ECE459: Programming for Performance Winter 2011

Lecture 22 — March 29, 2011

Patrick Lam *version 1*

This week, we'll talk about techniques for sacrificing accuracy in favour of performance; special-purpose languages for high-performance computing; and (on Thursday) we'll summarize the course.

Reduced-resource computation

In Assignment 4, you are manually implementing an optimization that trades off accuracy for performance. In that specific case, I outlined how some domain knowledge could enable you to skimp out on some unnecessary computation: points that are far away contribute only very small forces.

In [RHMS10], Martin Rinard summarizes two of his novel ideas for automatic or semiautomatic optimizations which trade accuracy for performance: early phase termination [Rin07] and loop perforation [HMS+09]. Both of these ideas are applicable to code we've learned about in this class.

Early phase termination

We've talked about barriers quite a bit. Recall that the idea is that no thread may proceed past a barrier until all of the threads reach the barrier. Waiting for other threads causes delays. Killing slow threads obviously speeds up the program. Well, that's easy.

"Oh no, that's going to change the meaning of the program!"

Let's consider some arguments about when it may be acceptable to just kill (discard) tasks. Since we're not completely crazy, we can develop a statistical model of the program behaviour, and make sure that the tasks we kill don't introduce unacceptable distortions. Then when we run the program, we get an output and a confidence interval.

Two Examples. When might this work? Monte Carlo simulations are a good candidate; you're already picking points randomly. Raytracers can work as well. Both of these examples could spawn a lot of threads and wait for all threads to complete. In either case, you can compensate for missing data points, assuming that they look like the ones that you did compute.

Also recall that, in scientific computations, you're entering points that were measured (with some error) and that you're computing using machine numbers (also with some error). Computers are only providing simulations, not the ground truth; the question is whether the simulation is good enough.

Loop perforation

You can also apply the same idea to sequential programs. Instead of discarding tasks, the idea here is to discard loop iterations. Here's a simple example: instead of the loop,

```
for (i = 0; i < n; i++) sum += numbers[i];
```

simply write,

```
for (i = 0; i < n; i += 2) sum += numbers[i];
```

and multiply the end result by a factor of 2. This only works if the inputs are appropriately distributed, but it does give a factor 2 speedup.

Example domains. In [RHMS10], we can read that loop perforation works for evaluating forces on water molecules (in particular, summing numbers); Monte-Carlo simulation for swaption pricing; and video encoding. In that example, changing loop increments from 4 to 8 gives a speedup of 1.67, a signal to noise ratio decrease of 0.87%, and a bitrate increase of 18.47%, producing visually indistinguishable results. The computation looks like this:

```
min = DBL_MAX;
index = 0;
for (i = 0; i < m; i++) {
  sum = 0;
  for (j = 0; j < n; j++) sum += numbers[i][j];
  if (min < sum) {
    min = sum;
    index = i;
  }
}
```

The optimization changes the loop increments and then compensates.

References

[HMS+09] Henry Hoffmann, Sasa Misailovic, Stelios Sidiroglou, Anant Agarwal, and Martin Rinard. Using code perforation to improve performance, reduce energy consumption, and respond to failures. Technical Report MIT-CSAIL-TR-2009-042, MIT CSAIL, Cambridge, MA, September 2009.

[RHMS10] Martin Rinard, Henry Hoffmann, Sasa Misailovic, and Stelios Sidiroglou. Patterns and statistical analysis for understanding reduced resource computing. In *Proceedings of Onward! 2010*, pages 806–821, Reno/Tahoe, NV, USA, October 2010. ACM.

[Rin07] Martin Rinard. Using early phase termination to eliminate load imbalances at barrier synchronization points. In *Proceedings of OOPSLA 2007*, pages 369–386, Montreal, Quebec, Canada, October 2007.