Informatik und Angewandte Kognitionswissenschaft
Lehrstuhl für Hochleistungsrechnen

Thomas Fogal
Prof. Dr. Jens Krüger

High-Performance Computing

http://hpc.uni-due.de/teaching/wt2014/nbody.html

Exercise 6 (100–512 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 that you may steal if you are completely at a loss. However, I hope that you will create your own project or propose a variant on something below.

Your proposal will go through two stages. This allows us to review and revise your project before you are committed to it.

The first version of your proposal is due on January 20th. The second version is due on January 27th. I encourage you to discuss your topic with me informally, but I need to receive something *in writing* on those dates.

Do not sweat the writing much; you are not graded on it. It can be as simple as a few bullet points, if your idea can be communicated succinctly.

Your assignment does not need to have anything to do with n-body simulations, and using your old code is entirely optional. You should work in groups, but there must be a very clear delineation of work so that grades can be individually assigned. Each group member must submit their own proposal.

Make sure your proposal covers at least these topics:

- What it is you intend to implement.
- A list of risks or problems (things you do not already know how to do; new libraries that you will use; potential performance problems that might impede progress; etc.) and what resources exist for mitigating those risks.
- How long this will take to implement versus how much time you have available.
- Final artifacts (scalability graphs, images, test cases that can easily be run, or even a write-up about your project) that can be used to evaluate your work.
- The number of points it should be worth (between 100 and 512), including justification.
- The parts you are responsible for and the parts your partner[s] are responsible for.

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. The number of points can range from 100 to 512. Note that the other assignments totalled 365 points.

Projects will be due on *February 23rd, 2015*.

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 that are 'close'; each node will represent some region of space, with leaves containing particles, and internal nodes representing groups of particles that 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 would be to implement the Barnes-Hut algorithm in your n-body simulation, and characterize the performance benefit.

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 MPI error detection

As you have no doubt realized, writing correct MPI programs is difficult. Each MPI function takes a number of parameters, and it is easy to set them wrong. It is almost impossible for errors to be caught at compile-time. Furthermore,

errors at execution time manifest in difficult-to-understand ways. For example, consider the code in Listing 1:

```
int sz;
MPI_Comm_size(MPI_COMM_WORLD, &sz);
for(size_t i=0; i < (size_t)sz; i++) {
  MPI_Send(v, n, MPI_INT, i, 42, MPI_COMM_WORLD);
}
```

Listing 1: Example loop that might be seen in a naïve `broadcast` implementation. The loop has an error that can be difficult to debug.

This code has a subtle bug: since the code does not check *which* MPI process is running the code, it will send to *all* MPI processes. When it tries to send to itself, it cannot setup a receive and will therefore deadlock.

Understanding what is happening and fixing the deadlock can be difficult. The situation could be considerably simpler, if the implementation of `MPI_Send` started with:

```
int MPI_Send(void* buf, int n, MPI_Datatype type, int dest, int tag,
             MPI_Comm comm) {
  int rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  if(dest == rank) {
    abort();
  }
  ...
}
```

Listing 2: Sample `MPI_Send` implementation that detects the aforementioned error.

What other errors could such an implementation detect?

In this project, your task would be to implement a subset of MPI that does more extensive parameter validation than most MPI libraries. You would create a library that implements some of the standard MPI functions; an '`mpicc`' wrapper that invokes `gcc` and forces said library to be linked in; and a limited '`mpirun`' that only accepts an `-np` (number of processors) parameter of 2. To communicate, your library would use TCP sockets under the hood.

2.3 Conway's Game of Life

In 1970 John Conway simplified an idea originally given by John von Neumann and thereby 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[1]:
  for x from 0 to dims[0]:
    switch(live_neighbors(grid[x,y])):
      case 0:
      case 1: kill(grid[x,y]); break;
      case 2:
      case 3:
        if(deadp(grid[x,y])) { resurrect(grid[x,y]); }
        break;
      default:
        if(alivep(grid[x,y])) { kill(grid[x,y]); }
```

Listing 3: Conway's game of life.

Your task is to write a simulation that 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. 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.

2.4 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

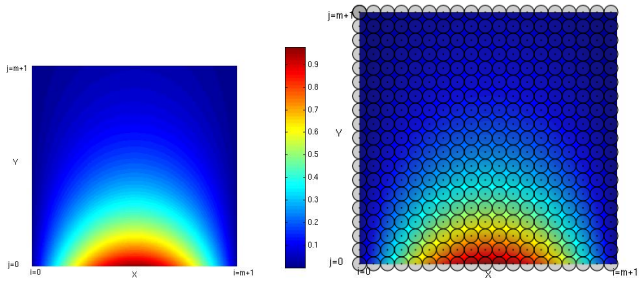

Figure 1: Heat transfer problem example. At the bottom, where $i = X$, the boundary has a constant high heat (1.0). The heat dissipates to neighboring areas and eventually reaches a steady state like the one depicted here. On the right side, a grid is overlayed on the domain; simulations calculate the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

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 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:

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

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 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 that contains the temperature at every grid point.

2.5 Visualization

Earlier, I provided a script that 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 that 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.

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.