Using Hypergraphs and Bipartite Graphs for Run-Time Data and Computation Reordering

Michelle Mills Strout

November 3, 2006

Applications that make heavy use of sparse data structures are difficult to parallelize and reschedule for improved data locality. Examples of such applications include mesh-quality optimization, non-linear equation solvers, linear equation solvers, finite element analysis, N-body problems, and molecular dynamics simulations. Sparse data structures introduce irregular memory references (e.g., `A[B[i]]`), which inhibit compile-time, performance-improving transformations. Run-time reordering transformations dynamically reorder computation and data dynamically to improve data locality, which is a measure of how effectively the computation is able to use the cache hierarchy in a machine. Such transformations use various models to represent the relationships between computation and data, and then apply heuristics for reordering the computation and data. Guiding the selection of reordering heuristics is an open problem.

A key component of run-time reordering is the modeling of relationships between computation and data. The heuristics are then developed to solve an NP-Hard problem for the given model. The model affects the overhead of the inspector in terms of the complexity for constructing the model and applying various heuristics to the model. The model also affects which heuristics can be used for data and/or iteration reordering. The three main models that have appeared in the literature are graphs, hypergraphs, and bipartite graphs.

Figure 1 shows a loop that iterates over triangles in a mesh. Figure 2 gives an example triangular mesh and presents the graph, hypergraph, and bipartite graph models for the loop in Figure 1 assuming that its input is the triangular mesh in Figure 2.

```
for i = 1 to N
  ... A( p(i) ) ...
  ... A( q(i) ) ...
  ... A( r(i) ) ...
endfor
```

Figure 1: Example loop with irregular memory references. The loop visits `N` triangles and performs some computation on the data associated with the three nodes in each triangle.

In the graph model, each node represents a data item. Each node in the graph has an edge to other nodes that are accessed in the same iteration (ie. part of the same triangle). Many runtime reordering heuristics operate on the graph model [3, 4, 15, 1, 5, 14, 13, 8, 6, 12]. Data reordering occurs by placing the nodes in the graph in an order. Since nodes that are adjacent in the graph are accessed in the same iteration of the loop, the computation's data locality is improved if such nodes are ordered close to each other. Data reordering algorithms on the graph model heuristically solve the graph layout problem minimal linear arrangement [9], or optimal linear ordering [7]. Another approach involves heuristically partitioning the graph and then ordering the nodes by partition.

In the graph model, iterations do not have a direct correspondence to the nodes or edges unless only two items are accessed per iteration. Therefore iterations are typically reordered using variations of lexicographical sort, with each iteration having the vector of data indices being accessed as its key. As mentioned in [16], creating a dual graph where each node represents an iteration and its neighbors are other iterations that access the same data location is problematic. For example, in Figure 2 all iterations but iteration 3 would be part of a fully connected subgraph due to the fact that they all access node 5. In general, such a dual graph has worst-case size of $O(N^2)$, where N is the number of iterations in the loop. This means that heuristics that operate on a graph representation can not be directly used to reorder iterations when the graph model is used.

Figure 2 exhibits the spatial locality hypergraph and temporal locality hypergraph for the example loop and triangular mesh. The *spatial locality hypergraph* has a node for all of the data associated with one node

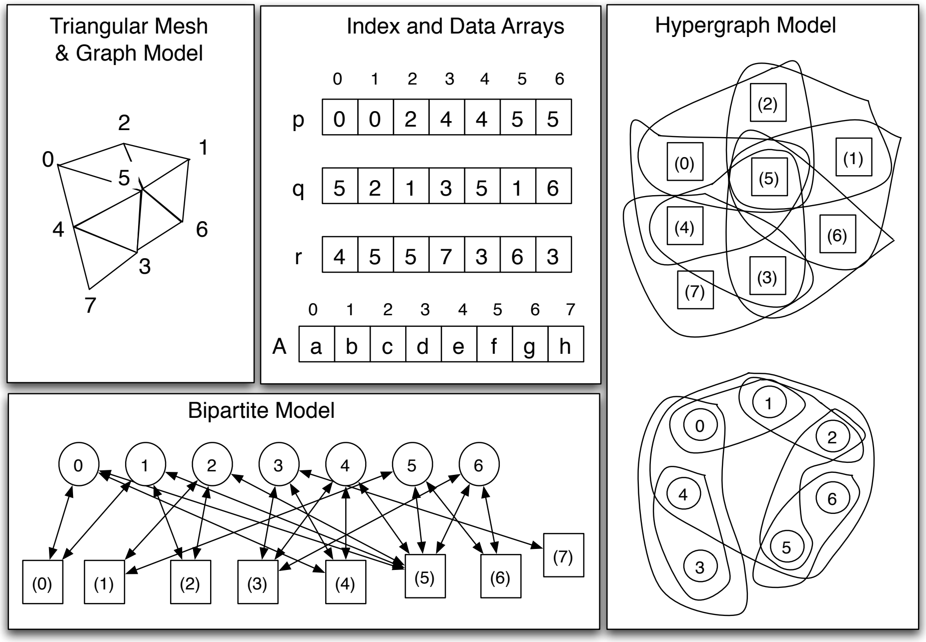

Figure 2: An example triangular mesh, its index and data arrays, hypergraph model, and bipartite model.

in the triangular mesh and a hyperedge for each iteration of the loop. Each hyperedge includes the data that will be accessed in that particular iteration of the loop. The *temporal locality hypergraph* is the dual of the spatial locality hypergraph. Each hyperedge now represents all of the data associated with one node in the original triangular mesh and each node in the hypergraph represents an iteration of the loop. Building these hypergraphs is not expensive, and it is possible to apply more heuristics to both data and iteration reordering using the hypergraph model. In previous work [16, 17], we show results where reordering heuristics based on hypergraphs result in better performance.

The bipartite graph model we present contains a set of nodes for the data items in the loop and a set of nodes for the iterations in the loop. The bipartite graph in Figure 2 uses circular nodes for the iterations and square nodes for data. If a data item is accessed by an iteration there is a bidirectional edge between the two. The space and time needed to construct the bipartite graph is equivalent to constructing both the spatial and temporal locality hypergraphs. The goal is to apply current heuristics based on the graph model to the bipartite model for simultaneous data and iteration reordering.

Hypergraph and bipartite models have been proposed and used for reducing the parallel communication volume [10, 2] and for improving data locality [11]. The most prevalent example for both models is determining separate partitionings and/or reorderings for the rows and columns in a sparse matrix used in the context of a computation like sparse matrix-vector multiply. For sparse matrix-vector computation, we envision the rows and columns still resulting in the same reordering, but we consider the possibility of reordering the iterations or the order in which the nonzeros in the matrix are visited.

This talk covers the various models of computation to data mappings: graphs, hypergraphs, and bipartite graphs in terms of the time and space cost for constructing them, the time and space cost for applying various reordering heuristics to each of the models, and predications about the effectiveness of the heuristics. Such an analysis indicates that graphs are more expensive to generate and likely to result in less effective reorderings. Hypergraph and bipartite graph representations incur the same cost for construction, but more research is needed to determine which model is more effective.

References

[1] I. Al-Furaih and S. Ranka. Memory hierarchy management for iterative graph structures. In *Proceedings of the 1st Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing*, pages 298–302, March 30–April 3, 1998.

[2] Cevdet Aykanat, Ali Pinar, and Umit V. Catalyurek. Permuting sparse rectangular matrices into block-diagonal form. *SIAM Journal of Scientific Computing*, 25(6):1860–1879, 2004.

[3] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 24th National Conference ACM*, pages 157–172, 1969.

[4] R. Das, D. Mavriplis, J. Saltz, S. Gupta, and R. Ponnusamy. The design and implementation of a parallel unstructured euler solver using software primitives. *AIAA Journal*, 32:489–496, March 1992.

[5] C. Ding and K. Kennedy. Improving cache performance in dynamic applications through data and computation reorganization at run time. In *Proceedings of the 1999 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 229–241, May 1–4, 1999.

[6] Jinghua Fu, Alex Pothen, Dimitri Mavriplis, and Shengnian Ye. On the memory system performance of sparse algorithms. In *Eighth International Workshop on Solving Irregularly Struct ured Problems in Parallel*, 2001.

[7] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[8] H. Han and C. Tseng. A comparison of locality transformations for irregular codes. In *Proceedings of the 5th International Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers*, volume 1915 of *Lecture Notes in Computer Science*. Springer, 2000.

[9] L. H. Harper. Optimal assignments of numbers to vertices. *SIAM Journal*, 12(1):131–135, 1964.

[10] Bruce Hendrickson and Tamara G. Kolda. Graph partitioning models for parallel computing. *Parallel Computing*, 26(12):1519–1534, 2000.

[11] E. Im and K. Yelick. Optimizing sparse matrix-vector multiplication on SMPs. In *Ninth SIAM Conference on Parallel Processing for Scientific Computing*. SIAM, March 1999.

[12] E. Im and K. Yelick. Optimizing sparse matrix computations for register reuse in sparsity. In V. N. Alexandrov, J. J. Dongarra, B. A. Juliano, R. S.Renner, and C. J. K. Tan, editors, *Computational Science - ICCS 2001*, volume 2073 of *Lecture Notes in Computer Science*, pages 127–136. Springer, May 28-30, 2001.

[13] J. Mellor-Crummey, D. Whalley, and K. Kennedy. Improving memory hierarchy performance for irregular applications. In *Proceedings of the 1999 ACM SIGARCH International Conference on Supercomputing (ICS)*, pages 425–433, June 20–25 1999.

[14] N. Mitchell, L. Carter, and J. Ferrante. Localizing non-affine array references. In *Proceedings of the 1999 International Conference on Parallel Architectures and Compilation Techniques*, pages 192–202, October 12–16, 1999.

[15] J. P. Singh, C. Holt, T. Totsuka, A. Gupta, and J. Hennessy. Load balancing and data locality in adaptive hierarchical N-body methods: Barnes-Hut, fast multipole, and radiosity. *Journal of Parallel and Distributed Computing*, 27(2):118–141, June 1995.

[16] Michelle Mills Strout and Paul Hovland. Using hypergraphs to improve iteration reordering heuristics. Presented at the SIAM Workshop on Combinatorial Scientific Computing, February 2004.

[17] Michelle Mills Strout and Paul D. Hovland. Metrics and models for reordering transformations. In *Proceedings of the The Second ACM SIGPLAN Workshop on Memory System Performance (MSP)*, pages 23–34, June 8 2004.