Brief Announcement: Towards Soft Optimization Techniques for Parallel Cognitive Applications

Woongki Baek, JaeWoong Chung, Chi Cao Minh, Christos Kozyrakis, and Kunle Olukotun
Computer Systems Laboratory
Stanford University
{wkbaek, jwchung, caominh, kozyraki, kunle}@stanford.edu

Categories and Subject Descriptors: D.1.3 [Programming Techniques]: Concurrent Programming – parallel programming

General Terms: Performance, Algorithms

Keywords: Optimization, Cognitive Applications, Parallel Programming, Parallel Algorithms

1. INTRODUCTION

The world's data is growing at exponential rates. To deal with this glut of information, computer systems must be able to understand and interpret data in ways that provide their users with practical knowledge [1]. To process ever increasing data sets, often, within real-time constraints, cognitive applications are stressing the performance, memory bandwidth, and storage capabilities of modern computers. Uniprocessor systems cannot meet the performance requirements even for relatively small data sets [3]. Parallel computers such as multi-core systems or larger-scale shared memory multiprocessors can provide the scalable performance necessary. In theory, cognitive tasks should map well on parallel systems as each node can process a subset of the input data or operate on a portion of the current state of knowledge. In practice, however, there are many issues that can lead to sub-optimal scalability for cognitive applications on parallel systems. Such tasks use irregular data structures such as sparse arrays and graphs which frequently lead to work imbalance, synchronization overheads, excessive communication across nodes, and poor cache locality on parallel systems.

This paper suggests soft optimization techniques for cognitive applications running on shared memory multiprocessors. Our goal is to eliminate execution inefficiencies and discover additional opportunities for higher performance. Our motivation is that cognitive applications exhibit *soft computing* properties [6]. Conventional applications, such as on-line transaction processing, operate on precise data and have strict accuracy requirements. On the other hand, cognitive applications are processing inherently noisy inputs, must handle uncertainty, and eventually produce an acceptable approximation of the "correct" answer. Hence, we may be able to skip large amount of computation and communication among concurrent threads in order to improve performance without necessarily degrading the quality of the output significantly.

We suggest the following soft properties as the most promising for performance optimizations: *user-defined correctness*, *redundancy in computation and communication patterns*, and *inherent adaptivity to errors* [6, 4]. Based on these properties, we propose four types of of general optimization techniques that target common bottlenecks in parallel systems. Specifically, they *reduce the per-thread workload*, *mitigate work imbalance*, *reduce inter-thread communication*, or *decrease synchronization overhead*. In this paper, we review the optimization techniques and summarize the major performance correctness tradeoffs. As a convincing case study, we evaluate loopy belief propagation (LBP) [5] on an aggressive parallel system. We demonstrate that that soft optimizations lead to large performance improvements on an aggressive parallel system (3.7×) without significantly impacting application accuracy.

We believe that soft computing optimizations can be a valuable resource for algorithm and application developers and should be the target for parallel programming model and system designers.

SPAA'07, June 9–11, 2007, San Diego, California, USA.
ACM 978-1-59593-667-7/07/0006.

2. SOFT OPTIMIZATIONS

Focusing on soft properties such as *user-defined correctness*, *redundancy in computation and communication patterns*, and *inherent adaptivity to errors*, we suggest the following four sets of soft optimizations:

O1) Reducing computation: The first set of optimizations exploit various types of redundancies in order to reduce the work per thread without significantly reducing the output accuracy.

Data dropping: Cognitive tasks receive superfluous incoming data but only a small portion of data carries critical or actually new information that is usually critical for accuracy.

Lazy computation: As a graph-based task converges, each node is allowed to decide independently whether it should continue to compute lazily or not based on its or its neighbor's convergence.

Solution pruning: Aggressively pruning solutions with low fitness that are unlikely to be the optimal one can significantly reduce work per thread.

O2) Mitigating imbalance: There are two major sources of imbalance in parallel programs. *Workload imbalance* occurs when we unevenly distribute work across threads. *Region imbalance* occurs when threads with further work are idling on a synchronization point (e.g., barrier), waiting for slower threads to catch up. To deal with these imbalances, we suggest the following optimizations:

Adaptive workload discarding: To deal with workload imbalance, we can make busy threads reduce their computation more aggressively than idling threads.

Selective barriers: To reduce region imbalance, we can allow threads to selectively skip some barriers. Skipping barriers in iterative algorithms may cause some threads to use slightly out of date versions of their inputs as they run ahead of other threads. However, if threads are synchronized at some reasonable frequency, the iterative nature of these algorithms allows threads to eventually operate on the newer inputs.

O3) Reducing communication: Frequent communication between threads can be very expensive, especially for large-scale parallel systems. We suggest the following technique to reduce communication overheads:

Adaptive communication: In graph-based tasks, each graph node communicates messages to its neighbors for convergence. In portions of the graph that reach convergence early, additional messages are redundant and should be dropped.

O4) Reducing synchronization: Synchronization primitives such as locks are necessary for correctness but can lead to significant performance losses as they block threads from operating in parallel. As cognitive applications have relaxed correctness requirements and can adapt to errors, we can reduce synchronization frequency using the following techniques:

Imprecise updates: In cognitive tasks, it is common to scan and test all the variables and update a global *active* working set for the next iteration. Since the number and distribution of those variables are not known in advance, static distribution or privatization may not be possible and synchronization primitives are used heavily in many cases. By applying imprecise updates for non-critical codes for accuracy, we can eliminate synchronization and significantly enhance the performance.

Removing synchronization: For cases where conflicts between threads are expected to be rare, we can remove synchronization primitives completely. In the rare case that a conflict occurs, it may cause a localized error in the application state which will be corrected over time through the iterative process.

3. CASE STUDY: LOOPY BELIEF PROPAGATION

To understand the impact of soft optimizations on application performance and correctness, we developed the following optimized versions of the parallel code for LBP, an iterative cognitive application that calculates and propagates beliefs on an unstructured graph [5].

Adaptive message version 1 (MSG1): This version reduces both communication and computation (O1 and O3). When both a sender and a receiver have converged, the sender does not calculate or send new belief messages. This technique may also mitigate imbalances because threads assigned more nodes can often drop more workloads than threads with less nodes (O2).

Adaptive message version 2 (MSG2): This version reduces both communication and computation as well (O1 and O3). When only a sender has converged, it does not create and send messages with new beliefs to its neighbors regardless of the state of each neighbor. By skipping messages more aggressively compared to version 1, we can expect further performance improvements. By the same reasoning with version 1, this technique would also mitigate imbalances (O2).

Lazy belief computation (LazyBC): This version focuses on computation reduction only (O1). A node goes into a *lazy* mode when the difference in beliefs from the previous and current iterations is within a certain threshold. Once in lazy mode, a node does not calculate new beliefs. It goes back to *busy* mode only when a new message arrives.

We evaluate the optimized versions of LBP on an simulated, aggressive, shared memory multiprocessor with hardware support for transactional memory (HTM) [2]. In Figure 1, we have shown the normalized execution time on 32 processors. Execution time is normalized to that of the base sequential code (time 1.0). Each bar is broken down into time executing useful operations, time spent for servicing cache misses, synchronization time due to imbalance, communication time (commit), and time spent on atomic blocks (transactions) that are rolled back (or violated). The base version suffers from workload imbalance due to the uneven graph partitioning. By applying the MSG1 and MSG2 techniques, useful and synchronization time are significantly reduced. This is primarily

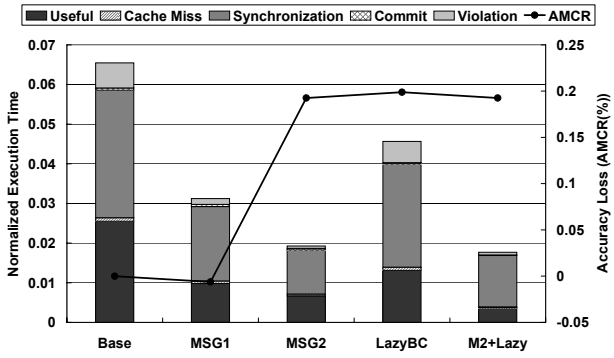

Figure 1: Normalized execution time and additional misclassification rate (AMCR) of base and optimized versions of LBP on 32 processors.

due to the fact that threads assigned with more nodes will have more opportunities to drop more messages. MSG2 shows better performance as it drops messages more aggressively compared to MSG1. LazyBC also reduces the useful computation. Compared to MSG2, LazyBC performs worse because it considers all neighbors of a node before it eliminates some work, while the adaptive messaging technique does finer grain elimination. The right-most bar represents a combined optimization (MSG2 and LazyBC). As expected, this version provides the best possible performance with an overall speedup of 56.1 on 32 processors (3.7× over a speedup of 15.3 of the base version.). We have also empirically verified that the soft optimizations lead to higher performance gains on larger scale parallel systems (e.g., 64 processors).

In Figure 1, we also present the additional miss classification rate (AMCR) of each LBP version. We use AMCR as a metric of accuracy for LBP. Qualitatively, AMCR indicates how many additional misclassifications have occurred by applying our optimizations on the classification model compared to the base version of the algorithm. Numerically, AMCR is defined as

$$\text{AMCR} = \frac{N_{misses} - N_{base\ misses}}{N_{total\ classifications}}$$

Throughout all the versions, accuracy loss is within 0.2%, which indicates the outstanding tolerance of LBP to soft optimizations. Dropping a large percentage of redundant belief calculation and messages is not critical for accuracy but leads to large performance gains. It is interesting to note that MSG1 produced higher accuracy than the original code. Probabilistic nature of cognitive applications sometimes allows soft optimization techniques to deliver both better performance and higher accuracy. The benefits presented above are robust across multiple data sets. We have omitted the detailed results due to space limitations.

4. REFERENCES

[1] S. Cass. Winner: a fountain of knowledge. *IEEE Spectr.*, 41(1):68–75, 2004.

[2] H. Chafi *et al*. A scalable, non-blocking approach to transactional memory. In *13th International Symposium on High Performance Computer Architecture (HPCA)*. Feb 2007.

[3] P. Dubey. Recognition, mining and synthesis moves computers to the era of tera. *Technology@Intel Magazine*, pages 1–10, February 2005.

[4] X. Li and D. Yeung. Exploiting soft computing for increased fault tolerance. In *Proceedings of Workshop on Architectural Support for Gigascale Integration*, June 2006.

[5] K. Murphy *et al*. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 467–47, San Francisco, CA, 1999. Morgan Kaufmann.

[6] L. A. Zadeh. Fuzzy logic, neural networks, and soft computing. *Commun. ACM*, 37(3):77–84, 1994.