# Lecture 17 — Early Termination, Reduced-Resource Computation

Patrick Lam & Jeff Zarnett
patrick.lam@uwaterloo.ca, jzarnett@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

October 31, 2019

# The N-Body Simulation

A common physics problem that programmers are asked to simulate is the N-Body problem.

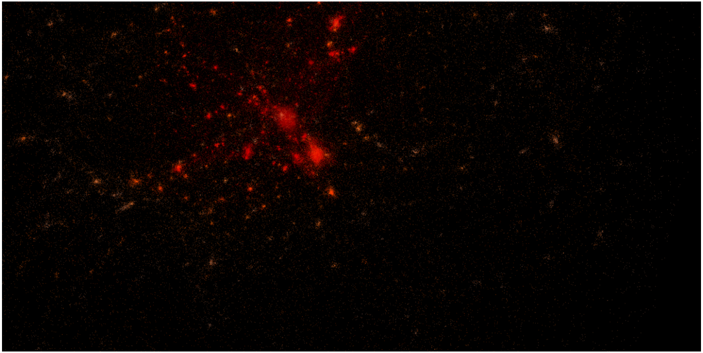

Image Credit: Michael L. Umbricht

# The N-Body Simulation

Let's assume it's OpenCL converted and is optimized.

Can we use `float` instead of `double`?

What if we want more?

# Estimation is Okay

Points that are far away contribute only very small forces.

So you can estimate them (crudely).

The idea is to divide the points into a number of “bins” which are cubes representing a locale of some sort.

Then, compute the centre of mass for each bin.

When calculating the forces: centre of mass for faraway bins; individual particles for nearby particles.

# This used to be an assignment...

A more concrete explanation with an example: suppose the space is divided into $[0, 1000]^3$, so we can take bins which are cubes of length 100.

This gives 1000 bins.

To increase the accuracy, what should we do?

To increase the speed, what should we do?

# Compute Centre of Mass

Compute all of the masses in parallel: create one thread per bin, and add a point's position if it belongs to the bin:

```
int xbin, ybin, zbin; // initialize with bin coordinates
int b;
for (i = 0; i < POINTS; i++) {
    if (pts[i] in bin coordinates) {
        cm[b].x += pts[i].x; // y, z too
        cm[b].w += 1.0f;
    }
}
cm[b].x /= cm[b].w; // etc
```

Note that this parallelizes with the number of bins.

# Map Points

For the next step, the program needs to keep track of the points in each bin.

In a second phase, iterate over all bins again, this time putting coordinates into the proper element of `binPts`, a two-dimensional array.

# Save Time

The payoff from all these calculations is to save time while calculating forces.

In this example, we'll compute exact forces for the points in the same bin and the directly-adjacent bins in each direction

That makes 27 bins in all, with 6 bins sharing a square, 12 bins sharing an edge, and 8 bins sharing a point with the centre bin).

If there is no adjacent bin (i.e., this is an edge), just act as if there are no points in the place where the nonexistent bin would be.

# More Overhead

Necessarily, writing the program like this is going to mean more than one kernel.

This does mean there is overhead for each kernel, meaning the total amount of overhead goes up.

Is it worth it?

With 500*64 points:

- OpenCL, no approximations (1 kernel): 0.182s
- OpenCL, with approximations (3 kernels): 0.168s

With 5000*64 points:

- OpenCL, no approximations (1 kernel): 6.131s
- OpenCL, with approximations (3 kernels): 3.506s

# Trading Accuracy for Performance

Consider Monte Carlo integration.
It illustrates a general tradeoff: accuracy vs performance.
Martin Rinard generalized the accuracy vs performance tradeoff with:

- early phase termination [OOPSLA07]
- loop perforation [CSAIL TR 2009]

# Early Phase Termination

We've seen barriers before.
No thread may proceed past a barrier until all of the threads reach the barrier.

This may slow down the program: maybe one of the threads is horribly slow.

Solution: kill the slowest thread.

“Oh no, that’s going to change the meaning of the program!”

# Early Phase Termination: When is it OK anyway?

OK, so we don't want to be completely crazy.

Instead:

- develop a statistical model of the program behaviour.
- only kill tasks that don't introduce unacceptable distortions.

When we run the program:
get the output, plus a confidence interval.

# Early Phase Termination: Two Examples

Monte Carlo simulators:
Raytracers:

- already picking points randomly.

In both cases: spawn a lot of threads.

Could wait for all threads to complete;
or just compensate for missing data points,
assuming they look like points you did compute.

# Early Phase Termination: Another Justification

In scientific computations:

- using points that were measured (subject to error);
- computing using machine numbers (also with error).

Computers are only providing simulations, not ground truth.

Actual question: is the simulation is good enough?

# Loop Perforation

Like early-phase termination, but for sequential programs:
throw away data that's not actually useful.

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

$\Downarrow$

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

This gives a speedup of $\sim 2$ if `numbers[]` is nice.

Works for video encoding: can't observe difference.

# Applications of Reduced Resource Computation

Loop perforation works for:

- evaluating forces on water molecules (summing numbers);
- Monte-Carlo simulation of swaption pricing;
- video encoding.

More on the video encoding example:
Changing loop increments from 4 to 8 gives:

- speedup of 1.67;
- signal-to-noise ratio decrease of 0.87%;
- bitrate increase of 18.47%;
- visually indistinguishable results.

# Video Encoding Skeleton Code

```
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.