From Petascale to the Pocket: Adaptively Scaling Parallel Programs for Mobile SoCs

Adam Fidel Nancy M. Amato Lawrence Rauchwerger

Parasol Laboratory
Department of Computer Science and Engineering
Texas A&M University
{fidel, amato, rwerger}@cse.tamu.edu

Categories and Subject Descriptors

Software [**Programming Techniques**]: Concurrent Programming: Parallel programming

Keywords

Energy-Efficient Computing, Parallel Libraries, Shared Memory Optimization

1. OVERVIEW

With resource-constrained mobile and embedded devices being outfitted with multicore processors, there exists a need to allow existing parallel programs to be scaled down to efficiently utilize these devices. We study the marriage of programming models originally designed for distributed-memory supercomputers with smaller scale parallel architectures that are shared-memory and generally resource-constrained.

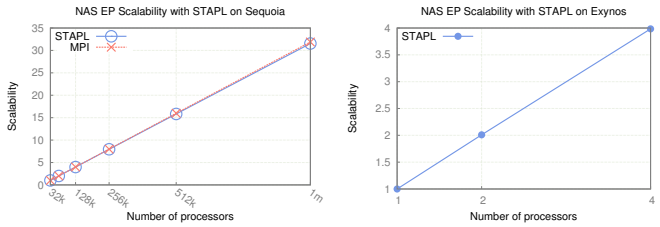
We propose techniques to allow parallel software to be aware of and adapt to power requirements of the application at the parallel algorithm level. Another level of adaptivity we propose is recognition of when costly, but otherwise necessary serialization of objects for communication can be safely avoided and to enable a zero-copy transfer without the need to rewrite the code.

We implement our techniques in STAPL [2], which is a framework for parallel C++ code development. Figure 1(a) shows scalability of a STAPL implementation of the NAS Parallel Benchmark EP on up to 1 million cores on a Blue Gene/Q, which scales near-linearly up to 1,048,576 cores and is able to match a hand-tuned MPI reference implementation. Figure 1(b) shows the same STAPL implementation of the NAS EP benchmark, unmodified, executing on a 4-core Android tablet and achieves 3.98x scalability on 4 cores.

2. ALGORITHMIC SELECTION

For fundamental parallel operations, there could exist multiple algorithms that, although functionally equivalent, have

PACT'14, August 24–27, 2014, Edmonton, AB, Canada.
ACM 978-1-4503-2809-8/14/08.
http://dx.doi.org/10.1145/2628071.2671426.



(a) (b)

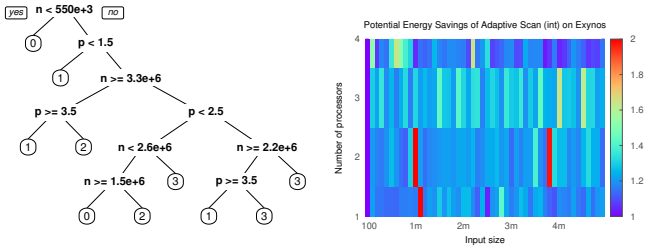
Figure 1: STAPL scalability of EP on (a) Sequoia and (b) tablet.

very different performance characteristics. Each algorithm has a distinct performance profile which is dependent on, amongst many things, the input size, processor count, datatype or if it is executed on a shared- or distributed-memory machine. It is not possible for an application developer to be completely aware of all factors that may affect the performance of the actual computation, as many of these parameters are unknown at compile-time and dependent on the environment in which the application will be eventually launched. For this reason, machine learning can be utilized to provide automated selection of the best implementation to execute which will optimize for both performance and energy consumption for a given configuration of the problem.

Thomas et al. [7] provide a framework for adaptive algorithmic selection using statistical and machine learning techniques to create a model for a given parallel operation describing the best implementation to execute based on known parameters. This model is created at install-time through the use of training: exploring the space of configurations to succinctly describe the performance behavior of a given algorithm based on a small number of samples. Once these samples are collected, learning is conducted through either decision tree learning as in this work, or others such as standard back propagation neural networks, Bayes naive classifiers or any learning technique suitable for the problem. Finally, the model is used to predict the performance of unseen configurations of the problem and choose the appropriate algorithm that best satisfies the performance requirements.

2.1 Energy Efficient Scan

In this work, we considered optimizing both performance and energy consumption of the fundamental scan (partial sum) operation using four scan algorithms: binomial tree [6], simultaneous binomial tree (pointer jumping or Hillis-



(a) (b)

Figure 2: Adaptive scan (a) classification tree and (b) potential for energy savings.

Steele) [3], a scan presented in Blelloch [1], and the standard tree-based parallel algorithm presented in JàJà [4]. We considered a tablet platform EXYNOS 4 quad-core tablet running Android 4.0.4. EXYNOS consists of an ARM Cortex-A9 clocked at 1.5 GHz with 1GB of DDR2 800Mega main memory and 1MB of L2 cache.

To build an accurate performance model and learner, we collected information of the form $E(n, p, t, a)$ that represents the energy consumed in Joules for the scan algorithm a with an input size of n, p processors and datatype t. Thus, $\max_a\{E(n, p, t, a)\}/\min_a\{E(n, p, t, a)\}$ gives the potential for energy savings (the ratio of the most energy-expensive scan to the least expensive). Figure 2(b) plots this metric for the EXYNOS platform with 32-bit integers and illustrates that there is indeed potential for energy savings for large portions of the space. Our adaptive scan algorithm uses the decision tree in Figure 2(a). We were able to achieve a 1.1x geometric mean for the $\mathcal{R}_e$ metric, where $\mathcal{R}_e(n, p, t) = \mathcal{E}(n, p, t)/\min_a\{E(n, p, t, a)\}$ and $\mathcal{E}$ is the energy used by the adaptive prefix scan algorithm. Experimentally, we found that the best scan algorithm to use varied rapidly across the space, and thus was not a concept that can easily be modeled by decision tree learning. We are looking into using more advanced learners other than decision trees. In addition, future work will evaluate the potential for adaptive algorithmic selection for performance and energy in an entire application, instead of a single algorithm.

3. SINGLE-NODE OPTIMIZATION

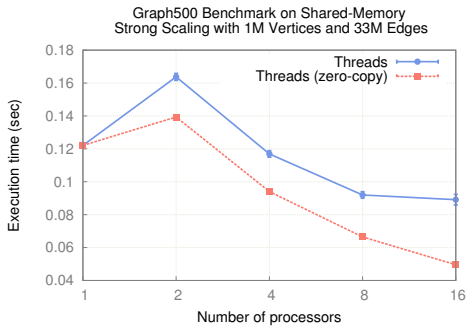


Figure 3: Experimental evaluation of zero-copy Graph 500.

STAPL's runtime system supports shared-memory remote method invocation (RMI) through the use of shared mailboxes. When a request is made from one processing element to another, the sending processor creates a buffer, copies the RMI's arguments into this buffer, and places a reference to the buffer in the destination processor's mailbox. However, in the case where the arguments are guaranteed to be immutable for the duration of the RMI, it is safe to forgo a copy of the arguments into a temporary buffer and instead pass a reference to the arguments themselves. This way, the owner-computes and atomic semantics of remote method invocation is preserved, while still saving the need to copy. Many researchers have addressed this issue at the compiler level [5], whereas this work attempts to provide a library-based solution.

To evaluate the effectiveness of a zero-copy approach, we considered a benchmark that makes heavy use of fine-grained communication. Figure 3(a) shows strong scaling for the Graph 500 benchmark (scale 20) on a shared-memory 16-core node of a Cray cluster consisting of a single-socket 16-core AMD Interlagos chip. We evaluate two versions: a standard shared-memory version and a version that employs the zero-copy optimization. Experimentally, we show that bypassing copies of immutable RMI arguments provides a benefit in raw execution time, dropping time from 89 ms to 49.5 ms at 16 cores resulting in a modest speedup of 1.79x.

Acknowledgements

This research supported in part by NSF awards CNS-0551685, CCF-0833199, CCF-0830753, IIS-0916053, IIS-0917266, EFRI-1240483, RI-1217991, by NIH NCI R25 CA090301-11, by DOE awards DE-AC02-06CH11357, B575363, by Samsung, by Award KUS-C1-016-04, made by King Abdullah University of Science and Technology (KAUST). This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

4. REFERENCES

[1] Guy E Blelloch. Prefix sums and their applications. In *Sythesis of parallel algorithms*, pages 35—60. 1990.

[2] Antal Buss, Harshvardhan, Ioannis Papadopoulos, Olga Pearce, Timmie Smith, Gabriel Tanase, Nathan Thomas, Xiabing Xu, Mauro Bianco, Nancy M. Amato, and Lawrence Rauchwerger. STAPL: Standard template adaptive parallel library. In *Proc. Annual Haifa Experimental Systems Conference (SYSTOR)*, pages 1–10, New York, NY, USA, 2010. ACM.

[3] W. Daniel Hillis and Guy L. Steele, Jr. Data parallel algorithms. *Commun. ACM*, 29(12):1170–1183, 1986.

[4] J. JàJà. *An Introduction Parallel Algorithms*. Addison–Wesley, Reading, Massachusetts, 1992.

[5] Fangzhou Jiao, N. Mahajan, J. Willcock, A. Chauhan, and A. Lumsdaine. Partial globalization of partitioned address spaces for zero-copy communication with shared memory. In *High Performance Computing (HiPC), 2011*, pages 1–10, Dec 2011.

[6] Peter Sanders and Jesper Larsson Träff. Parallel prefix (scan) algorithms for mpi. In *EuroPVM/MPI'06*, pages 49–57, Berlin, Heidelberg, 2006. Springer-Verlag.

[7] Nathan Thomas, Gabriel Tanase, Olga Tkachyshyn, Jack Perdue, Nancy M. Amato, and Lawrence Rauchwerger. A framework for adaptive algorithm selection in STAPL. In *Proc. ACM SIGPLAN Symp. Prin. Prac. Par. Prog. (PPoPP)*, pages 277–288, Chicago, IL, USA, 2005. ACM.