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 3 (90 Points)

All assignments should be pushed to your personal Git repository. Assignments are due at midnight on the due date. No late assignments will be accepted.

All assignments must include a `makefile` for compiling your assignments. Assignments which do not compile will receive 0 points. Assignments that do not satisfy the test inputs will receive 0 points.

Please **do not include output** other than what was requested by the assignment details.

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.

1 Introduction

In this assignment you will introduce MPI-based parallelism into your N-Body simulation. The primary objective is to accelerate the computation when there are a large number of particles. A secondary consideration is that the distributed memory system allows a larger total overall memory, so you should be able to model a larger number of particles.

There are some minor differences when using the DUE Cray supercomputer. I will note any such differences in this document as we come to them.

2 MPI

For our simulations, the runtime depends strongly on the number of particles present you simulate.

The runtime of an N-Body simulation depends strongly on the number of cores. To speed the process up, we will parallelize the code to run on any number of cores. The additional cores could be some of the cores on a large cluster, like `duecray`, or they could just be other cores on your local machine. Code written to the MPI standard is written the same in both cases; the runtime environment takes care of applying it to both system configurations.

We will use the `duecray.uni-due.de` machine in this course. You can access the machine by ssh'ing to that hostname and using the special '`_hpcc`' login name that was provided for you. There are a few differences to using MPI on this machine versus using MPI on your local machine.

2.1 Compilation

MPI programs need additional includes and must link against extra libraries to function correctly. You would need to identify these for whatever MPI implementation you have installed and then add them to your `CFLAGS`, `LDFLAGS`, and `LDLIBS` in your makefile. However, most MPI implementations come with a wrapper script, `mpicc`, that invokes the C compiler and implicitly adds the flags you need. So the easiest and most portable solution is to simply execute

```
$ export CC=mpicc
```

before you run '`make`'. If you use the appropriate variables in your makefile, this should make your build use the correct MPI-based compiler.

2.1.1 `duecray`

As we mentioned in class, we will use the GNU compilers for this course. However, `duecray`'s default environment sets up the PGI compiler suite. To change this, you need to run:

```
module swap PrgEnv-pgi PrgEnv-gnu
```

before you compile your programs. Note that, for complicated reasons, `module` commands cannot be put into makefiles; you must execute this manually on every login!

2.2 Execution

MPI programs also need to be executed by a special program, '`mpirun`'. Just add this in front of the command you would normally run, adding `-np` N to execute the progam using N processes.

```
$ mpirun -np 4 ./nbody -i input.csv -n 4 -e 0.1 -d 10.0 -t 0.5 -o 2
```

2.2.1 `duecray`, PBS

On the `duecray` cluster there is a job scheduler, PBS, that you must run your jobs through. The scheduler figures out when to run your jobs as opposed to any other users of the system. Note that this means your program might not run immediately.

Instead of simply giving the job scheduler your `mpirun` command, you give it a PBS script. This is essentially a shell script with a set of `#PBS` directives at the top and the actual command embedded in the script. The directives communicate metadata such as how long the job is expected to run. An example PBS script is given in Listing 1:

```
1  #PBS -N tjf_loves_hpc
2  #PBS -l mppwidth=4
3  #PBS -l mppnppn=2
4  #PBS -l mppdepth=1
5  #PBS -l walltime=0:30:00
6  #PBS -o stdout
7  #PBS -e stderr
8
9  cd $PBS_O_WORKDIR
10 aprun -n 4 /homes/adc701q/test-assignments/hw
11 exit 0
```

Listing 1: Simple PBS job submission script.

This creates a job named '`tjf_loves_hpc`' in the queue. Names do not have much purpose other than helping you to identify your jobs when you query the system wondering if they have run yet. Note the use of `aprun` instead of `mpirun`.

Due to filesystem issues on the cray, I recommend you create a subdirectory under `/scratch` and run/create your jobs from there.

3 Work distribution

You will need to decide how to distribute particles among your processors. Use `MPI_Comm_size` and `MPI_Comm_rank` to compute the subset of particles

that the current machine should utilize. You will then need to distribute those particles to the processors as appropriate.

That particle-to-processor mapping is then set in stone for the life of the process. Each process will calculate updates for only the particles it 'owns'. By dividing the work this way and computing updates in parallel, you should (hopefully) reduce the runtime of your simulation code.

4 Synchronization

At the end of iteration you must update all the particles' positions across all processors.

4.1 Particle positions

There are a couple ways you could think about this, highlighted by Figure 1. The first is to have every process do sends to every other process in a distributed fashion. The second is to think of one process (typically the process with rank 0) as the 'master' process that collects and broadcasts all particles.

It is not important which method you choose, for this assignment. Do note that performance will be a factor in later assignments, however.

5 Input

The command line options your program should accept as well as the input file format it utilizes are both unchanged from the previous assignment. You should of course be creating your own test files, but here is another which may be of use to you:

```
"mass", "x", "y", "z", "vx", "vy", "vz"
1000.0, 07.3713,   0.3218,   3.1043,  1.1833, 2.0134,   2.2291
1500.0, 10.2746,  26.4729,  70.5028,  8.0630, 8.4932,   6.0171
 330.0, 02.2911,  09.1276,  10.8019,  6.3234, 1.1237,  15.5061
1394.0, 24.1717,  10.4758,  15.2072,  2.0334, 4.7838, 101.1451
```

Listing 2: Sample input file (CSV).

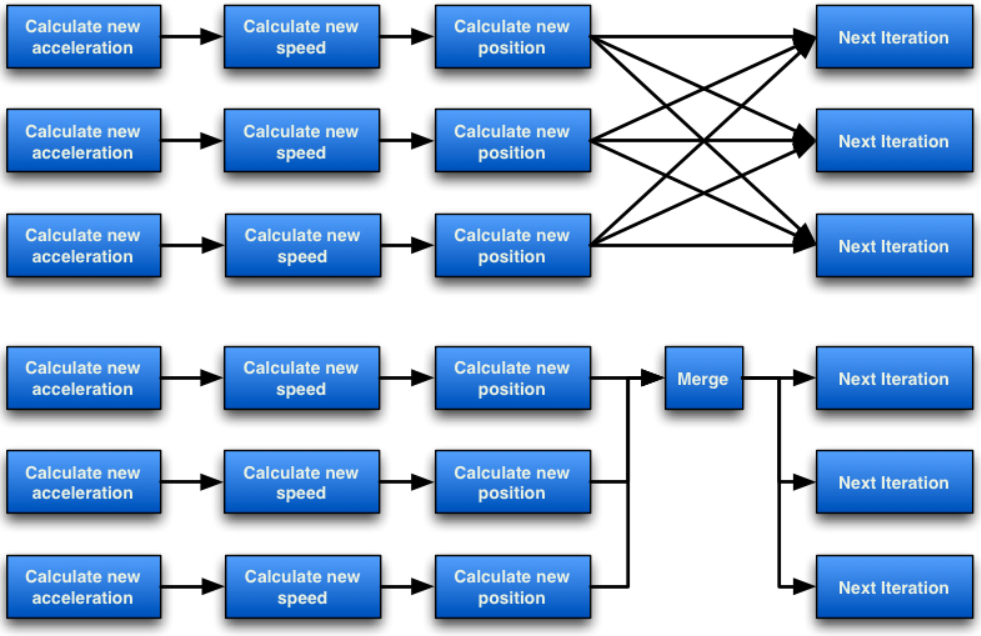

Figure 1: Different ways of synchronizing information, such as the particle positions in your simulation. The first scatters everything in a multitude of sends; the latter method gathers everything to one process and then broadcasts it out to everyone.

6 Output Format

The output file format is unchanged from the previous assignment.

7 *sine qua non*

As always, submit your assignment by committing your code to your personal repository. Every assignment utilizes a new repository. The name for this assignment is **as3-*username***, where *username* is your name on our git server.

All assignments must include a makefile to compile your program. No makefile, no points!