of runs in order to search the input space for possible invalid cases, to gather statistical data on the relationship between input data and output degradation, and to find the error distribution due to approximation-induced nondeterminism. Since approximate computing research is still in its early stages, many works have not been so rigorous in their evaluations. As the field develops, such rigor will become necessary for properly comparing new techniques.

3.2 Comparison

This section compares the techniques in this article using our evaluation framework. Table 2 gives a summary of the main points discussed in this section.

3.2.1 Communication Cost Reduction. Compression reduces communication cost mainly by a reduction in the amount of data to be communicated (reduce Q). The amount of data that is reduced will be dependent on the compression ratio that, in turn, is dependent on the compression algorithm and the entropy of the data. Lossy compression typically yields greater compression ratios than lossless at the cost of some data degradation. The effect of reducing the amount of data is an improvement in the available communication bandwidth. In a high-contention scenario, however, bandwidth improvement will also lead to an improvement in communication latency. In [6], link compression using a Frequent Pattern Compression scheme was shown to reduce bandwidth between 23% and 41%. In [41], compression in the network interface controller of an NoC was able to reduce network latency by 20%, on average. Wiseman et al. [170] compare the compression ratios between four different compression schemes (Burrows-Wheeler, Lempel-Ziv, Arithmetic, Huffman), achieving between 10% and 50% compression. Bui et al. [20] include compression ratios between Zlib and BloSC, resulting in an average compression ratio close to 2x. In comparison, Iverson et al. [68] showed the benefit of lossy compression over lossless. In their evaluation, lossy compression was able to reduce the dataset size to 2% to 5% of original size, while lossless compression only achieved 40% to 80%, which is consistent with results from other works.

The relaxed synchronization technique is more limited in terms of its ability to reduce communication cost. However, relaxed synchronization may reduce the number of synchronization messages needed to be transmitted (reduce N). Depending on the synchronization protocol used, this reduction in the number of messages can improve contention and interconnect energy even if these messages are typically small.

Approximate value prediction reduces communication cost by skipping a fraction of data fetches (called the drop rate or approximation degree). The result is an improvement in bandwidth requirements. Value prediction also improves average memory latency to CPU through speculative execution; however, the actual channel latency is not affected. The two main papers on approximate value prediction are LVA [97] and RFVP [173]. LVA reported an average 37.2% reduction of interconnect traffic with an approximation degree of 16 (meaning that only 1 out 17 loads will trigger a memory access) and over 39% reduction in the number of fetches from main memory. RFVP reported GPU bandwidth reductions of 20% to 90% for quality degradations of less than 10%. These results show the potential benefits of their techniques in reducing communication. Note that these results vary depending on the application, output degradation level, and target architecture.

3.2.2 Performance. Compression has the most performance benefit for applications that are bandwidth constrained [6]. Nevertheless, even applications that are not bandwidth constrained may still see performance improvements due to compression since communication typically occurs in bursts. Bui et al. [20] reported a 40% increase in performance from using lossless compression. Moreover, Bicer et al. [18] showed the performance benefits of using lossy compression. Their lossy compression algorithm improved performance by over 2x compared to lossless compression. However, compression may not be beneficial in cases in which the overhead of compression and decompression is greater than the communication reduction. In [75], Krintz and Sucu developed the ACE (Adaptive On-the-fly Compression) scheme to deal specifically with cases for which compression may not be beneficial and show how speedups may change depending on network and system. Their results show an average speedup of around 9% to 20%. Likewise, Alameldeen and Wood [6] show modest average speedups for compression alone of 3% to 20%.

Table 2. Comparison of the Surveyed Approximation Techniques in Terms of Evaluation Parameters

	Compression	**Relaxed Synchronization**	**Value Prediction**
Communication Cost Reduction	Reduces Q Link compression reduces bandwidth Better compression ratios with lossy compression	Reduces N Reduces number of synchronization messages	Reduces N Reduces number of data fetches required
Performance	Performance improvement dependent on bandwidth constraint Performance degradation possible	Performance improvement dependent on synchronization overhead No performance degradation	Performance improvement from speculative execution of long latency operations and bandwidth reduction No performance degradation
Energy Reduction	Energy reduction dependent on the trade-off between compression and decompression overhead and compression ratio	Energy reduction dependent on the reduction in execution time	Energy reduction dependent on the approximation degree
Applicability	Typically, best compression ratios will be application specific. General methods exist but with lower compression ratios.	Technique is applicable to any domain, but implementation will be application specific.	Application agnostic since this is a low-level hardware feature. The application's value predictability will influence performance improvements.
Overhead	Compression and decompression time is main overhead. Hardware engines and parallel compression algorithms can help.	Little to no runtime overhead associated with this technique, except if using runtime monitors. The main cost is in programmer effort for code annotations.	Chip area and power overhead for prediction tables. May require program annotations for load values to predict.
Output Degradation	Controllable on most compression methods by selecting the compression ratio	Controllable by selecting which synchronization point to relax	Controllable by changing the approximation degree (LVA) or drop rate (RFVP)

Performance benefits under relaxation depend on the frequency of accessing a synchronization point and the ratio of conflicting accesses in the original application. Hence, for the set of applications used in the study [126], we see a wide range of speedup benefits with respect to the original execution. For instance, their results show a speedup of 13x to 15x for kmeans, 2.6x for ssca2, 1.5x for labyrinth, 3.2x for BFS of Graph500, and negligible for yada.

In both LVA and RFVP, performance gains are related to the level of allowable output degradation. LVA uses what they call the *approximation degree* to tune the level of output degradation. Approximation degree is the number of main memory accesses that are skipped on cache misses. LVA reported an average speedup of 8.5% for an approximation degree of 0 with about an additional 2% with approximation degree of 16. LVA performance benefits come primarily from L1 miss latency reduction (41%) due to lower misses per kilo instructions (MPKI). RFVP reported a 36% speedup with 10% quality loss on GPU benchmarks and an 8% speedup with 0.9% quality loss on CPU. These results show that highly parallel architectures, such as GPUs, stand to gain the most in terms of performance improvements while modest speedups with very low output errors are possible on CPUs.

3.2.3 Energy Reduction. Compression improves energy efficiency by reducing the number of bits that need to be transmitted over the communication channel. However, the compression and decompression steps also consume energy; thus, as for performance, energy efficiency will be dependent on this trade-off. [145] showed that memory energy is usually much higher than compression and decompression. In [41], it was shown that link compression can reduce network power in an NoC by an average of 10%. Relaxed synchronization energy reduction comes primarily from the reduced application execution time. The energy savings from approximate value prediction comes from reduced runtime and fewer fetches from memory. The energy overhead includes primarily the predictor logic and table. RFVP reported a 31% energy reduction with an 8.8% quality loss and 2x energy reduction with 10% quality loss on the GPU benchmarks. For the CPU benchmarks, RFVP reported 6% energy reduction with 0.9% quality loss. LVA reported an average 12.6% energy reduction with approximation degree 16.

3.2.4 Applicability. Compression has the widest applicability of all the techniques. However, the benefits of compression will vary greatly between applications depending on the compressibility of the application's data. Lossless compression can be applied to any application, but lossy compression should be applied only to error-tolerant applications. Relaxation is also applicable to a wide range of applications, from data mining and machine learning to numerical optimization, graph processing, financial modeling, mathematical programming, and design automation, as shown in [126]. Applications with the greatest number of nonessential synchronization points will have the greatest benefits. As a hardware feature, approximate value prediction is application agnostic. However, the best results will depend on the predictability of the data.

3.2.5 Overhead. The main source of overhead for compression is in the compression and decompression. It is possible to reduce these overheads with the use of specialized hardware or more efficient parallel compression algorithms. Relaxed synchronization, on the other hand, has no runtime overhead. It does, however, require modifications to either binary or source code. Value prediction increases the chip area and may require program modifications. Nevertheless, in many cases, the chip area overhead can replace existing structures. For example, in the RFVP work, it was shown that even simple and small predictors on the order of 10 to 20KB can achieve good prediction rates and speedups that exceed a comparable increase in cache size. Additional programmer effort can also be viewed as a type of overhead if value prediction requires code annotations for identifying the load values to predict.

3.2.6 Output Degradation. Compression incurs output degradation only in the case of lossy compression. The level of output degradation can be controlled by the selection of compression algorithm. Many algorithms may also allow some control over compression ratio. Most papers on link compression have used lossless compression since it can be applied to any dataset, not only to error-resilient datasets. Iverson et al. [68] demonstrated lossy compression of scientific datasets of up to 1% of the original size with average PSNR of 43.00 and 3% of the original size with average PSNR of 63.30. They also showed a mechanism to bound the compression error, which is important for the technique to be useful for a wider range of applications.

Output degradation can be controlled in relaxed synchronization by the choice of which synchronization points to relax. This will require a classification of the synchronization points in terms of their impact on output quality and program safety either manually by the programmer or through profiling tools. It is also possible to devise methods that choose whether to relax based on runtime monitoring of the output degradation, even though such mechanisms may be difficult to implement in practice.

For approximate value prediction, both LVA and RFVP implement mechanisms for controlling the amount of output degradation. One method is to control how often data is fetched from memory on a cache miss. In traditional value predictors, cache misses still trigger a read to memory to retrieve the data. The processor continues execution speculatively while waiting for the predicted data to arrive so that its prediction can be verified. In the proposed approximate designs, however, they can selectively decide not to fetch a block into the L1 cache, since no verification is required, while still allowing a few fetches in order to update the value predictors. In LVA, this is called the "approximation degree"; in RFVP, it is called the "drop rate." Another method is to relax the confidence windows. Value predictors often include a confidence estimator that determines whether a prediction should be performed or not. Since the approximate design is more resilient to mispredictions, the confidence threshold can be relaxed. Both methods allow fine runtime control over the speedup versus quality trade-off and are a major advantage of this technique. In general, the authors have shown that significant speedups are achievable for very–low–output degradation.

3.3 Discussion

The comparison above has shown some of the trade-offs with each of the three techniques (compression, relaxed synchronization, and value prediction) as they relate to approximate communication. It is also worth noting that the metrics in our evaluation framework have a degree of interdependency. For example, communication reduction will typically lead to performance improvement for most applications. However, the actual speedup will depend on the level of the system's communication bottleneck. Among the three techniques, compression and value prediction were shown to provide significant bandwidth and latency improvements; however, only modest speedups were seen on many systems. This points to the fact that most of the performance benefit from communication reduction comes from improving the scalability of parallel applications rather than single-core performance. Therefore, approximate communication techniques will find the best applications on systems with a high core count.

The importance of considering the application's characteristics was also demonstrated. Relaxed synchronization, for example, is most effective with the help of intimate knowledge of the application. Even though compression and value prediction can be made fairly efficient using general-purpose algorithms, there are unique characteristics of the application, such the sparsity and entropy of the data, that significantly affect the scalability and/or performance benefits they will yield for the given application. Therefore, the best approximate communication techniques will be application aware, that is, they will be able to adapt to the running application's unique characteristics.

Finally, the possibilities of combining these techniques is also worth mentioning. This is most beneficial when the techniques are either complementary, meaning that they support each other, or orthogonal, meaning that they target distinct bottlenecks in the system and are able to work independently. An example of using the techniques in a complementary fashion would be using value prediction to approximate the data values in critical code sections that have been relaxed under relaxed synchronization. An example of an orthogonal combination would be using compression and value prediction. It would be possible to design these techniques such that they would have minimum interaction, thus allowing value prediction to simultaneously reduce the number of communication messages while compression could reduce the size of those messages.

4 POTENTIAL RESEARCH DIRECTIONS

Based on the discussion above, this section includes some suggestions for future research. The goal of these suggestions is to serve as a brainstorming of ideas on the most efficient ways to apply these surveyed techniques as well as some ideas for other promising techniques for approximate communication.

For compression, we see an opportunity for research into adaptive lossy link compression. Such a technique could combine the high compression ratios of lossy compression with the communication cost reduction from link compression. Making compression adaptable would allow for a fluid trade-off of performance and error degradation, as well as a dynamic response to the system state. As shown in Section 2.1, this technique could be tailored for either NoC or high-performance computing architectures.

For relaxed synchronization, there is a need to better understand the relationship between synchronization points and output quality. Since most research on synchronization has been done under the assumption that any error is unacceptable, there is not much research that considers the exact effect of relaxing synchronization. In order for this technique to find greater use, there would also need to be profiling tools and/or iterative compilers to aid in the selection of synchronization points to relax.

For value prediction, it would be worth investigating the performance benefits of approximate load value prediction on massively parallel architectures for which value prediction may have the greatest benefits due to large communication overheads. There is also the need for improved programming models and profiling tools to help in the development of applications for approximate value prediction. Finally, there are opportunities for variations on hardware-centric design. For architectures with very long communication latencies, software-based or hardware- assisted value prediction may provide performance benefits and lower implementation costs.

There is also a plethora of available software optimizations similar to relaxed synchronization that could be applied to existing applications. Approximate algorithms are commonly used for the solution of NP-complete problems, but those are not the only applications in which only an approximate solution is good enough. Similar concepts can also be applied to algorithms that have an exact polynomial time solution but that could be optimized through approximation. This also applies to the communication patterns within an application. In many cases, it may be faster or more energy efficient to transmit only approximations of the data, such as a statistical distribution or a max and min, rather than the entire dataset at the cost of only a small accuracy loss. Thus, there is opportunity for developing tools that could aid developers in making these decisions. One example of work in this area is Paraprox [132]. Paraprox is an automated tool for software-only optimizations based on approximation of communication patterns in data parallel programs. It is a compiler-based software system that automatically detects common communication patterns in data-parallel programs and substitutes them with approximate versions. It specifically targets common communication patterns such as map, scatter/gather, reduction, scan, stencil, and partition. Thus, it provides a software framework in which several techniques can be selectively applied as appropriate.

Finally, additional research may explore employing accelerators or processors-in-memory (PIM) [91, 115, 143, 182] to speed up certain approximation kernels. For example, one may conceive of using the logic layer of a 3D stacked DRAM to implement an accelerator that performs efficient, lossy compression on data to be transmitted over the memory channel. Accelerators could also be designed to perform approximations or analysis on data in-memory so that only summarized results need to be transmitted.

5 CONCLUSION

Approximate computing is a new paradigm that exploits an application's inherent error resilience and allows a trade-off between output inaccuracy and performance or energy efficiency. Nonetheless, communication is one of the biggest challenges to achieving scalable performance and energy efficiency in parallel systems. This leads to an opportunity to extend the approximate computing concepts to approximate communication, which would allow improved communication reduction through approximation techniques. This article surveyed three promising techniques (compression, relaxed synchronization, and value prediction), providing an overview of existing research on each and their applicability to approximate communication. The techniques were evaluated using a common framework composed of six parameters: communication cost reduction, performance, energy reduction, applicability, overhead, and output degradation. Finally, we presented several ideas for future research into each technique and additional techniques worth exploring for approximate communication.

The main contribution of this article was to demonstrate some of the opportunities available for tackling the communication bottleneck issues on massively parallel systems by applying approximate computing concepts to existing techniques found in the literature. The final results yielded several suggestions for future research in this area. For example, we have found that lossy link compression is a promising application of compression to approximate communication. This technique would combine the bandwidth reduction from link compression with the higher compression ratios and lower overheads from lossy compression. Likewise, approximate value prediction, when applied to massively parallel systems, would have the potential for further exploiting the scalability benefits from value prediction in parallel systems while avoiding the high costs from traditional value prediction.

ACKNOWLEDGEMENT

This work was supported in part by National Science Foundation grant no. CCF-1438286. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

REFERENCES

[1] Tor M. Aamodt and Paul Chow. 2008. Compile-time and instruction-set methods for improving floating-to fixed-point conversion accuracy. *ACM Transactions on Embedded Computing Systems* 7, 3, 26.

[2] Bülent Abali, Hubertus Franke, Dan E. Poff, Robert A. Saccone, Jr., Charles O. Schulz, Lorraine M. Herger, and T. Basil Smith. 2001. Memory expansion technology (MXT): software support and performance. *IBM Journal of Research and Development* 45, 2, 287–301.

[3] Don Adams. 1993. CRAY T3D System Architecture Overview Manual. Retrieved November 29, 2017 from ftp://ftp.cray.com/product-info/mpp/T3D_Architecture_Over/T3D.overview.html.

[4] Ismail Akturk, Karen Khatamifard, and Ulya R. Karpuzcu. 2015. On quantification of accuracy loss in approximate computing. In *Workshop on Duplicating, Deconstructing and Debunking (WDDD'15)*. 15.

[5] Alaa R. Alameldeen and David A. Wood. 2004. Adaptive cache compression for high-performance processors. In *Proceedings of the 31st Annual International Symposium on Computer Architecture*. IEEE, 212–223.

[6] Alaa R. Alameldeen and David A. Wood. 2007. Interactions between compression and prefetching in chip multiprocessors. In *IEEE 13th International Symposium on High Performance Computer Architecture (HPCA'07).* IEEE, 228–239.
[7] George Almási, Philip Heidelberger, Charles J. Archer, Xavier Martorell, C. Chris Erway, José E. Moreira, B. Steinmacher-Burow, and Yili Zheng. 2005. Optimization of MPI collective communication on BlueGene/L systems. In *Proceedings of the 19th Annual International Conference on Supercomputing (ICS'05).* ACM, New York, NY, 253–262. DOI: http://dx.doi.org/10.1145/1088149.1088183
[8] Carlos Alvarez, Jesus Corbal, and Mateo Valero. 2005. Fuzzy memoization for floating-point multimedia applications. *IEEE Transactions on Computers* 54, 7, 922–927.
[9] Gene M. Amdahl. 1967. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference.* ACM, 483–485.
[10] Baik Song An, Manhee Lee, Ki Hwan Yum, and Eun Jung Kim. 2012. Efficient data packet compression for cache coherent multiprocessor systems. In *Data Compression Conference (DCC'12).* IEEE, 129–138.
[11] Mohammad Ashraful Anam, Paul N. Whatmough, and Yiannis Andreopoulos. 2013. Precision-energy-throughput scaling of generic matrix multiplication and discrete convolution kernels via linear projections. In *IEEE 11th Symposium on Embedded Systems for Real-time Multimedia (ESTIMedia'13).* IEEE, 21–30.
[12] Jason Ansel, Cy Chan, Yee Lok Wong, Marek Olszewski, Qin Zhao, Alan Edelman, and Saman Amarasinghe. 2009. *PetaBricks: A Language and Compiler for Algorithmic Choice.* Vol. 44. ACM.
[13] Jason Ansel, Yee Lok Wong, Cy Chan, Marek Olszewski, Alan Edelman, and Saman Amarasinghe. 2011. Language and compiler support for auto-tuning variable-accuracy algorithms. In *Proceedings of the 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization.* IEEE Computer Society, 85–96.
[14] Woongki Baek and Trishul M. Chilimbi. 2010. Green: A framework for supporting energy-conscious programming using controlled approximation. In *ACM SIGPLAN Notices*, Vol. 45. ACM, 198–209.
[15] Arnab Banerjee, Pascal T. Wolkotte, Robert D. Mullins, Simon W. Moore, and Gerard J. M. Smit. 2009. An energy and performance exploration of network-on-chip architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 17, 3, 319–329.
[16] Carl J. Beckmann and Constantine D. Polychronopoulos. 1990. Fast barrier synchronization hardware. In *Proceedings of the 1990 ACM/IEEE Conference on Supercomputing (Supercomputing'90).* IEEE Computer Society, Washington, DC, USA, 180–189.
[17] K. Bergman and others. 2008. Exascale computing study: Technology challenges in achieving exascale systems. *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO)*, Tech. Rep 15 (2008).
[18] Tekin Bicer, Jian Yin, Dereck Chiu, Gagan Agrawal, and Karen Schuchardt. 2013. Integrating online compression to accelerate large-scale data analytics applications. In *IEEE 27th International Symposium on Parallel & Distributed Processing (IPDPS'13).* IEEE, 1205–1216.
[19] Mark Buckler, Wayne Burleson, and Greg Sadowski. 2013. Low-power networks-on-chip: Progress and remaining challenges. In *2013 IEEE International Symposium on Low Power Electronics and Design (ISLPED'13).* IEEE, 132–134.
[20] Huy Bui, Hal Finkel, Venkatram Vishwanath, Salman Habib, Katrin Heitmann, Jason Leigh, Michael Papka, and Kevin Harms. 2014. Scalable parallel I/O on a blue gene/Q supercomputer using compression, topology-aware data aggregation, and subfiling. In *22nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP'14).* IEEE, 107–111.
[21] Surendra Byna, Jiayuan Meng, Anand Raghunathan, Srimat Chakradhar, and Srihari Cadambi. 2010. Best-effort semantic document search on GPUs. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units.* ACM, 86–93.
[22] Brad Calder, Glenn Reinman, and Dean M. Tullsen. 1999. Selective value prediction. In *Proceedings of the 26th International Symposium on Computer Architecture.* IEEE, 64–74.
[23] Ramon Canal, Antonio González, and James E. Smith. 2000. Very low power pipelines using significance compression. In *Proceedings of the 33rd Annual ACM/IEEE International Symposium on Microarchitecture.* ACM, 181–190.
[24] Vito Cappellini. 1985. *Data Compression and Error Control Techniques with Applications.* Academic Press, Inc., Cambridge, MA.
[25] Michael Carbin, Sasa Misailovic, and Martin C. Rinard. 2013. Verifying quantitative reliability for programs that execute on unreliable hardware. In *ACM SIGPLAN Notices*, Vol. 48. ACM, 33–52.
[26] Srimat T. Chakradhar and Anand Raghunathan. 2010. Best-effort computing: Re-thinking parallel software and hardware. In *47th ACM/IEEE Design Automation Conference (DAC'10).* IEEE, 865–870.
[27] Jie Chen and W. Watson. 2008. Software barrier performance on dual quad-core opterons. *International Conference on Networking, Architecture, and Storage, 2008 (NAS'08).* 303–309.
[28] Yen-Kuang Chen, Jatin Chhugani, Pradeep Dubey, Christopher J. Hughes, Daehyun Kim, Sanjeev Kumar, Victor W. Lee, Anthony D. Nguyen, and Mikhail Smelyanskiy. 2008. Convergence of recognition, mining, and synthesis workloads and its implications. *Proceedings of IEEE* 96, 5, 790–807.

[29] Vinay K. Chippa, Hrishikesh Jayakumar, Debabrata Mohapatra, Kaushik Roy, and Anand Raghunathan. 2013. Energy-efficient recognition and mining processor using scalable effort design. In *IEEE Custom Integrated Circuits Conference (CICC'13)*. IEEE, 1–4.

[30] Vinay Kumar Chippa, Debabrata Mohapatra, Kaushik Roy, Srimat T. Chakradhar, and Anand Raghunathan. 2014. Scalable effort hardware design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22, 9, 2004–2016.

[31] Vinay K. Chippa, Swagath Venkataramani, Srimat T. Chakradhar, Kaushik Roy, and Anand Raghunathan. 2013. Approximate computing: An integrated hardware approach. In *Asilomar Conference on Signals, Systems and Computers*. IEEE, 111–117.

[32] Vinay K. Chippa, Swagath Venkataramani, Kaushik Roy, and Anand Raghunathan. 2014. StoRM: A stochastic recognition and mining processor. In *Proceedings of the 2014 International Symposium on Low Power Electronics and Design*. ACM, 39–44.

[33] Kyungsang Cho, Yongjun Lee, Young H. Oh, Gyoo-cheol Hwang, and Jae W. Lee. 2014. eDRAM-based tiered-reliability memory with applications to low-power frame buffers. In *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED'14)*. IEEE, 333–338.

[34] Marcelo Cintra and Josep Torrellas. 2002. Eliminating squashes through learning cross-thread violations in speculative parallelization for multiprocessors. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*. IEEE, 43–54.

[35] R. J. Cintra. 2011. An integer approximation method for discrete sinusoidal transforms. *Circuits, Systems, and Signal Processing* 30, 6, 1481–1501.

[36] Renato J. Cintra and Fábio M. Bayer. 2011. A DCT approximation for image compression. *IEEE Signal Processing Letters* 18, 10, 579–582.

[37] Daniel Citron and Larry Rudolph. 1995. Creating a wider bus using caching techniques. In *Proceedings of the 1st IEEE Symposium on High-Performance Computer Architecture*. IEEE, 90–99.

[38] Paul Coteus, H. Randall Bickford, Thomas M. Cipolla, Paul Crumley, Alan Gara, Shawn Hall, Gerard V. Kopcsay, Alphonso P. Lanzetta, Lawrence S. Mok, Rick A. Rand, Richard A. Swetz, Todd Takken, Paul La Rocca, Christopher Marroquin, Philip R. Germann, and Mark J. Jeanson. 2005. Packaging the blue gene/L supercomputer. *IBM Journal of Research and Development* 49, 2–3, 213–248.

[39] David E. Culler, Jaswinder Pal Singh, and Anoop Gupta. 1999. *Parallel Computer Architecture: A Hardware/Software Approach*. Gulf Professional Publishing, Houston, TX.

[40] William J. Dally and Brian Towles. 2001. Route packets, not wires: On-chip interconnection networks. In *Proceedings of the Design Automation Conference*. IEEE, 684–689.

[41] Reetuparna Das, Asit K. Mishra, Chrysostomos Nicopoulos, Dongkook Park, Vijaykrishnan Narayanan, Ravishankar Iyer, Mazin S. Yousif, and Chita R. Das. 2008. Performance and power optimization through data compression in network-on-chip architectures. In *IEEE 14th International Symposium on High Performance Computer Architecture (HPCA'08)*. IEEE, 215–225.

[42] Marc De Kruijf, Shuou Nomura, and Karthikeyan Sankaralingam. 2010. Relax: An architectural framework for software recovery of hardware faults. In *ACM SIGARCH Computer Architecture News*, Vol. 38. ACM, 497–508.

[43] Li Deng and Douglas O'Shaughnessy. 2003. *Speech Processing: A Dynamic and Optimization-Oriented Approach*. CRC Press, Boca Raton, FL.

[44] J. Dongarra, P. Luszczek, and A. Petitet. 2003. The LINPACK benchmark: Past, present, and future. *Concurrency and Computation: Practice and Experience* 15, 9, 803–820.

[45] Zidong Du, Avinash Lingamneni, Yunji Chen, Krishna Palem, Olivier Temam, and Chengyong Wu. 2014. Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators. In *19th Asia and South Pacific Design Automation Conference (ASP-DAC'14)*. IEEE, 201–206.

[46] Peter Düben, Jeremy Schlachter, Sreelatha Yenugula, John Augustine, Christian Enz, K. Palem, T. N. Palmer, and others. 2015. 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*. EDA Consortium, 764–769.

[47] Pradeep Dubey. 2005. Recognition, mining and synthesis moves computers to the era of Tera. *Technology@ Intel Magazine* 9, 2, 1–10.

[48] Peter Elias. 1955. Predictive coding–I. *IRE Transactions on Information Theory* 1, 1, 16–24.

[49] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. 2011. Dark silicon and the end of multicore scaling. In *International Symposium on Computer Architecture*.

[50] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. 2012. Architecture support for disciplined approximate programming. In *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems*.

[51] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. 2012. Neural acceleration for general-purpose approximate programs. In *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 449–460.

[52] Marius Evers, Po-Yung Chang, and Yale N. Patt. 1996. Using hybrid branch predictors to improve branch prediction accuracy in the presence of context switches. In *ACM SIGARCH Computer Architecture News*, Vol. 24. ACM, 3–11.

[53] Yuntan Fang, Huawei Li, and Xiaowei Li. 2012. SoftPCM: Enhancing energy efficiency and lifetime of phase change memory in video applications via approximate write. In *IEEE 21st Asian Test Symposium (ATS'12)*. IEEE, 131–136.

[54] Eric Freudenthal and Olivier Peze. 1988. Efficient Synchronization Algorithms Using Fetch-and-Add on Multiple Bitfield Integers. Ultracomputer Note 148.

[55] Shrikanth Ganapathy, Georgios Karakonstantis, Adam Teman, and Andreas Burg. 2015. Mitigating the impact of faults in unreliable memories for error-resilient applications. In *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 102.

[56] Bart Goeman, Hans Vandierendonck, and Koen De Bosschere. 2001. Differential FCM: Increasing value prediction accuracy by improving table usage efficiency. In *7th International Symposium on High-Performance Computer Architecture (HPCA'01)*. IEEE, 207–216.

[57] Inigo Goiri, Ricardo Bianchini, Santosh Nagarakatte, and Thu D. Nguyen. 2015. Approxhadoop: Bringing approximations to mapreduce frameworks. In *ACM SIGARCH Computer Architecture News*, Vol. 43. ACM, 383–397.

[58] Jill R. Goldschneider. 1997. *Lossy Compression of Scientific Data Via Wavelets and Vector Quantization*. Ph.D. thesis, University of Washington, Seattle, WA. https://digital.lib.washington.edu/researchworks/handle/1773/5881?show=full.

[59] Beayna Grigorian and Glenn Reinman. 2015. Accelerating divergent applications on SIMD architectures using neural networks. *ACM Transactions on Architecture and Code Optimization* 12, 1, 2.

[60] Vaibhav Gupta, Debabrata Mohapatra, Sang Phill Park, Anand Raghunathan, and Kaushik Roy. 2011. IMPACT: Imprecise adders for low-power approximate computing. In *Proceedings of the 17th IEEE/ACM International Symposium on Low-power Electronics and Design*. IEEE Press, 409–414.

[61] Erik G. Hallnor and Steven K. Reinhardt. 2004. A compressed memory hierarchy using an indirect index cache. In *Proceedings of the 3rd Workshop on Memory Performance Issues: In Conjunction with the 31st International Symposium on Computer Architecture*. ACM, 9–15.

[62] Maurice Herlihy, J. Eliot, and B. Moss. 1993. *Transactional Memory: Architectural Support for Lock-Free Data Structures*. Vol. 21. ACM.

[63] T. Hoefler, T. Mehlan, F. Mietke, and W. Rehm. 2004. A survey of barrier algorithms for coarse grained supercomputers. *Chemnitzer Informatik Berichte* 4, 3 (2004).

[64] Henry Hoffmann, Sasa Misailovic, Stelios Sidiroglou, Anant Agarwal, and Martin Rinard. 2009. Using code perforation to improve performance, reduce energy consumption, and respond to failures. Technical Report MIT-CSAIL-TR-2209-037, EECS, MIT.

[65] Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, Anant Agarwal, and Martin Rinard. 2011. Dynamic knobs for responsive power-aware computing. In *ACM SIGPLAN Notices*, Vol. 46. ACM, 199–212.

[66] Chih-Chieh Hsiao, Slo-Li Chu, and Chen-Yu Chen. 2013. Energy-aware hybrid precision selection framework for mobile GPUs. *Computers & Graphics* 37, 5, 431–444.

[67] Jiawei Huang, John Lach, and Gabriel Robins. 2012. A methodology for energy-quality tradeoff using imprecise hardware. In *Proceedings of the 49th Annual Design Automation Conference*. ACM, 504–509.

[68] Jeremy Iverson, Chandrika Kamath, and George Karypis. 2012. Fast and effective lossy compression algorithms for scientific datasets. In *Euro-Par 2012 Parallel Processing*. Springer, 843–856.

[69] Yuho Jin, Ki Hwan Yum, and Eun Jung Kim. 2008. Adaptive data compression for high-performance low-power on-chip networks. In *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 354–363.

[70] Andrew B. Kahng and Seokhyeong Kang. 2012. Accuracy-configurable adder for approximate arithmetic designs. In *Proceedings of the 49th Annual Design Automation Conference*. ACM, 820–825.

[71] Georgios Karakonstantis, Debabrata Mohapatra, and Kaushik Roy. 2012. Logic and memory design based on unequal error protection for voltage-scalable, robust and adaptive DSP systems. *Journal of Signal Processing Systems* 68, 3, 415–431.

[72] Georgios Keramidas, Chrysa Kokkala, and Iakovos Stamoulis. 2015. Clumsy value cache: An approximate memoization technique for mobile GPU fragment shaders. In *Workshop on Approximate Computing (WAPCO'15)*.

[73] Daya Shanker Khudia and Scott Mahlke. 2014. Harnessing soft computations for low-budget fault tolerance. In *47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'14)*. IEEE, 319–330.

[74] Hyungjun Kim, Pritha Ghoshal, Boris Grot, Paul V. Gratz, and Daniel A. Jiménez. 2011. Reducing network-on-chip energy consumption through spatial locality speculation. In *Proceedings of the 5th ACM/IEEE International Symposium on Networks-on-Chip*. ACM, 233–240.

[75] Chandra Krintz and Sezgin Sucu. 2006. Adaptive on-the-fly compression. *IEEE Transactions on Parallel and Distributed Systems* 17, 1, 15–24.

[76] Parag Kulkarni, Puneet Gupta, and Milos Ercegovac. 2011. Trading accuracy for power with an underdesigned multiplier architecture. In *24th International Conference on VLSI Design (VLSI Design'11)*. IEEE, 346–351.

[77] Didier Le Gall. 1991. MPEG: A video compression standard for multimedia applications. *Communications of the ACM* 34, 4, 46–58.

[78] Jae Bum Lee and Chu Shik Jhon. 1998. Reducing coherence overhead of barrier synchronization in software DSMs. In *Supercomputing'98: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing (CDROM)*. IEEE Computer Society, Washington, DC, USA, 1–18.

[79] Jang-Soo Lee, Won-Kee Hong, and Shin-Dug Kim. 1999. Design and evaluation of a selective compressed memory system. In *International Conference on Computer Design (ICCD'99)*. IEEE, 184–191.

[80] Kangmin Lee, Se-Joong Lee, and Hoi-Jun Yoo. 2006. Low-power network-on-chip for high-performance SoC design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 14, 2, 148–160.

[81] Moon-Sang Lee, Young-Jae Kang, Joon-Won Lee, and Seung-Ryoul Maeng. 2002. OPTS: Increasing branch prediction accuracy under context switch. *Microprocessors and Microsystems* 26, 6, 291–300.

[82] Sungju Lee, Heegon Kim, Yongwha Chung, and Daihee Park. 2012. Energy efficient image/video data transmission on commercial multi-core processors. *Sensors* 12, 11, 14647–14670.

[83] Sangpil Lee, Keunsoo Kim, Gunjae Koo, Hyeran Jeon, Won Woo Ro, and Murali Annavaram. 2015. Warped-compression: Enabling power efficient GPUs through register compression. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. ACM, 502–514.

[84] Larkhoon Leem, Hyungmin Cho, Jason Bau, Quinn A. Jacobson, and Subhasish Mitra. 2010. ERSA: Error resilient system architecture for probabilistic applications. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'10)*. IEEE, 1560–1565.

[85] Debra A. Lelewer and Daniel S. Hirschberg. 1987. Data compression. *ACM Computing Surveys* 19, 3, 261–296.

[86] Krisda Lengwehasatit and Antonio Ortega. 2004. Scalable variable complexity approximate forward DCT. *IEEE Transactions on Circuits and Systems for Video Technology* 14, 11, 1236–1248.

[87] Mikko H. Lipasti and John Paul Shen. 1996. Exceeding the dataflow limit via value prediction. In *Proceedings of the 29th Annual ACM/IEEE International Symposium on Microarchitecture*. IEEE Computer Society, 226–237.

[88] Mikko H. Lipasti, Christopher B. Wilkerson, and John Paul Shen. 1996. Value locality and load value prediction. *ACM SIGOPS Operating Systems Review* 30, 5, 138–147.

[89] Shaoshan Liu, Christine Eisenbeis, and Jean-Luc Gaudiot. 2010. A theoretical framework for value prediction in parallel systems. In *39th International Conference on Parallel Processing (ICPP'10)*. IEEE, 11–20.

[90] Song Liu, Karthik Pattabiraman, Thomas Moscibroda, and Benjamin G. Zorn. 2009. Flicker: Saving refresh-power in mobile devices through critical data partitioning. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'09)*.

[91] Gabriel H. Loh, Nuwan Jayasena, M. Oskin, Mark Nutter, David Roberts, Mitesh Meswani, Dong Ping Zhang, and Mike Ignatowski. 2013. A processing in memory taxonomy and a case for studying fixed-function pim. In *Workshop on Near-Data Processing (WoNDP'13)*.

[92] Enrico Magli and Gabriella Olmo. 2003. Lossy predictive coding of SAR raw data. *IEEE Transactions on Geoscience and Remote Sensing* 41, 5, 977–987.

[93] Milo M. K. Martin, Daniel J. Sorin, Harold W. Cain, Mark D. Hill, and Mikko H. Lipasti. 2001. Correctly implementing value prediction in microprocessors that support multithreading or multiprocessing. In *Proceedings of the 34th Annual ACM/IEEE International Symposium on Microarchitecture*. IEEE Computer Society, 328–337.

[94] John M. Mellor-Crummey and Michael L. Scott. 1991. Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM Transactions on Computer Systems* 9, 1, 21–65. DOI: http://dx.doi.org/10.1145/103727.103729

[95] Jiayuan Meng, Srimat Chakradhar, and Anand Raghunathan. 2009. Best-effort parallel execution framework for recognition and mining applications. In *IEEE International Symposium on Parallel & Distributed Processing (IPDPS'09)*. IEEE, 1–12.

[96] Jiayuan Mengte, Anand Raghunathan, Srimat Chakradhar, and Surendra Byna. 2010. Exploiting the forgiving nature of applications for scalable parallel execution. *IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE.

[97] Joshua San Miguel, Mario Badr, and Natalie Enright Jerger. 2014. Load value approximation. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 127–139.

[98] Sasa Misailovic, Michael Carbin, Sara Achour, Zichao Qi, and Martin C. Rinard. 2014. Chisel: Reliability-and accuracy-aware optimization of approximate computational kernels. In *ACM SIGPLAN Notices*, Vol. 49. ACM, 309–328.

[99] Sasa Misailovic, Deokhwan Kim, and Martin Rinard. 2013. Parallelizing sequential programs with statistical accuracy tests. *ACM Transactions on Embedded Computing Systems* 12, 2s, 88.

[100] Sasa Misailovic, Stelios Sidiroglou, Henry Hoffmann, and Martin Rinard. 2010. Quality of service profiling. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. ACM, 25–34.

[101] Sasa Misailovic, Stelios Sidiroglou, and Martin C. Rinard. 2012. Dancing with uncertainty. In *Proceedings of the 2012 ACM Workshop on Relaxing Synchronization for Multicore and Manycore Scalability*. ACM, 51–60.

[102] Sparsh Mittal. 2016. A survey of techniques for approximate computing. *ACM Computing Surveys* 48, 4, 62.

[103] Debabrata Mohapatra, Vinay K. Chippa, Anand Raghunathan, and Kaushik Roy. 2011. Design of voltage-scalable meta-functions for approximate computing. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'11)*. IEEE, 1–6.

[104] Debabrata Mohapatra, Georgios Karakonstantis, and Kaushik Roy. 2009. Significance driven computation: A voltage-scalable, variation-aware, quality-tuning motion estimator. In *Proceedings of the 2009 ACM/IEEE International Symposium on Low Power Electronics and Design*. ACM, 195–200.

[105] Thierry Moreau, Mark Wyse, Jacob Nelson, Adrian Sampson, Hadi Esmaeilzadeh, Luis Ceze, and Mark Oskin. 2015. SNNAP: Approximate computing on programmable socs via neural acceleration. In *IEEE 21st International Symposium on High Performance Computer Architecture (HPCA'15)*. IEEE, 603–614.

[106] Michel Mouly, Marie-Bernadette Pautet, and Thomas Foreword By-Haug. 1992. *The GSM System for Mobile Communications*. Telecom Publishing.

[107] Tarun Nakra, Rajiv Gupta, and Mary Lou Soffa. 1999. Global context-based value prediction. In *Proceedings of the 5th International Symposium on High-Performance Computer Architecture*. IEEE, 4–12.

[108] Sriram Narayanan, John Sartori, Rakesh Kumar, and Douglas L. Jones. 2010. Scalable stochastic processors. In *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 335–338.

[109] D. Nikolopoulos and T. Papatheodorou. 2000. Fast synchronization on scalable cache-coherent multiprocessors using hybrid primitives. In *Proceedings of the 14th International Symposium on Parallel and Distributed Processing (IPDPS'00)*. IEEE Computer Society, Washington, DC, USA, 711.

[110] Peter Noll. 1997. MPEG digital audio coding. *IEEE Signal Processing Magazine* 14, 5, 59–81.

[111] NVIDIA. 2014. NVIDIA GTX 980 Whitepaper. Retrieved November 29, 2017 from https://international.download.nvidia.com/geforce-com/international/pdfs/GeForce_GTX_980_Whitepaper_FINAL.PDF.

[112] Simon Ogg and Bashir Al-Hashimi. 2006. Improved data compression for serial interconnected network on chip through unused significant bit removal. In *19th International Conference on VLSI Design. Held jointly with 5th International Conference on Embedded Systems and Design*. IEEE, 5 pp.

[113] Soontorn Oraintara, Ying-Jui Chen, and Truong Q. Nguyen. 2002. Integer fast Fourier transform. *IEEE Transactions on Signal Processing* 50, 3, 607–618.

[114] David J. Palframan, Nam Sung Kim, and Mikko H. Lipasti. 2014. Precision-aware soft error protection for GPUs. In *IEEE 20th International Symposium on High Performance Computer Architecture (HPCA'14)*. IEEE, 49–59.

[115] J. Thomas Pawlowski. 2011. Hybrid memory cube (HMC). In *Hot Chips*, Vol. 23.

[116] Gennady Pekhimenko, Evgeny Bolotin, Mike O'Connor, Onur Mutlu, Todd C. Mowry, and Steve Keckler. 2015. Toggle-aware compression for GPUs. In *IEEE Computer Architecture Letters*. 14, 2 (2015), 164–168. DOI: 10.1109/LCA.2015.2430853

[117] Gennady Pekhimenko, Evgeny Bolotin, Nandita Vijaykumar, Onur Mutlu, Todd C. Mowry, and Stephen W. Keckler. 2016. A case for toggle-aware compression for GPU systems. In *IEEE International Symposium on High Performance Computer Architecture (HPCA'16)*. IEEE, 188–200.

[118] Arthur Perais and André Seznec. 2014. EOLE: Paving the way for an effective implementation of value prediction. In *ACM SIGARCH Computer Architecture News*, Vol. 42. IEEE Press, 481–492.

[119] Arthur Perais and André Seznec. 2014. Practical data value speculation for future high-end processors. In *IEEE 20th International Symposium on High Performance Computer Architecture (HPCA'14)*. IEEE, 428–439.

[120] Arthur Perais and André Seznec. 2015. BeBoP: A cost effective predictor infrastructure for superscalar value prediction. In *IEEE 21st International Symposium on High Performance Computer Architecture (HPCA'15)*. IEEE, 13–25.

[121] Calton Pu and Lenin Singaravelu. 2005. Fine-grain adaptive compression in dynamically variable networks. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*. IEEE, 685–694.

[122] Abbas Rahimi, Amirali Ghofrani, Kwang-Ting Cheng, Luca Benini, and Rajesh K. Gupta. 2015. Approximate associative memristive memory for energy-efficient GPUs. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 1497–1502.

[123] Vara Ramakrishnan and Isaac D. Scherson. 1999. Efficient techniques for nested and disjoint barrier synchronization. *Journal of Parallel and Distributed Computing* 58, 2, 333–356. DOI: http://dx.doi.org/10.1006/jpdc.1999.1556

[124] Easwaran Raman, Ram Rangan, David I. August, and others. 2008. Spice: Speculative parallel iteration chunk execution. In *Proceedings of the 6th Annual IEEE/ACM International Symposium on Code Generation and Optimization*. ACM, 175–184.

[125] Ashish Ranjan, Swagath Venkataramani, Xuanyao Fong, Kaushik Roy, and Anand Raghunathan. 2015. Approximate storage for energy efficient spintronic memories. In *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 195.

[126] Lakshminarayanan Renganarayana, Vijayalakshmi Srinivasan, Ravi Nair, and Daniel Prener. 2012. Programming with relaxed synchronization. In *Proceedings of the 2012 ACM Workshop on Relaxing Synchronization for Multicore and Manycore Scalability*. ACM, 41–50.

[127] Martin Rinard. 2013. Parallel synchronization-free approximate data structure construction. In *HotPar*.

[128] Martin C. Rinard. 2012. Unsynchronized techniques for approximate parallel computing. In *RACES Workshop*.

[129] Antonio Roldao-Lopes, Amir Shahzad, George A. Constantinides, and Eric C. Kerrigan. 2009. More flops or more precision? Accuracy parameterizable linear equation solvers for model predictive control. In *17th IEEE Symposium on Field Programmable Custom Computing Machines (FCCM'09)*. IEEE, 209–216.

[130] Cindy Rubio-González, Cuong Nguyen, Hong Diep Nguyen, James Demmel, William Kahan, Koushik Sen, David H. Bailey, Costin Iancu, and David Hough. 2013. Precimonious: Tuning assistant for floating-point precision. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, 27.

[131] David Salomon. 2004. *Data Compression: The Complete Reference*. Springer Science & Business Media, New York, NY.

[132] Mehrzad Samadi, Davoud Anoushe Jamshidi, Janghaeng Lee, and Scott Mahlke. 2014. Paraprox: Pattern-based approximation for data parallel applications. In *ACM SIGARCH Computer Architecture News*, Vol. 42. ACM, 35–50.

[133] Mehrzad Samadi, Janghaeng Lee, D. Anoushe Jamshidi, Amir Hormati, and Scott Mahlke. 2013. Sage: Self-tuning approximation for graphics engines. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 13–24.

[134] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. 2011. EnerJ: Approximate data types for safe and general low-power computation. In *ACM SIGPLAN Notices*, Vol. 46. ACM, 164–174.

[135] Adrian Sampson, Jacob Nelson, Karin Strauss, and Luis Ceze. 2014. Approximate storage in solid-state memories. *ACM Transactions on Computer Systems* 32, 3, 9.

[136] Jack Sampson, Ruben Gonzalez, Jean-Francois Collard, Norman P. Jouppi, Mike Schlansker, and Brad Calder. 2006. Exploiting fine-grained data parallelism with chip multiprocessors and fast barriers. In *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, Washington, DC, USA, 235–246. DOI:http://dx.doi.org/10.1109/MICRO.2006.23

[137] Joshua San Miguel and N. Enright Jerger. 2014. Load value approximation: Approaching the ideal memory access latency. In *Workshop on Approximate Computing Across the System Stack*.

[138] J. Sartori and R. Kumar. 2010. Low-overhead, high-speed multi-core barrier synchronization. In *HiPEAC*. 18–34.

[139] John Sartori and Rakesh Kumar. 2013. Branch and data herding: Reducing control and memory divergence for error-tolerant GPU applications. *IEEE Transactions on Multimedia* 15, 2, 279–290.

[140] Vijay Sathish, Michael J. Schulte, and Nam Sung Kim. 2012. Lossless and lossy memory I/O link compression for improving performance of GPGPU workloads. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*. ACM, 325–334.

[141] Yiannakis Sazeides and James E. Smith. 1997. The predictability of data values. In *Proceedings of the 30th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 248–258.

[142] Steven L. Scott. 1996. Synchronization and communication in the T3E multiprocessor. *SIGOPS Operating Systems Review* 30, 5, 26–36. DOI:http://dx.doi.org/10.1145/248208.237144

[143] Marko Scrbak, Mahzabeen Islam, Krishna M. Kavi, Mike Ignatowski, and Nuwan Jayasena. 2015. Processing-in-memory: Exploring the design space. In *Architecture of Computing Systems (ARCS'15)*. Springer, 43–54.

[144] André Seznec. 2011. A new case for the TAGE branch predictor. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 117–127.

[145] Ali Shafiee, Meysam Taassori, Rajeev Balasubramonian, and A. K. Davis. 2014. MemZip: Exploring unconventional benefits from memory compression. In *IEEE 20th International Symposium on High Performance Computer Architecture (HPCA'14)*. IEEE, 638–649.

[146] Li Shang, Li-Shiuan Peh, and Niraj K. Jha. 2003. Dynamic voltage scaling with links for power optimization of interconnection networks. In *Proceedings of the 9th International Symposium on High-Performance Computer Architecture (HPCA'03)*. IEEE, 91–102.

[147] Shisheng Shang and Kai Hwang. 1995. Distributed hardwired barrier synchronization for scalable multiprocessor clusters. *IEEE Transactions on Parallel Distributed Systems* 6, 6, 591–605. DOI:http://dx.doi.org/10.1109/71.388040

[148] Majid Shoushtari, Abbas BanaiyanMofrad, and Nikil Dutt. 2015. Exploiting partially-forgetful memories for approximate computing. *IEEE Embedded Systems Letters* 7, 1, 19–22.

[149] Stelios Sidiroglou-Douskos, Sasa Misailovic, Henry Hoffmann, and Martin Rinard. 2011. Managing performance vs. accuracy trade-offs with loop perforation. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*. ACM, 124–134.

[150] María Soler and José Flich. 2013. Power saving by NoC traffic compression. In *European Conference on Parallel Processing*. Springer, 465–476.

[151] Renée St. Amant, Amir Yazdanbakhsh, Jongse Park, Bradley Thwaites, Hadi Esmaeilzadeh, Arjang Hassibi, Luis Ceze, and Doug Burger. 2014. General-purpose code acceleration with limited-precision analog computation. *ACM SIGARCH Computer Architecture News* 42, 3, 505–516.

[152] J. Gregory Steffan, Christopher B. Colohan, Antonia Zhai, and Todd C. Mowry. 2002. Improving value communication for thread-level speculation. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*. IEEE, 65–75.

[153] Ayswarya Sundaram, Ameen Aakel, Derek Lockhart, Darshan Thaker, and Diana Franklin. 2008. Efficient fault tolerance in multi-media applications through selective instruction replication. In *Proceedings of the 2008 Workshop on Radiation Effects and Fault Tolerance in Nanometer Technologies.* ACM, 339–346.

[154] Mark Sutherland, Joshua San Miguel, and Natalie Enright Jerger. 2015. Texture cache approximation on GPUs. In *Workshop on Approximate Computing Across the Stack*.

[155] M. B. Taylor. 2012. Is dark silicon useful? Harnessing the four horsemen of the coming dark silicon apocalypse. In *Design Automation Conference*.

[156] Bradley Thwaites, Gennady Pekhimenko, Hadi Esmaeilzadeh, Amir Yazdanbakhsh, Onur Mutlu, Jongse Park, Girish Mururu, and Todd Mowry. 2014. Rollback-free value prediction with approximate loads. In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*. ACM, 493–494.

[157] Ye Tian, Qian Zhang, Ting Wang, Feng Yuan, and Qiang Xu. 2015. Approxma: Approximate memory access for dynamic precision scaling. In *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI.* ACM, 337–342.

[158] Vassilis Vassiliadis, Konstantinos Parasyris, Charalambos Chalios, Christos D. Antonopoulos, Spyros Lalis, Nikolaos Bellas, Hans Vandierendonck, and Dimitrios S. Nikolopoulos. 2015. A programming model and runtime system for significance-aware energy-efficient computing. In *ACM SIGPLAN Notices*, Vol. 50. ACM, 275–276.

[159] Swagath Venkataramani, Srimat T. Chakradhar, Kaushik Roy, and Anand Raghunathan. 2015. Approximate computing and the quest for computing efficiency. In *Proceedings of the 52nd Annual Design Automation Conference.* ACM, 120.

[160] Swagath Venkataramani, Vinay K. Chippa, Srimat T. Chakradhar, Kaushik Roy, and Anand Raghunathan. 2013. Quality programmable vector processors for approximate computing. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture.* ACM, 1–12.

[161] Swagath Venkataramani, Ashish Ranjan, Kaushik Roy, and Anand Raghunathan. 2014. AxNN: Energy-efficient neuromorphic systems using approximate computing. In *Proceedings of the 2014 International Symposium on Low Power Electronics and Design.* ACM, 27–32.

[162] Swagath Venkataramani, Kaushik Roy, and Anand Raghunathan. 2013. Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits. In *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 1367–1372.

[163] Nandita Vijaykumar, Gennady Pekhimenko, Adwait Jog, Abhishek Bhowmick, Rachata Ausavarungnirun, Chita Das, Mahmut Kandemir, Todd C. Mowry, and Onur Mutlu. 2015. A case for core-assisted bottleneck acceleration in GPUs: enabling flexible data compression with assist warps. In *ACM SIGARCH Computer Architecture News*, Vol. 43. ACM, 41–53.

[164] Oreste Villa, Gianluca Palermo, and Cristina Silvano. 2008. Efficiency and scalability of barrier synchronization on NoC based many-core architectures. In *CASES'08: Proceedings of the 2008 International Conference on Compilers, Architectures and Synthesis for Embedded Systems.* ACM, New York, NY, USA, 81–90. DOI: http://dx.doi.org/10.1145/1450095.1450110

[165] Gregory K. Wallace. 1992. The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics* 38, 1, xviii–xxxiv.

[166] Kai Wang and Manoj Franklin. 1997. Highly accurate data value prediction using hybrid predictors. In *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture.* IEEE Computer Society, 281–290.

[167] Zhou Wang, Eero P. Simoncelli, and Alan C. Bovik. 2003. Multiscale structural similarity for image quality assessment. In *Conference Record of the 37th Asilomar Conference on Signals, Systems and Computers*, Vol. 2. IEEE, 1398–1402.

[168] Terry A. Welch. 1984. A technique for high-performance data compression. *Computer* 6, 17, 8–19.

[169] Benjamin Welton, Dries Kimpe, Jason Cope, Christina M. Patrick, Kamil Iskra, and Robert Ross. 2011. Improving I/O forwarding throughput with data compression. In *IEEE International Conference on Cluster Computing (CLUSTER'11)*. IEEE, 438–445.

[170] Yair Wiseman, Karsten Schwan, and Patrick Widener. 2005. Efficient end to end data exchange using configurable compression. *ACM SIGOPS Operating Systems Review* 39, 3, 4–23.

[171] Qiang Xu, Todd Mytkowicz, and Nam Sung Kim. 2016. Approximate computing: A survey. *IEEE Design & Test* 33, 1, 8–22.

[172] Xin Xu and H. Howie Huang. 2015. Exploring data-level error tolerance in high-performance solid-state drives. *IEEE Transactions on Reliability* 64, 1, 15–30.

[173] Amir Yazdanbakhsh, Gennady Pekhimenko, Bradley Thwaites, Hadi Esmaeilzadeh, Onur Mutlu, and Todd C. Mowry. 2016. RFVP: Rollback-free value prediction with safe-to-approximate loads. *ACM Transactions on Architecture and Code Optimization* 12, 4, 62.

[174] Rong Ye, Ting Wang, Feng Yuan, Rakesh Kumar, and Qiang Xu. 2013. On reconfiguration-oriented approximate adder design and its application. In *Proceedings of the International Conference on Computer-Aided Design*. IEEE Press, 48–54.

[175] Yavuz Yetim, Sharad Malik, and Margaret Martonosi. 2015. CommGuard: Mitigating communication errors in error-prone parallel execution. In *ACM SIGPLAN Notices*, Vol. 50. ACM, 311–323.

[176] Yavuz Yetim, Margaret Martonosi, and Sharad Malik. 2013. Extracting useful computation from error-prone processors for streaming applications. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'13)*. IEEE, 202–207.

[177] Felix Zahn, Steffen Lammel, and Holger Fröning. 2017. Early experiences with saving energy in direct interconnection networks. In *IEEE 3rd International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB'17)*. IEEE, 33–40.

[178] Felix Zahn, Pedro Yebenes, Steffen Lammel, Pedro J. Garcia, and Holger Fröning. 2016. Analyzing the energy (dis-)proportionality of scalable interconnection networks. In *2nd IEEE International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB'16)*. IEEE, 25–32.

[179] Hang Zhang, Mateja Putic, and John Lach. 2014. Low power gpgpu computation with imprecise hardware. In *Proceedings of the 51st Annual Design Automation Conference*. ACM, 1–6.

[180] Qian Zhang, Ting Wang, Ye Tian, Feng Yuan, and Qiang Xu. 2015. ApproxANN: An approximate computing framework for artificial neural network. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 701–706.

[181] Huiyang Zhou and Thomas M. Conte. 2005. Enhancing memory-level parallelism via recovery-free value prediction. *IEEE Transactions on Computers* 54, 7, 897–912.

[182] Qiuling Zhu, Bilal Akin, H. Ekin Sumbul, Fazle Sadi, James C. Hoe, Larry Pileggi, and Franz Franchetti. 2013. A 3D-stacked logic-in-memory accelerator for application-specific data intensive computing. In *IEEE International 3D Systems Integration Conference (3DIC'13)*. IEEE, 1–7.

[183] Weirong Zhu, Vugranam C. Sreedhar, Ziang Hu, and Guang R. Gao. 2007. Synchronization state buffer: Supporting efficient fine-grain synchronization on many-core architectures. In *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA'07)*. ACM, New York, NY, USA, 35–45. DOI: http://dx.doi.org/10.1145/1250662.1250668

Received July 2016; revised July 2017; accepted September 2017