Context-Aware Adaptive Approximation

Lakshminarayana Renganarayana, Vijayalakshmi Srinivasan, Ravi Nair, Daniel Prener
IBM T.J. Watson Research Center
{lrengan, viji, nair, prener}@us.ibm.com

Abstract

We are at the threshold of an explosion in new data. Relaxation of the requirement of accurate execution provides us an opportunity to deploy low-power, low-cost technology in the processing of this vast data. In our prior work, we have shown that "Relaxed" synchronization is a useful approximation technique that promises to provide rich rewards in improved latency and reduced energy consumption without unduly compromising the quality of results. From our work on relaxed synchronization, we identify 3 factors, namely, *stability*, *redundancy*, and *feedback*, that appear to be critical to successful use of approximate computing based on the usage context.

1. Introduction

We are at the threshold of an explosion in new data, produced not only by large, powerful scientific and commercial computers, but also by the billions of low-power devices of various kinds. The traditional techniques of processing such information by first storing them in databases and then manipulating and serving them through large computers are becoming too expensive. The cost in acquiring and running such machines can be contained by recognizing that there is an exactness implied by traditional computing that is not needed in the processing of most new types of data. This relaxation of accuracy can help in the wider exploitation of relatively more energy-efficient modes of computing like cluster computing. More importantly, this relaxation of the requirement of accurate execution provides us an opportunity to deploy in the processing of this vast new data the same low-power, low-cost technology that was used to generate the data in the first place. Such energy-efficient circuits suffer from greater unreliability and variability in performance when used in the high-efficiency mode, but these problems can be addressed by changing the way we design such systems, changing the nature of the algorithms for such systems, and by modifying the expectation of the quality of results produced by such systems. We call this the "approximate computing" paradigm.

There are two sources of inaccuracies in approximate computing. The first arises from imperfect execution of an algorithm. This can be due to problems with the design of the algorithm or of the hardware, due to faults that occur after deployment of the hardware, due to the variability of operation of circuits when pushed to their design limits, or due to malicious attacks on systems. The second arises from inaccuracy in the data stream itself because of missing data or modified data, produced intentionally, as through data compression, or unintentionally, as through faulty communication channels.

One of the primary goals of approximate computing is to determine inaccuracies that are acceptable, hence requiring no rectification, because the produced results are acceptable albeit different from precise computation. In addition, for the inaccuracies that are not acceptable, the goal of approximate computing is to combat these sources of inaccuracies inexpensively and in an energy-efficient manner while producing results that may be different, yet acceptable. Computing models that achieve this goal have to address both the detection and the correction of such inaccuracies. The detection of such inaccuracies can be done either by the user observing and reacting to a wrong result as in media applications, by the algorithm expecting a range of correct results as in estimation techniques, or by the run-time monitoring of the execution of the system. The correction of system behavior can be done either by re-execution, by modification of code, or by attempting a different approximation algorithm. Future systems will need to combine all these techniques into a single dynamically optimized system that employs feedback from the user to guide the high-level choice of energy-efficient algorithms, and employs prediction based on past experience to guide the low-level energy-efficient execution of the system so as to guarantee the required precision (or approximation) desired by the usage context.

Figure 1 shows the possible opportunities for approximate computing across the system stack with suggested approaches to achieve the same. We refer the readers to [1] for a summary of prior work on approximate computing at different levels of the system stack.

In order to test our concepts on approximate computing we performed a series of experiments on relaxing synchronization in parallel applications [1]. Section 2 presents a summary of this work. Our experiments point to factors that help in the successful adoption of approximate computing in general. These are listed in Section 3.

2. "Relaxed" Synchronization

Synchronization overhead is often a major performance limiting factor in parallel applications. In our prior work [1] we proposed a methodology that can be used to reduce (and in some cases completely eliminate) synchronization overhead. The first step is to choose parallel code regions in the pro-

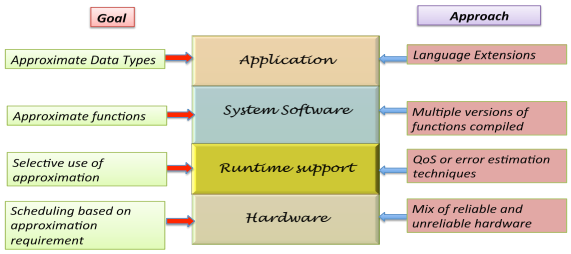

Figure 1: Approximating Computing : Opportunities across the Stack

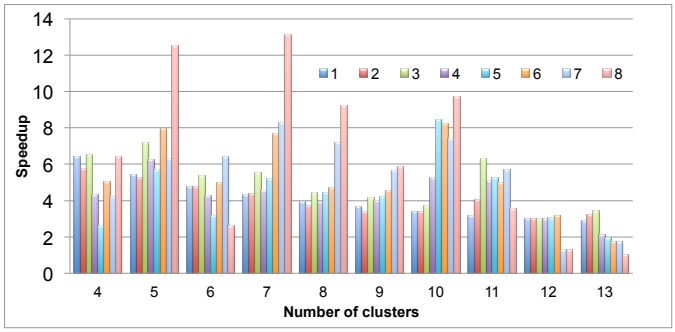

Figure 2: Speedup of relaxed versus original versions of Kmeans. The bars within a group represent the number of threads used in the parallel execution.

gram that use synchronizations. For each parallel code region, the following four key steps are needed to succeed in exploiting relaxed synchronization:

1. Choose relaxation points: Relaxing synchronization used to ensure that threads use the most recent value of a variable are excellent candidates.
2. Identify criteria that quantify the quality or acceptability of the solution: Most applications have explicit acceptability criteria. For example, the convergence criteria in an iteratively converging computation is typically an acceptability metric as well. All solutions that meet the covergence criteria are accepted as valid.
3. Restructure code to enable switching execution between the original and/or relaxed version based on the usage context.
4. Select profitable degree of relaxation via experimentation or analysis: The most profitable degree of relaxation is dependent on many factors, including the platform of execution and the input data size. Hence the choice of degree of relaxation is empirical; we have found that it is useful to restructure the code so that this degree of relaxation can be controlled via a parameter.

We have used this framework to experiment with several benchmark parallel applicaitons [1]. As an illustration we show in Figure 2 the results obtained for the *Kmeans* clustering algorithm. Approximation through relaxed synchronization on this algorithm improves performance up to 13x.

3. Lessons Learned

Relaxed synchronization, as we have demonstrated, is a useful approximation technique that promises to provide rich rewards in improved latency and reduced energy consumption without unduly compromising the quality of results. Obviously such approximation techniques cannot be employed for all problems and it would be useful to know a priori whether a certain combination of problem and solution technique would lend itself to approximate computing. From our work on relaxed synchronization, we identify 3 factors that appear to be critical to successful use of approximate computing.

- *Stability*: Slowly changing values in computation allows best exploitation of approximate computing. For example, iteratively converging algorithms tend to slowly converge to the eventual solution allowing ample opportunities to approximate values without affecting the quality of the solution.
- *Redundancy*: Approximate computing becomes useful when the potentially incorrect path to a solution can be replaced by an alternate path. For example, our relax-and-check framework falls back to the fully synchronized version if the quality of the results is not acceptable.
- *Feedback*: In any computation, a *sanity checker*, deployed either continuously or at appropriate points in the execution, provides the ability to detect potentially unacceptable results. Such checkers when deployed at a high level, e.g. at the algorithmic or at the application level, can handle divergences from expected values due to both hardware and software errors. In our relax-and-check framework, the convergence condition built into the algorithm is used as the sanity checker which, if not satisfied, prompts the algorithm to switch to the synchronized version.

Interestingly, traditional computing does not depend on any of these factors while the human brain relies heavily on all these three factors. It is clear that if we wish to solve problems that resemble those that a human brain is good at solving, and with good energy efficiency, we have to move away from traditional computing paradigms and into the realm of approximate computing.

References

[1] L. Renganarayana, V. Srinivasan, R. Nair, and D. Prener. Programming with relaxed synchronization. In *Proceedings of the 2012 ACM Workshop on Relaxing Synchronization for Multicore and Manycore Scalability*, RACES '12, pp. 41–50.