Parallel Programming Performance Hinges on Memory Access Efficiency

Ganesh Gopalakrishnan, School of Computing, University of Utah, Salt Lake City

Main Idea: Many parallel programming and multicore books start with the mantra of "core bounty," potentially creating false expectations. This assignment is designed to stress early (and often) that (a) cache access efficiency is paramount, and (b) false sharing is almost too easy to introduce. The first exercise is a straightforward row-major (versus column-major) access example: the student can time this calculation for different array sizes. The second (false sharing) is a simple PThreads example (inspired by Mattson's talks) whose worker thread code is shown. Basically, the user can start this thread function in many threads and keep increasing the constant `Pad`. As the separation between the words accessed increases (by increasing `Pad`), fewer threads suffer from false-sharing induced misses, and the speed goes up. When `Pad` rises beyond a cache-line size, the performance will significantly rise.

```
// Bad (Column-major) vs. Good (Row-major) Cache Access Patterns
   for (int i=0; i<16384; ++i)
    for (int j=0; j<16384; ++j)
      A[j][i]++;  // Change this to A[i][j]++

// False Sharing shown by this thread worker function
void *Thr_fn(void *rank) {
  int my_rank = (long) rank;
  for (long i=0; i<1024*1048576; ++i) { // 2^20
    A[my_rank*Pad]++;
  }
  return NULL;
}
```

Concepts Covered: (1) Cache organization (good to mention that this varies between C and Fortran); (2) fetches are cache-line sized, and cache coherence protocols manage the cache at a cache-line granularity.

Targeted Students: Any student taking parallel computing or an earlier class must receive multiple exposures to this basic topic.

In what contexts used before: This was assigned to an UG class I am teaching now (10 students). In hindsight, I was glad that I introduced this example early in the course.

What prerequisites assumed: Students only need a basic grounding in C and Pthreads programming, and basic cache organization.

Strengths/Weaknesses: It illuminates the dominant role of memory access efficiency. It does not cover cache tiling (but it could be an advanced exercise).

Variations of Interest: One can ask students to sieve for Primes in separate threads, and then have the threads record the primes found into a central data structure. Potential solutions can be these, with the following caveats: (1) Can we use a `std::vector<bool>`? Answer: No, as this is not thread-safe (races). (2) Can we use a `std::vector<int>`? Answer: Yes, as this is thread-safe, but it can suffer from false sharing. This will lead them to understand concurrent data structures that are safe and more efficient. Contrast between OpenMP accesses (adjacent threads must not access adjacent locations) and GPU accesses (adjacent threads *must* access adjacent locations) can be illuminating.