Using Hypergraphs to Improve Iteration Reordering Heuristics

Michelle Mills Strout and Paul D. Hovland

October 31, 2003

Irregular applications exhibit poor performance on current computer architectures because of their inefficient use of the memory hierarchy. Figure 1 shows iteration over an edge list as an example of the types of memory references that occur in irregular applications. Run-time data and iteration reordering transformations have been shown to improve the locality of such loops and therefore the performance of the irregular benchmarks [2, 1, 3, 10, 8, 6, 4]. Data reordering algorithms heuristically solve the graph layout problem minimal linear arrangement [7], or optimal linear ordering [5], based on the graph with one vertex for each data item and edges connecting data accessed within the same iteration (see the spatial locality graph in Figure 1). Reordering the edges in this graph reorders the computation or iterations of the loop. Typically edges are ordered as an afterthought based on some variant of the lexicographical ordering enforced by the data. We propose that the data and iteration reordering transformations should be viewed as approximating minimal linear arrangments on two separate hypergraphs. We posit that heuristics aimed at solving this problem on the temporal locality hypergraph (the dual of the spatial locality hypergraph) will result in better performance. Our preliminary results show that there is some positive correlation between the minimal linear arrangement metric of the spatial and temporal locality hypergraphs and previous performance results. One simple new heuristic based on the temporal locality hypergraph shows improvement with the linear arrangement metric, and we plan to conduct experiments to verify that using this new heuristic to reorder iterations also improves performance.

Application performance depends on efficient memory hierarchy usage. In almost all modern computers, whenever a memory location is referenced by a program, the data in the referenced location and nearby locations are brought into a fast but small data cache. Any additional references to data already in the

```
for i=1 to N
  ... X[l[i]] ...
  ... X[r[i]] ...
endfor
```

	i=1	2	3	4	5	6
l	B	D	A	B	D	B
r	F	E	C	C	F	D

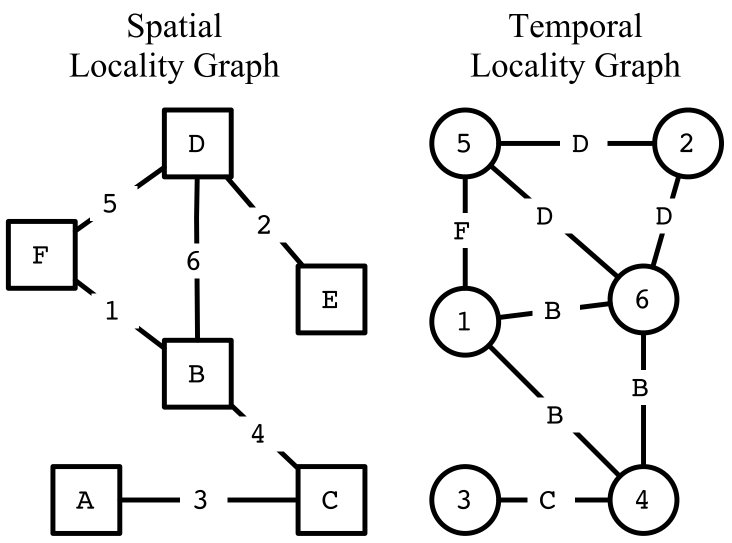

Figure 1: Example loop with two memory references. Given the shown values in the l and r index arrays, the corresponding spatial and temporal locality graphs are shown.

cache-line (before the cache-line is evicted from the cache) will be one or two orders of magnitude faster then references to main memory. When such usage occurs during the execution of a program it is referred to as *spatial locality* for reuse within a cache-line and *temporal locality* for reuse of the same element in a cache-line.

The locality of a program can be improved by changing the order of computation (iteration reordering) and/or the assignment of data to memory locations (data reordering) so that references to the same or nearby locations occur relatively close in time during the execution of the program. Run-time reordering transformations use *inspector/executor* strategies [9] to effectively reorder irregular applications. An inspector traverses the memory reference pattern (eg. edges in the edge list example) at runtime, generates data-reordering and iteration-reordering functions based on the observed pattern, creates new schedules, and remaps affected data structures accordingly. The executor is a transformed version of the original program that uses the schedules and remapped data structures generated by the inspector.

In general, run-time *data-reordering* transformations improve the spatial locality in a computation. If each data item v is mapped to storage location $\sigma(v)$, then the spatial locality metric based on minimal linear arrangement is $\sum_{(v,w)\in G_{SL}(E)} |\sigma(v) - \sigma(w)|$, where $G_{SL}(E)$ is the set of edges in the spatial locality graph. Some common heuristics for determining the data reordering function σ are Reverse Cuthill-McKee (RCM) [2], consecutive packing (CPACK) [3], and graph partitioning heuristics [6]. Such heuristics operate on the spatial locality graph.

Temporal locality occurs when the *same* memory location is reused before its cache-line is evicted. In general, run-time *iteration reordering* improves the temporal and spatial locality within an irregular application. Using the concept of a temporal locality graph (see Figure 1), the minimal linear arrangement metric can also be used to measure the effectiveness of an iteration reordering. Specifically, if each iteration i is mapped to a relative time $\delta(i)$, then the temporal locality metric is $\sum_{(i,j)\in G_{TL}(E)} |\delta(i) - \delta(j)|$, where $G_{TL}(E)$ is the set of edges in the temporal locality graph. However, current iteration reordering heuristics are based on a lexicographical ordering of the edges in the spatial locality graph. We argue that heuristics for iteration reordering should instead be based on the structure of the temporal locality graph.

One argument against using temporal locality graphs is that they can be expensive to generate. Notice that in Figure 1 many edges are introduced because data elements `D` and `B` are accessed by a number of iterations in the loop. Due to this, we suggest that hypergraphs would more generally and efficiently represent both the temporal and spatial locality information. Hypergraphs are also desirable because the spatial and temporal locality hypergraphs are duals and therefore it is possible to efficiently translate between them (ie. equivalent to switching between compressed sparse row representation and compressed sparse column).

The spatial/temporal locality metric can easily be extended to hypergraphs by summing the distances between all data/iterations connected by the same hyperedge across all hyperedges. We found that there is some positive correlation between the performance data reported by Strout *et al.* [11] and such spatial and temporal locality metrics. We also found that a simple heuristic, consecutive packing for iteration reordering, based on the temporal locality hypergraph resulted in a better temporal locality metric than the lexicographical sorting based heuristics. Consecutive packing for iteration reordering packs each iteration into a new order upon visiting the ordered hyperedges of the temporal locality hypergraph. Each hyperedge is associated with a data element, so the current data order induces an ordering on the hyperedges. Based on our preliminary findings, it is probable that this heuristic will result in overhead similar to lexicographical sorting based reorderings and improved executor performance, therefore heuristics based on the temporal hypergraph representation should be investigated further.

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

[3] 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.

[4] 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.

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

[6] 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.

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

[8] 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.

[9] Ravi Mirchandaney, Joel H. Saltz, Roger M. Smith, David M. Nicol, and Kay Crowley. Principles of runtime support for parallel processors. In *Proceedings of the 1988 ACM International Conference on Supercomputing (ICS)*, pages 140–152, July 1988.

[10] 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.

[11] Michelle Mills Strout, Larry Carter, and Jeanne Ferrante. Compile-time composition of run-time data and iteration reorderings. In *Proceedings of the 2003 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, June 2003.