Multi-objective Optimization of Sparse Array Computations

Una-May O'Reilly, Nadya Travinin Bliss, Sanjeev Mohindra
Julie Mullen, Eric Robinson,
unamay@csail.mit.edu
MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA 02139
{nt, smohindra, jsm, erobinson}@ll.mit.edu
MIT Lincoln Laboratory, Lexington, MA 02420

Abstract

Many knowledge extraction and decision support applications contain graph or graphical model algorithms at the core of their computation. Due to data dependency and irregularity of access, these algorithms are not well suited to parallelization and, if parallelized, do not perform well using standard data distribution techniques. Traditionally, these applications would run off-line following a data collect. With recent increases in number of sensor modalities and size of data collections, it is desirable to both develop algorithmic and computational techniques in order to provide front-end levels of performance for this application domain. Here, we present a multi-objective performance optimization algorithm that exploits the fact that graph algorithms can be cast as a collection of sparse linear algebra operations. The optimization is performed using a multi-objective genetic algorithm. The algorithm is developed within the Lincoln Laboratory code analysis and optimization framework, pMapper. Preliminary findings indicate that the multi-objective genetic algorithm conducts a localized search with a high fraction of well performing maps and thus yields better performance results.

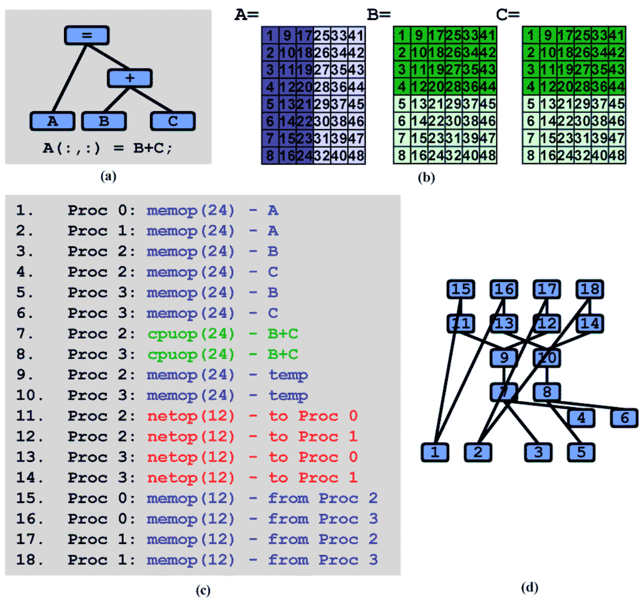
Introduction

Graphs and graphical models are key computational drivers in many decision support applications. As the amount of collected sensor data grows, a need arises to achieve front-end levels of performance for traditionally back-end applications. A key challenge in achieving the performance goals is exposing opportunities for parallelism within the algorithm.

This challenge can be addressed by recasting graph algorithm as linear algebra operations on an adjacency matrix. An adjacency matrix **A** is a representation of a graph with $\mathbf{A}(i,j) = 1$ if there is an edge between vertex i and node j and $\mathbf{A}(i,j) = 0$ otherwise. A common operation performed on a graph involves selecting a vertex or set of vertices (representing an entity or a location) and finding paths through the graph. The linear algebraic equivalent of this operation is a matrix-vector (for a single vertex case) or a matrix-matrix (for a set of vertices) multiplication on a commonly very sparse matrix. Access to a linear algebraic representation has a number of benefits: frequently significantly shorter algorithms, parallel implementations can leverage the decades of experience with parallel linear algebra, and performance issues on COTS parallel processors with standard data distribution techniques become readily apparent [1].

While the linear algebraic algorithm formulation exposes opportunities for parallelism, achieving high-performance for sparse array computations requires sophisticated code optimization. Traditional levels of performance on sparse computations on COTS platform yield <1% efficiency.

To address the performance challenge, we developed a multi-objective optimization algorithm that minimizes the complexity of the code while maximizing computational performance. The complexity of the parallel code is measured in terms of the size of dependency graph. The dependency graph is generated with the pMapper framework that includes a parametric model of the hardware, lazy evaluation code analysis, and automatic generation of low-level instructions (computation and communication). Figure 1 presents an example of a parse graph for a simple addition statement (1a), a notional set of data mappings (1b), a set of low level instructions generated from the parse graph and the maps (1c), and finally, a corresponding dependency graph (1d). The rest of the abstract presents the details of the multi-objection optimization algorithm and preliminary results.



```
1.   Proc 0: memop(24) - A
2.   Proc 1: memop(24) - A
3.   Proc 2: memop(24) - B
4.   Proc 2: memop(24) - C
5.   Proc 3: memop(24) - B
6.   Proc 3: memop(24) - C
7.   Proc 2: cpuop(24) - B+C
8.   Proc 3: cpuop(24) - B+C
9.   Proc 2: memop(24) - temp
10.  Proc 3: memop(24) - temp
11.  Proc 2: netop(12) - to Proc 0
12.  Proc 2: netop(12) - to Proc 1
13.  Proc 3: netop(12) - to Proc 0
14.  Proc 3: netop(12) - to Proc 1
15.  Proc 0: memop(12) - from Proc 2
16.  Proc 0: memop(12) - from Proc 3
17.  Proc 1: memop(12) - from Proc 2
18.  Proc 1: memop(12) - from Proc 3
```

Figure 1: The coarse-grained signal flow graph (CGSFG) (a) and maps (b) determine individual operations (c) and the dependency graph (d).

Mapping and Routing

A previously implemented nested genetic algorithm (Nested-GA) [2,3] has handled the routing and mapping of

*This work is sponsored by the Department of the Air Force under Air Force contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

sparse arrays for a matrix multiplication kernel. GAs were chosen due to their suitability to the discrete and combinatorial nature of the problem (the mapping problem in this context is NP-complete). Despite the very large search space of the problem, the Nested-GA has demonstrated finding irregular mappings that are 10-100 times better than 2D block-cyclic maps.

The objective of the Nested-GA is to find <map, route> pairs that minimize the execution latency of the algebraic kernel. The fitness function directly uses pMapper's execution latency estimation based upon the dependency graph. Because the mapping determines the dependency graph, the execution latency is a (non-linear) function of the dependency graph. This implies that some intrinsic, likely complex, properties of the dependency graph are causal predictors of execution latency. For example, for the same matrix operation, a small dependency graph can be frequently expected (but not guaranteed) to demonstrate a shorter latency than a bigger one. These observations suggest that, if a reasonably well corelated property can be identified, a subspace of the mapping search space defined by the better values of the property will have a higher frequency of better maps than average.

We have implemented a multi-objective GA (MOGA) that has the central concepts of NSGA2 [4] to test this hypothesis. The MOGA extends the Nested-GA by adding a second objective (a dependency graph property that corelates with execution latency) and selectively favoring candidate mappings that form the 'non-dominated front' of the current population for each generation. A mapping is non-dominated if there are no mappings that are better than it on ***both*** objectives. For each successive front, the mappings are sorted on each objective, and tagged with a 'crowding distance'. This metric expresses how close a mapping is to its neighbours on the front. The selection of parents for the next generation proceeds by a k-competitor tournament. Instead of the single objective determining the winner, the tournament winner is the mapping with the lowest ranked front. If more than one mapping is on the best front, the one with the largest crowding distance is the winner. This ensures a well-spaced front. Mappings at the end points of the best front are also propogated into the next generation to ensure a non-shrinking range.

Figure 2 and 3 present preliminary results for MOGA using the size of the dependency graph as the second objective. Figure 2 shows the mean dependency graph size in the population and the size of the dependency graph of the best mapping, in terms of execution latency, each generation of a run. Figure 3 shows the non-dominated front of the initial population, middle-of-run population and final generation of a MOGA run. The plot illustrates that while the smallest dependency graph does not have the shortest execution latency, the second objective influences the search to focus on better performing mappings.

Summary and Future Work

We have presented a multi-objective genetic algorithm that minimizes complexity of parallel computation (encoded by the size of the dependency graph) along with execution latency. This new multi-objective GA exploits the observation that the dependency graph is a function of the map, and, the execution latency is a function of the dependency graph. Thus, biasing the stochastic search to localized subspaces defined by a dependency graph property that is causally related to execution latency favors finding better maps. In the future we intend to investigate whether there are other, more rational, though still relatively simple, dependency graph properties that are similarly helpful. We plan to use regression analysis to further explore the dependency between properties and execution latency. While it is not likely that a generally predictive model of execution latency, based on dependency graph properties, can be learned, understanding the limits on the generality may indicate problem subclasses for which specific properties can be exploited.

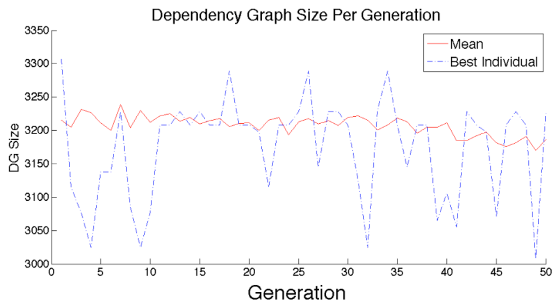


Figure 2: Dependency graph size for population mean and best individual of a MOGA run indicates a trend to smaller graphs. This is correlated with fitness improvement (see Figure 3).

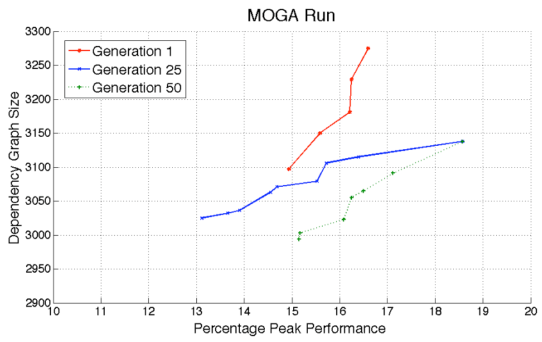


Figure 3: Non-dominated fronts of a MOGA run in generations 1, 25 and 50 demonstrate trend to smaller dependency graphs with faster maps.

References

[1] J. Kepner, N.T. Bliss, E. Robinson, "Linear Algebraic Graph Algorithms for Back End Processing", *HPEC 2008 Workshop*, Lexington, MA, September 2008.
[2] N. T. Bliss, S. Mohindra, V. Aggarwal, U.M. O'Reilly, "Analysis and Mapping of Sparse Matrix Computations" *HPEC 2007 Workshop*, Lexington, MA, September 2007.
[3] N. T. Bliss, S. Mohindra, U.M. O'Reilly, "Performance Modeling and Mapping of Sparse Computations". 2008 DoD HMCMP Users Group Conference, pp. 448-456, ISBN: 978-0-7695-3515-9, July 14-17, 2008.
[4] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, "A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II" , Proceedings of the Parallel Problem Solving from Nature VI Conference, 2000.