Reducing Data Movement with Approximate Computing Techniques

Stephen P. Crago
Information Sciences Institute and Department of Electrical Engineering
University of Southern California
Arlington, Virginia, USA
crago@isi.edu

Donald Yeung
Electrical and Computing Engineering Department
University of Maryland
College Park, Maryland, USA
yeung@umd.edu

***Abstract*—Data movement is the dominant factor that limits performance and efficiency in today's architectures, and we do not expect that to change in future architectures. In this paper, we describe how approximate computing techniques can be applied to communication at the algorithm level, in conventional computer architectures, and in the architectures being explored as we go beyond Moore's Law. We present results that demonstrate potential performance gains and the effect of approximations in traditional computer architectures. We describe how these techniques may be applied to future architectures based on probabilistic, approximate, stochastic, and neuromorphic computing, as well as more conventional heterogeneous and 3D architectures.**

Keywords—approximate computing, data movement, communication, neuromorphic computing, stochastic computing

I. Introduction

As the end of Moore's Law approaches, there are two factors that will drive the future of computer architectures. First, traditional silicon CMOS will no longer scale because of present limitations in power consumption and near-term limitations in the size of transistors and memory cells relative to atomic scale. Second, data movement has become the dominant factor in both performance and energy consumption today [3][10], as shown in Fig. 1, and will become more dominant over time. New technologies such as superconducting logic, quantum computing, memristors, and 3D integration are being investigated to address limitations in the scalability of silicon CMOS. However, many of these technologies and the architictures that are based on them will still be limited by the cost in both time and power for moving data. The cost of moving data is fundamentally limited by physics, and as long as application program sizes are scaled beyond the computational capacity of a small, self-contained computing device with limited communication requirements, communication (data movement) will be an issue. With the prevalance of big data and analytics, we have no reason to believe that the advent of new computing architectures and paradigms will eliminate large-scale computations that require communication between processing elements. In this paper, we propose an agenda for reducing the cost of data movement using the techniques of approximate computing.

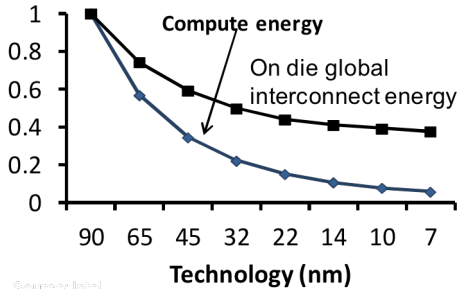

Fig. 1. Ratio of data movement to computation [10]

II. Algorithmic Opportunities

It has become well recognized that there are classes of applications that can tolerate some level of approximation. Most obviously, heuristics have been used to accelerate optimization functions for which it is infeasible to find true optimal solutions, and series expansions have been used to approximate trigonometric functions for hundreds of years. More recently lossy compression is used to reduce the number of bits used to store data for some applications, especially when the original data may be imperfect to start with, such as for camera images.

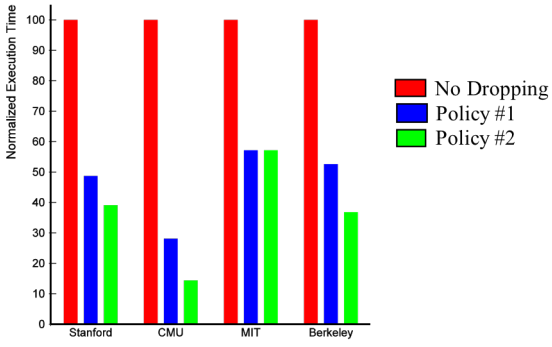

Fig. 2. Improvement in performance for loopy belief propogation when intelligent dropping messages

978-1-5090-1370-8/16/$31.00 ©2016 IEEE

The brain is known to be composed of noisy components, and while the brain often makes mistakes, there are many tasks for which it is still much better than computers, leading to the likelihood that algorithms inspired by the brain, such as neural networks and deep learning, will be tolerant of approximate computing.

As an example, we present an analysis of the effect of introducing approximations into the loopy belief propogation algorithm. The loopy belief algorithm is used for inference on large Bayesian/Markov networks for applications such as classifying web pages. The loopy belief algorithm is based on a graphical model where beliefs, or probabilities, are stored in the nodes, and edges are used to communicate dependencies between those beliefs. The algorithm is iterative, passing beliefs (*i.e.*, messages) between connected nodes until the beliefs converge. In our experiments, we found that we could randomly drop 20% of the messages between nodes to improve performance by 1.34x with very little degradation in solution quality. (Note that the messages being dropped are algorithmic and do not necessarily directly correspond to messages beween processor nodes in a computer architecture.)

Next, we experimented with dropping messages more intelligently using various heuristics. In one heuristic, we drop a message if the belief values for the nodes sending and receiving the message haven't changed significantly since the last iteration (Policy #1). In another heuristic, we drop a message if the receiving node's belief value is highly biased (Policy #2). Fig. 2 shows that Policies #1 and #2 achieve a 2.3x and 3.5x speedup compared to no message dropping, again with little change in quality of results. We believe that these examples show that approximate computing can be used to gain performance and increase efficiency by reducing data movement. While these examples are modest, we have explored

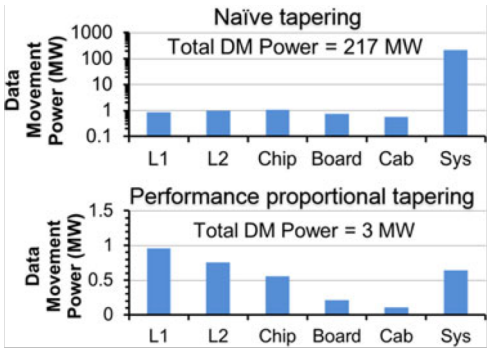

Fig. 4. Data movement power in an exascale system [1]

only a few of a wide range of techniques that are available for exploration. Furthermore, combining these algorithmic techniques with architectural techniques may lead to further gains.

III. Architectural Opportunities

A. Conventional Architectures

In order to understand the likely costs of data movement in architectures of the future, we first consider today's architctures.

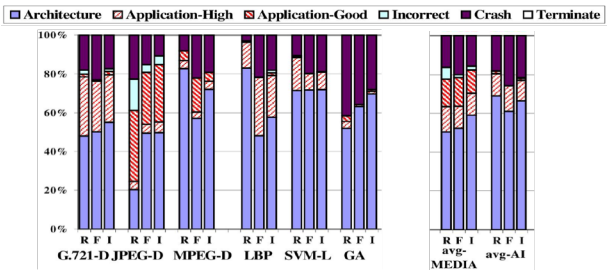

Fig. 3. Inherent fault tolerance in register file (R), fetch queue (F), and issue queue (I) [9]

Data movement can be best understood in the context of the memory hierarchy, which has been designed to keep as much data as close (in space and time) to the processing elements as possible within reasonable cost. The cost of data movement generally increases as we move down (away from the processing element) in the hierarchy, but the total energy and time consumed at that level of the hierarchy is a function of the absolute cost of data movement, the frequency of the operations at that level, and the data access patterns. The closest memory elements to the processing elements are in a register file, where data is staged for computation and directly referenced by instructions. Next, there are normally two levels of memory cache associated with a processor core, plus another level of cache on a chip that may be shared between cores. Main memory can be shared by muliple processors within a node, and data can be moved between nodes within a cabinet and between cabinets through a network interface. Typically, many or all elements of the memory hierarchy contribute signficantly to the cost of data movement, at least with respect to energy, as shown in Fig. 4, which was developed analytically for a potential exascale system under two communication scaling assumptions.

There are a variety of techniques that could be applied to reduce the cost of moving data in these architectures that can take advantage of the algorithmic opportunies discussed in the previous section. The technique closest to the algorithmic technique described in the loopy belief example is to drop messages in hardware between processing elements. The messages dropped may be between nearby processing elements within a chip, processing elements that are further away but on the same chip, between chips, between nodes within a cabinet, or between cabinets (or further). The savings of dropping messages will increase as the distance increases. If some messages can be dropped, it follows that some messages can also be delayed, leading to the possbility of combining messages to gain efficiency. Combining messages can reduce the number of bits sent by reducing packet or switching overhead or even allowing better compression on the message payloads. The allowance of dropping messages also leads to the possibility of using more efficient communication and coherency protocols. For example, if it is acceptable for some number of messages to get dropped, perhaps routers can be implemented with less costly buffering or tighter timing, or acknowledgement signals used in inter-cabinet communication can be eliminated. Other possibilities may be more akin to reducing the numerical precision of the values being communicated. Lower precision data payloads may be sent, or near-threshold logic (or the

equivalent) may be used that leads to noisy messages. While we have described these techniques in terms of messages, which are often thought of as communication between processor nodes, many or all of these techniques can be applied to data movement between any level of the memory hierarchy and processing elements or even between memory elements.

Not all stored values have similar purposes, and the effect of using these approximate techniques will be different for different kinds of values. For example, using an approximate data movement technique on a pointer could lead to an application failure if that pointer is used as a program counter destination. Fig. 3 illustrates how different kinds of data corruptions (changes in precision) can affect program behavior. The determination of which data elements can be approximated and which should be precise is application-dependent, and the software and hardware architectures that enable such determinations and optimizations based on those determinations is a rich area of research.

B. Novel Architectures

Architectures based on approximate computing, stochastic computing, and probabilistic computing are all designed with the assumption that the principals of approximate computing are applied to the architecture. Typically, such architectures focus on the computing, or processing elements. We believe this focus comes from the traditional view that computer architectures are optimized for area and logic cost, rather than to reduce data movement. However, the fact that such architectures are being designed to handle variations in precision may mean the architectures or design elements can possibly be adapted to approximation in data movement and communication as well.

Neuromorphic computing is also naturally amenable to variations in precision or approximation since the biological systems that inspire neuromorphic computing are based on unreliable components and are naturally approximate [2]. There are both digitial and analogic implementations of neuromorphic computing and a wide variety of architectures being explored. Neuromorphic architectures blur the lines between processing and memory, but data is still moving between processing elements (e.g. from one neuron to another via one or more synapses) and data movement between processing elements, especially those that are distant, are likely to be costly, both in terms of energy consumption and hardware resources. For neuromorphic architectures based on chip implementations, inter-chip communication will continue to be expensive relative to on-chip communication, especially if an architecture is scaled up to many boards or cabinets, and connectivity in neural architectures tends to be high, so finding opportunities for significant savings in data movement using approximation techniques is likely to be an important area for research.

In-memory computing reduces data movement by moving processing elements closer to where the data resides [3]. However, as was previously shown, even on-chip data movement can be expensive. Heterogeneous computing [4] and 3D chip and architecture implementations are being used to extend Moore's Law and to achieve continued performance improvements and greater efficiency from traditional CMOS silicon and more traditional architecture techniques (e.g. GPUs and FPGAs). The same arguments about scaling and data movement makes the approximate computing-based data movement techniques applicable. Design margins can be reduced when precision and reliability requirements are relaxed and communication protocol and error-checking or avoiding overheads can be used when approximations can be tolerated.

IV. Related Work

Li and Yeung [5] argue that program correctness is not black or white, but instead depends upon the user's acceptance of a program's output. They propose to interpret program correctness at the application level using application-specific fidelity metrics (as opposed to traditional strict interpretations of correctness at the architecture level which require all visible state to be perfectly correct). They conduct a fault injection campaign, and show that many multimedia and artificial intelligence workloads are resilient to faults under their more relaxed notion of application-level correctness. Although Li and Yeung focus on fault tolerance, the idea of using application-specific fidelity metrics to assess solution quality and program correctness is fundamental to all approximate computing techniques which try to tradeoff solution quality for performance or power consumption.

Many approximate computing techniques have focused on approximation at the software level. Loop perforation [6] omits iterations within loops that produce approximate results. This technique is similar to our example of dropping messages within the loopy belief propagation program. The Green system [8] provides a framework for controlling approximation. It proposes to create models that relate the performance gains and/or power savings to solution quality degradation afforded by software-based approximation techniques. Once created, these models can be used to control the degree of approximation at runtime.

In addition to approximation via software techniques, there have been many attempts at building approximate hardware as well. One promising approach is neural processing units (NPUs) [7], a specific example of the neuromorphic techniques mentioned in Section III-B. In this technique, code blocks in a program that can be approximated are identified. Then, a neural network is trained to mimic the code block's behavior. (Given a set of input values, the neural network produces an output similar to the code block, but with some error). The neural network is then implemented in hardware as an NPU, either via digital logic or analog circuits. Lastly, the code block is replaced with an instruction that invokes the computation on the NPU.

Finally, whether they are software or hardware based, the main focus of existing approximate computing techniques has been on reducing compute effort. In contrast, this paper focuses on using approximation to reduce data movement.

V. Conclusion

In this paper, we have illustrated the importance of data movement to the performance and efficiency of computer systems today, and have argued that data movement will continue to increase in importance even as computer architecture evolves or is revolutionized. We identified examples that demonstrate how data movement can be reduced at the algorithms level, and have showed how we can leverage

these algorithmic properties to reduce data movement in computer systems. We discussed how they could be used in conventional architectures today and how similar techniques could be applied as new architecures are developed to go beyond Moore's Law. Future work will determine the range of applications to which these techniques can be applied, what the programming models will be, and how new architectures will be able to reduce data movement to improve both performance and efficiency.

REFERENCES

[1] S. Borkar, "Role of interconnects in the future of computing," Journal of Lightwave Technology, Vol.. 31, No. 24, December 15, 2013.

[2] Merolla, P. A.; Arthur, J. V.; Alvarez-Icaza, R.; Cassidy, A. S.; Sawada, J.; Akopyan, F.; Jackson, B. L.; Imam, N.; Guo, C.; Nakamura, Y.; Brezzo, B.; Vo, I.; Esser, S. K.; Appuswamy, R.; Taba, B.; Amir, A.; Flickner, M. D.; Risk, W. P.; Manohar, R.; Modha, D. S. (2014). "A million spiking-neuron integrated circuit with a scalable communication network and interface". *Science* **345** (6197): 668.

[3] I. S. Choi, Y.-S. Kee, "Energy Efficient Scale-In Clusters with In-Storage Processing for Big-Data Analytics," Proceedings of the 2015 International Symposium on Memory Systems, October 2015.

[4] S. Che, J. Li, J. Lach, and K. Skadron, "Accelerating compute intensive applications with GPUs and FPGAs," Proceedings of the 6th IEEE Symposium on Application Specific Processors, June 2008.

[5] X. Li and D. Yeung, "Application level correctness and its impact on fault tolerance," Proceedings of the 13th International Symposium on High-Performance Computer Architecture, February 2007.

[6] S. Sidiroglou, S. Misailovic, H. Hoffman, and M. Rinard, "Managing performance vs. accuracy trade-offs with loop perforations," Proceedings of the ACM SIGSOFT Symposium, September 2011.

[7] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," Proceedings of the 45th International Symposium on Microarchitecture, 2012.

[8] W. Baek and T. Chilimbi, "Green: a framework for supporting energy-conscious programming using controlled approximation," Proceedings of the International Symposium on Programming Language Design and Implementation, June 2010.

[9] Xuanhua Li and Donald Yeung. Application-Level Correctness and its Impact on Fault Tolerance. In *Proceedings of the 13th International Symposium on High-Performance Computer Architecture (HPCA-XIII)*. Phoenix, AZ. February 2007.

[10] S. Borkar. How to stop interconnects from hindering the future of computing! In 2013 Optical Interconnects Conference, May 2013.