ALPS: Software Framework for Scheduling Parallel Computations with Application to Parallel Space-Time Adaptive Processing

Kyusoon Lee and Adam W. Bojanczyk
{kl224, awb8}@cornell.edu
School of Electrical and Computer Engineering
Cornell University, Ithaca, NY

The goal of this project is to automatically generate executable files for optimal parallel implementation of large computational tasks defined by their high-level algorithmic description. This goal is realized by a set of software tools called ALPS software framework. We present this framework in the context of a large signal processing application known as STAP.

Space-Time Adaptive Processing (STAP) refers to adaptive radar processing algorithms to cancel interferences and detect a target. Considering the required number of operations and the real-time requirements of radar processing algorithms, parallel processing is inevitable. However, good schedules for parallel implementation of such algorithms are hard to find manually. For example, FFT operations in pre-STAP processing and weight applications in STAP require access to orthogonal dimensions of data cubes. This suggests that different memory layout, or different distribution among processors of data might lead to different performances. This makes mapping and scheduling STAP algorithm on parallel computers even more challenging. ALPS framework was designed to address this difficulty.

ALPS Software Framework

The foundation of ALPS software framework is ALPS library. In [1] [2], we showed that various STAP algorithms have many tasks in common but executed in different orders, and suggested that prototyping of these algorithms can be accelerated by providing these common tasks as a library that we call ALPS library. ALPS library handles three-dimensional data cube of channel × range × PRI, and is based on LAPACK, FFTW, and MPI. It provides all the computation and communications functions required for performing STAP algorithms on parallel computers.

Within ALPS software framework, STAP algorithms are described by a Directed Acyclic Graph (DAG), G=(V,E) where V is a set of nodes indicating functions provided by ALPS library, and E is a set of edges indicating dependencies between tasks. Each function in ALPS library is benchmarked on parallel computers and modeled to predict the execution time under various configurations. Then, based on this execution time models, ALPS scheduler finds schedules that minimize the latency for a single input instance or that maximizes the throughput for continuous input instances. Finally, ALPS code generator automatically generates C++ codes to be compiled and linked with ALPS library that follow this schedule.

Algorithmic description

An example of the task graph for PRI-staggered STAP method is shown in Figure 1 (the detailed algorithmic parameters are now shown for simplicity). In this method, the data cubes are broken along PRI dimension into several smaller-sized sub-cubes (*split*). Then, after performing one-dimensional FFT along PRI dimension (*fft*), new cubes are built by taking *i*th PRI from all the sub-cubes (*recube*). For each of these smaller cubes, the optimal weights are formed and applied (*STAP*) to the target cube transformed in the same manner.

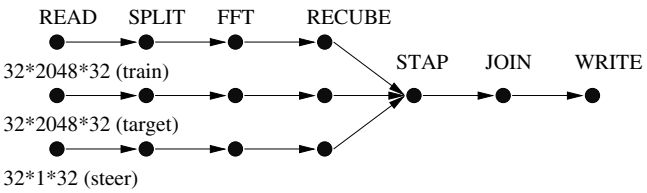

Figure 1: Task graph for PRI-staggered STAP algorithm

Benchmarking and Modeling

One possible approach to modeling the execution time of a function is to develop a theoretical model expressing the number of operations performed by the function [1]. However, developing theoretical models from implementation is not straight-forward and may not be doable manually. Moreover, the total number of operations is not always a good indicator of the execution time [4]. Instead of counting the number of operations performed by a task, we measure the actual execution time for the task for a range of input parameters characterizing the size of the task. The measurements are collected by ALPS benchmarker. From those measurements execution time models are built.

To model the execution times of tasks, we use the incremental least-squares approach. Assuming that the execution time models can be approximated by linear combination of certain terms of input parameters, we generate candidate terms. In [5] [6], we selected terms and their weights by solving the least-squares problem, $\min_{\mathbf{x}} \|A\mathbf{x} - \bar{\mathbf{y}}\|_2^2$ where the columns of A represent the candidate terms in the execution time model, and the elements of $\bar{\mathbf{y}}$ are the means of repeated measurements. The measurements are divided into training and test sets. For measurements in a training set, we select terms, one at a time, so the squared sum of the model errors is minimized. Once the model is built, it is verified on the measurements saved as a test set. We pick the model giving the minimum mean of relative errors on the test set,

$$(\text{mean of relative error}) = \frac{1}{N}\sum_{i}^{N}\frac{|\mathbf{a}_i\mathbf{x} - \bar{y}_i|}{\bar{y}_i}$$

where $\mathbf{a}_i$ is the *i*th row of A, $\bar{y}_i$ is the mean of measurements for the *i*th measurements, and N is the number of measurements in the test set.

However, the ordinary least-squares approach finds models that minimize relative errors in large measurements but often these models are inaccurate for small measurements. In order to eliminate the bias towards large measurements and further reduce the mean of relative errors, we use weighted least-squares formulation, $\min_{\mathbf{x}} \|WA\mathbf{x} - W\bar{\mathbf{y}}\|_2^2$ where *W* is a weight matrix. It turns out that the relative errors from the models generated by weighted least-squares are 20—30% smaller than those from the models generated by ordinary least-squares.

Scheduling and Code generation

Multi-processor scheduling problems are known to be NP-complete [7]. Hence, many heuristic approached have been proposed. For example, list-based algorithms [8] [9] [10] improve the latency by minimizing the length of the critical path in a task graph. These algorithms can be applied to arbitrary directed acyclic task graphs, and also guarantee certain bound on the optimality of the schedules they generate [11]. However, depending on the problem, they can be trapped in a local optimum, and they are hard to extend to the throughput maximization problem. Dynamic programming is another technique widely used in scheduling to minimize latency or maximize throughput [5] [14][15][16], and generates optimal solutions if the task graph has layered structures. [12][13] and [17] address the scheduling problem for a broader class of task graphs such as series-parallel graphs, but they heavily rely on the convexity of the computation and communication cost, which is not always the case.

ALPS scheduler extends the dynamic programming paradigm to tree-structured task graphs like the one shown in Figure 1. Our scheduling algorithm does not require the cost functions to be convex. It uses dynamic programming approach on linear task graph portion, but uses a list-based approach to schedule parallel branches of a task graph. It considers both data- and task-parallelism. As the scheduling algorithm covers much larger solution space than that of simple dynamic programming, the schedules found are as good as or often better than those computed by other published algorithms. It can be used in both latency minimization and throughput maximization. Once the schedule is found, ALPS code generator generates C++ code according to the schedule found.

We demonstrate the performance of ALPS software framework on two different linux clusters with 8-node and 16-node respectively, and compare it to the performance of other published scheduling algorithms.

References

[1] James M. Lebak, Robert C. Durie and Adam W. Bojanczyk, Toward A Portable Parallel Library for Space-Time Adaptive Methods. Technical Report CTC96TR242, Cornell University, June, 1996.

[2] James M. Lebak and Adam W. Bojanczyk, Design and Performance Evaluation of a Portable Parallel Library for Space-Time Adaptive Processing. IEEE Transactions on Parallel and Distributed Systems, Vol. 11, No. 3, March 2000.

[3] L.E. Brennan and F.M. Staudaher, Subclutter Visibility Demonstration. Technical Report RL-TR-92-21, Adaptive Sensors Incorporated, March, 1992.

[4] Matteo Frigo and Steven G. Johnson, "FFTW: An Adaptive Software Architecture for the FFT", Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, Vol. 3, 1998, pp. 1381—1384.

[5] Kyusoon Lee and Adam W. Bojanczyk, Performance Modeling and Optimization Framework for Space-Time Adaptive Processing. 3rd International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel Distributed Systems, Santa Fe, New Mexico, USA, 26-30 April 2004.

[6] Kyusoon Lee and Adam W. Bojanczyk, "ALPS: A Software Framework for Parallel Space-Time Adaptive Processing", PARA 2004, LNCS 3732, pp. 423—432, 2005.

[7] David Bernstein, Michael Rodeh, and Izidor Gertner, On the complexity of scheduling problems for parallel/pipelined machines. IEEE Transactions on Computers, 38, 9 (Sept. 1989), p. 1308—1313.

[8] Andrei Radulescu, Cristina Nicolescu, Arjan J.C. van Gemund and Pieter P. Jonker, "CPR: Mixed Task and Data Parallel Scheduling for Distributed Systems", in Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS), San Francisco, April 2001, pp. 39—47.

[9] Frederic Suter, Frederic Desprez, and Henri Casanova, "From Heterogeneous Task Scheduling to Heterogeneous mixed Parallel Scheduling", Euro-Par 2004, LNCS 3149, pp. 230—238, 2004.

[10] Savina Bansal, Padam Kumar, and Kuldip Singh, "An improved two-step algorithm for task and data parallel scheduling in distributed memory machines", Parallel Computing 32 (2006) pp. 759—774.

[11] M.R. Garey and R.L. Graham, "Bounds for Multiprocessor Scheduling with Resource Constraints", SIAM Journal of Computing, Vol. 4, No. 2, June 1975.

[12] G.N.Srinivasa Prasanna and Bruce R. Musicus, "Generalized Multiprocessor Scheduling for Directed Acyclic Graphs", Proceedings of the 1994 conference on Supercomputing.

[13] Shankar Ramaswamy, Sachin Sapatnekart and Prithviraj Benerjee, "A Framework for Exploiting Data and Functional Parallelism on Distributed Memory Multicomputers", IEEE Transactions on Parallel and Distributed Systems, November 1997, Vol. 8, No. 11, pp. 1098—1116.

[14] Jaspal Subhlok and Gary Vondran, "Optimal Use of Mixed Task and Data Parallelism for Pipelined Computations", Journal of Parallel and Distributed Computing, Vol. 60, 2000.

[15] H. Hoffman, J.V.Kepner, and R.A. Bond, "S3P: automatic, optimized mapping of signal processing applications to parallel architectures," in Proceedings of the Fifth Annual High-Performance Embedded Computing (HPEC) Workshop, Nov. 2001.

[16] Eddie Rutledge and Jeremy Kepner, "PVL: An Object Oriented Software Library for Parallel Signal Processing", IEEE Cluster 2001.

[17] David M. Nicol, Rahul Simha, and Alok N. Choudhary, "Optimal Processor Assignment for Pipeline Computations", IEEE Transactions on Parallel and Distributed Systems, Vol. 5(4), 1994.