CMSC 714
Lecture 17
Cache Tools

Alan Sussman

Notes

- Midterm exam Thursday, April 27
  - on readings through next Thursday
- Group Project interim report due April 21

Data and Computation Reordering

- Goal is to improve performance of *irregular* applications
  - ones with data access patterns not known until runtime
  - includes solving PDEs on unstructured or adaptive grids, n-body problems, etc.
  - in this paper, model the access pattern with an *interaction list* that specifies the data elements to access
- Runtime methods to do the same types of optimizations as are done for regular applications
  - ones where data access patterns (often to multi-dimensional arrays) are known at compile-time
  - e.g., loop blocking, interchange, data prefetching
- Methods to reorder data dynamically to improve memory hierarchy behavior
  - improve spatial locality
- Methods to reorder loop iterations
  - typically to improve spatial and temporal locality

Data and Computation Reordering

- Data reordering – reorder the data elements pointed to by the interaction list (since order really doesn't matter for getting the right answers) – to improve spatial locality
  - first touch method
    - first do a linear scan to determine order elements are accessed, then sort in that order, updating the interaction list to point to the relocated elements
  - space filling curve method
    - use element coordinate information to build a space filling curve that goes through all the elements, which preserves locality in multiple dimensions
    - sort elements according to position on the curve
- Computation reordering – reorder loop iterations, but don't change the locations of the data elements – to improve both spatial and temporal locality
  - space filling curve method
    - use positions of data elements as coordinates for the space filling curve
  - blocking method
    - recursive divide and conquer method to group elements – partition the overall coordinate space, and process elements one partition at a time
- Overall experimental results on 3 applications/kernels show that need to do both computation and data reordering to get best results, and should use space filling curves for both

MemSpy

- A tool for finding memory performance bottlenecks in serial and parallel programs
  - provides detailed view of cache misses
  - and both code- and data-centric views of the causes for cache misses
- Goals are to
  - separately report processor and memory time, to find memory bottlenecks
  - link bottlenecks back to data objects, not just code segments
  - provide memory stats detailed enough to enable programmer to fix bottlenecks
    - why did the cache misses occur?
- High overhead solution
  - use simulation to track cache behavior (no hardware support required)
  - uses Tango simulation/tracing system
    - instrument application via pre-processing, then trace every memory reference with a call to the memory simulator, which then calls MemSpy to compute aggregate statistics on cache events (hits, misses, replacements, etc.)

MemSpy

- Presents code and data oriented statistics
  - code and data divided into logical units – *code segments* and *data bins* – group statistics into each bin
  - code segment is a function/procedure – just need to trace function entry/exit
  - data bin can be a single object, or a group of objects
    - a bin is all memory ranges allocated at same point in source code with identical call paths (same stack)
- Data oriented statistics divided into 3 categories
  - compulsory misses (first use)
  - replacements (capacity misses, conflict misses)
  - invalidations (from cache coherence misses in an SMP)
- Code examples show the utility of data centric view, and breaking down misses into categories
- Performance of instrumented code is very poor, but claim is that it could be improved (never done?)
  - real problem is that multiprocessor execution is simulated by Tango via interleaving processes on a single processor, so does not scale
  - conclusion is that need hardware trace facility on a multiprocessor