Informatik und Angewandte Kognitionswissenschaft
Lehrstuhl für Hochleistungsrechnen

Rainer Schlönvoigt
Thomas Fogal
Prof. Dr. Jens Krüger

High-Performance Computing

http://hpc.uni-duisburg-essen.de/teaching/wt2013/pp-nbody.html

PROGRAMMING IN A NUTSHELL
MAN, I'LL NEVER FIGURE THIS BUG OUT.
HELL-YEAH
10 MINUTES LATER
10 MINUTES LATER
MAN, I'LL NEVER FIGURE THIS BUG OUT.
THREEPANELSOUL.com
matthew boyd - ian mcconville

Exercise 4 (90–210 Points)

1 "Choose your own adventure"

The final project is your choice! We have talked about many aspects of high-performance computing, mostly centering on data organization and communication. In the coming weeks we will discuss topics such as scalability, floating point issues, IO, sequential/temporal consistency and the memory wall. Of course, there are many topics in parallelism and high-performance computing that we simply will not have time to discuss. This is your chance to bring all of these aspects together as well as pursue something that you wish we covered in the course.

You must propose your own work for the final project. I give a few ideas below to start you off, but I hope you will create your own project or at least propose a minor variant on something below.

Your proposals are due by January 30th. I need to receive something *in writing* about what you will do by that time. This does not need to be a shining example of great writing, and can even be just a page of bullet points that outline what you want to do; there just needs to be something written that we agree on. I highly encourage you to discuss your project ideas with me informally beforehand. If your proposal is rejected, you will lose time figuring out details when you could be working on the assignment. Extensions will not be given due to inability to agree on a proposal.

Think big. This is the capstone project for the entire course. You have more time for it than other assignments. Changing 6 lines of your existing simulation and doing a run is not a good final project. To get full credit, you should propose something where you will have to learn something new, in addition to applying knowledge gained from the previous assignments. Try to 'wow' me.

Your assignment does not need to have anything to do with n-body simulations, and you are not required to use any of your old code. You may also work in groups, but there must be a very clear delineation of work so that grades can be individually assigned. Such projects should naturally be more complex than standalone projects.

Some aspects you should consider when you think about your project are:

- Do I already understand the basic approach I will need?
- Can this be implemented in the amount of time available?
- What can someone *see* at the end—how could this be evaluated?
- Are there good resources available for learning more?

You get to choose how important this project is to your grade. *Your proposal must also include the number of points the assignment will be worth.* This number cannot be less than 30% of the total course grade, nor can it exceed 50%.

Projects will be due on *March 21st, 2014*.

2 Project Ideas

2.1 Barnes-Hut acceleration

In n-body simulations with many particles, it often turns out that a large set of particles group together. If we are looking at the force on any one particular particle, then, it will mainly be influenced by the particles in its 'local group'. Particles in distant groups or just distant particles alone will have very little impact on a particle's trajectory.

Some smart people realized this long before us, of course, and decided to take advantage of this fact to accelerate the computation. Those people were Josh Barnes and Piet Hut, and the result is the 'Barnes-Hut' algorithm:

https://de.wikipedia.org/wiki/Barnes-Hut-Algorithmus

The basic idea is to impose a tree on the domain. The tree will group particles which are 'close'; each node will represent some region of space, with leaves containing particles, and internal nodes representing groups of particles which are close together. When computing the acceleration on any given particle, one then traverses the tree as deep as makes sense—comparing the particle's position to the region of space that the current node of the tree represents—and accumulates acceleration as normal. When accumulating from an internal node, one uses the averages of all the particles in that region.

Your task is to implement the Barnes-Hut algorithm in your n-body simulation, and characterize the performance improvement.

Warning: this is a 'data structures'-heavy task. If you have never implemented a tree before, be wary, and if you have significant trouble with pointers, be *very* wary.

2.2 Conway's Game of Life

In 1970 John Conway simplified an idea originally given by John von Neumann and single-handedly invented the field of cellular automata. The "game" he proposed was actually a simulation in which the "player" (I use these terms loosely) has control over only the initial state.

The game is played on a grid of cells. Each cell has binary state: it is either 'on' or 'off', sometimes referred to as 'alive' or 'dead'. Every successive iteration of the simulation proceeds by the following rules:

- if the cell is 'alive':

  - if it has ≤ 1 ‘alive’ neighbors, that cell dies.
  - if it has ≥ 4 ‘alive’ neighbors, that cell dies.
  - if it has 2 or 3 neighbors, it lives to the next iteration.

- if the cell is ‘dead’:

  - if it has exactly 3 ‘alive’ neighbors, it becomes ‘alive’.

Expressed in pseudo-code, the rules are something like this:

```
for y from 0 to dims[0]:
  for x from 0 to dims[1]:
    switch(live_neighbors(grid[x,y])):
      case 0:
      case 1: kill(grid[x,y]); break;
      case 2:
      case 3:
        if(deadp(grid[x,y])) { raise(grid[x,y]); }
        break;
      default:
        if(alivep(grid[x,y])) { kill(grid[x,y]); }
```

Listing 1: Conway’s game of life.

Your task is to write a simulation which takes an initial state and an iteration number, and outputs the state of the system after the given number of iterations. For the edge cases where `x,y` lie off of the defined grid, consider them to always be ‘dead’.

Of course, you need to accelerate this by doing it in parallel. To do this, you will need to give every process a portion of the grid which it is ‘responsible’ for. Of course, calculating `live_neighbors` is difficult at the boundaries of grids: the current process does not know the status of neighboring processes’ grid cells. You will need to communicate to exchange this information.

2.3 Steady state relaxation

In many engineering disciplines, one is interested with identifying the ‘steady state’ of a system given an initial state. As a very simple example: if we drop a ball from some height, it falls and eventually hits the ground. For some period of time, it may bounce and roll around a bit, but eventually the ball comes to rest and will not move until we apply some new external force to it. In this situation, the initial state is where we dropped the ball from and the steady state is the final resting location of the ball.

One common application of this idea is to identify the final temperature of a given region, given an initial state and boundary conditions. An

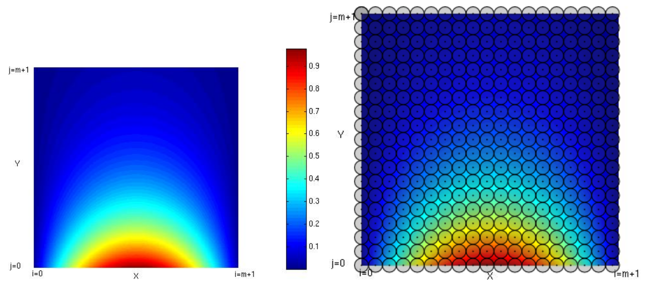

Figure 1: Heat transfer problem example. At the bottom, where $i = X$, the boundary has a constant high heat (1.0). The heat dissipates to neighboring areas and eventually reaches a steady state like the one depicted here. On the right side, a grid is overlayed on the domain; simulations calculate the temperature at each grid point.

example boundary condition might be that the left side of the simulation domain remains at a constant 100°C. The solution would be the temperature distribution when things stabilize.

The mathematics for solving such a system can be as simple as averaging adjacent grid points. If we consider a 2D case, we take the domain and discretize it by choosing where to place grid points, as in the right side of Figure 1. Let's refer to the field on the left side as U, and the discretized version (on right) as u. Then we might speak of u at a particular grid point (x, y), thus: $u_{x,y}$. With this notation the evolution of this steady state problem is simply:

$$u^{n+1}_{x,y} = \frac{u^n_{x-1,y} + u^n_{x+1,y} + u^n_{x,y-1} + u^n_{x,y+1}}{4} \tag{1}$$

where u^n represents u at timestep n.

To solve such a problem, we initialize all the grid points of u, set $n = 0$, and then solve for $n + 1$ everywhere. u^1 then becomes the input to solve for u^2, etc. The process continues until u^{i+1} is so close to u^i that there is, for all practical purposes, no real difference. Of course, there are issues at the grid boundaries: if $(x - 1)$ is -1, then we will at some point ask for the value at $u_{-1,y}$, which is invalid. In Figure 1, the boundary conditions are: if $x = X$ and $y = -1$, then 0.1. Otherwise, 0.0.

Your task is to create a parallel solver for such heat problems. As in the cellular automata (game of life) example, you will need to assign sets of grid points to different processors and exchange information at boundaries. The output would be a file which contains the temperature at every grid point.

2.4 Visualization

Earlier, I provided a script which will take one of your output CSV files and create a rudimentary visualization using ParaView. These visualizations are enough for you as a simulation author to see what is happening, but ultimately not very satisfying—in part due to their simplicity.

In this project, your task is to create more compelling visualizations, using considerably more particles and timesteps. You will perform some sort of 'interesting' simulation run: perhaps setting up particles in the same formation as bodies in our solar system, or as just a random 'galaxy' that you invent and think is interesting. Then you will run your simulation to produce a number of CSVs to use for creating a visualization. The end result should be a movie which depicts the evolution of the simulation run you put together.

Your proposal should include at least two aspects: some aspect of parallelism, either in your simulation or by using ParaView in parallel, and some method[s] to improve the visualizations. Consider how color and transparency might help, for example, or lines for particle trajectories, etc.

3 *sine qua non*

Assignments are due at midnight on the due date.

The proposals need only be a single file. They should be in some sort of document format that can be portably read, such as a plain text file or a PDF. Upload your document to Moodle.

Your final code must include a `Makefile` for compiling. Assignments which do not compile will receive 0 points.

Your assignment will be graded on the `duecray.uni-due.de` supercomputer. It does not matter if your program runs correctly on another machine; it must run correctly on `duecray` to receive credit.