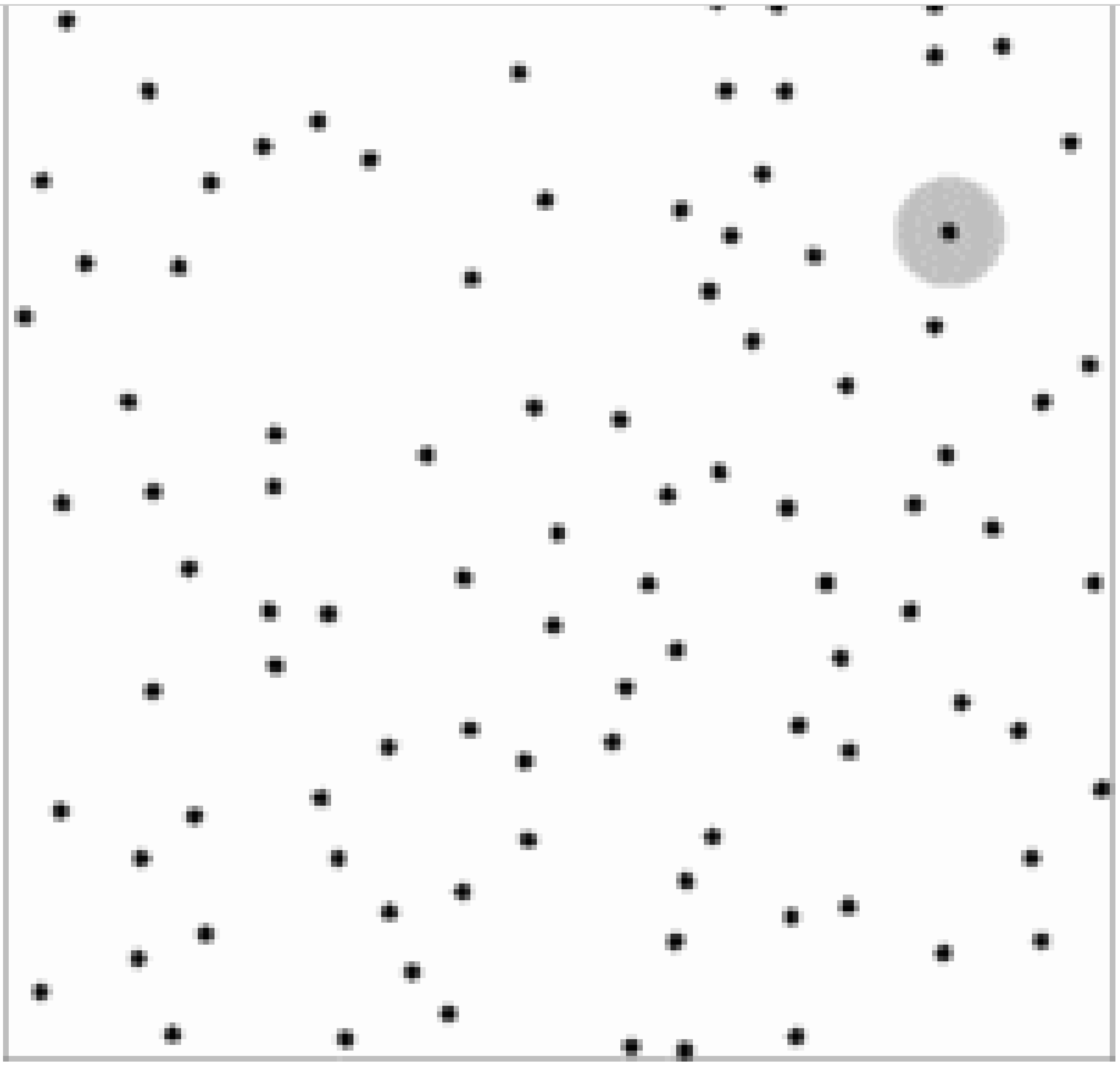
Homework 2 (Part 2)

Parallelizing a Particle Simulation

Overview

This assignment is an introduction to parallel programming using a distributed memory model. Most of this page will be similar to the HW 2-1 page.

In this assignment, we will be parallelizing a toy particle simulation (similar simulations are used in mechanics, biology, and astronomy). In our simulation, particles interact by repelling one another. A run of our simulation is shown here:



Asymptotic Complexity

Serial Solution Time Complexity

If we were to naively compute the forces on the particles by iterating through every pair of particles, then we would expect the asymptotic complexity of our simulation to be *O(n^2)*.

However, in our simulation, we have chosen a density of particles sufficiently low so that with *n* particles, we expect only *O*(*n*) interactions. An efficient implementation can reach this time complexity.

Parallel Speedup

Suppose we have a code that runs in time *T* = *O(n)* on a single processor. Then we'd hope to run close to time *T/p* when using *p* processors. After implementing an efficient serial *O(n)* solution, you will attempt to reach this speedup using MPI.

Important: You CANNOT use OpenMP for this assignment. All of your parallel speedup must come from distributed memory parallelism (MPI).

For Remote Students

Dear remote students, we are thrilled to be a part of your parallel computing learning experience and to share these resources with you! To avoid confusion, please note that the assignment instructions, deadlines, and other assignment details posted here were designed for the local students. You should check with your local instruction team about submission, deadlines, job-running details, etc. and utilize Moodle for questions. With that in mind, the problem statement, source code, and references should still help you get started (just beware of institution-specific instructions). Best of luck and we hope you enjoy the assignment!

Due Date: Wednesday March 6th, 2024 (11:59 PM PST)

Teams

You must use the same groups as with HW2-1. If this is a problem, please privately contact the GSIs.

Getting Set Up

The starter code is available on GitHub at https://github.com/Berkeley-CS267/hw2-2 and should work out of the box. To get started, we recommend you log in to perlmutter and download the first part of the assignment. This will look something like the following:

```
student@local:~> ssh demmel@perlmutter-p1.nersc.gov
student@perlmutter-p1:~> git clone https://github.com/Berkeley-CS267/hw2-2
student@perlmutter-p1:~> cd hw2-2
student@perlmutter-p1:~/hw2-2> ls
CMakeLists.txt common.h job-mpi main.cpp mpi.cpp
```

There are five files in the base repository. Their purposes are as follows:

CMakeLists.txt

The build system that manages compiling your code.

main.cpp

A driver program that runs your code.

common.h

A header file with shared declarations

job-mpi

A sample job script to run the MPI executable

mpi.cpp - - - You may modify this file.

A skeleton file where you will implement your mpi simulation algorithm. It is your job to write an algorithm within the simulate_one_step and gather_for_save functions.

P ;e **do not** modify any of the files besides mpi.cpp.

First, we need to make sure that the CMake module is loaded.

```
student@perlmutter:login11:~/hw2-2> module load cmake
```

You should put these commands in your ~/.bash_profile file to avoid typing them every time you log in.

Next, let's build the code. CMake prefers out of tree builds, so we start by creating a build directory.

```
student@perlmutter:login11:~/hw2-2> mkdir build
student@perlmutter:login11:~/hw2-2> cd build
student@perlmutter:login11:~/hw2-2/build>
```

Next, we have to **configure our build.** We can either build our code in **Debug** mode or **Release** mode. In debug mode, optimizations are disabled and debug symbols are embedded in the binary for easier debugging with GDB. In release mode, optimizations are enabled, and debug symbols are omitted. For example:

```
student@perlmutter:login11:~/hw2-2/build> cmake -DCMAKE_BUILD_TYPE=Release
..
-- The C compiler identification is GNU 8.3.0
...
-- Configuring done
-- Generating done
-- Build files have been written to: /global/homes/s/student/hw2-2/build
```

Once our build is configured, we may actually **execute** the build:

```
student@perlmutter:login11:~/hw2-2/build> make
Scanning dependencies of target mpi
[ 33%] Building CXX object CMakeFiles/mpi.dir/main.cpp.o
[ 66%] Building CXX object CMakeFiles/mpi.dir/mpi.cpp.o
[100%] Linking CXX executable mpi
[100%] Built target mpi
student@perlmutter:~/hw2-2/build> ls
CMakeCache.txt  CMakeFiles  cmake_install.cmake  Makefile  mpi job-mpi
```

We now have a **binary** (mpi) and a **job script** (job-mpi).

R nning our Code

```
student@perlmutter:login11:~/hw2-2/build> salloc -N 2 -C cpu -q interactive
-t 01:00:00
salloc: Pending job allocation 5294543
salloc: job 5294543 queued and waiting for resources
salloc: job 5294543 has been allocated resources
salloc: Granted job allocation 5294543
salloc: Waiting for resource configuration
salloc: Nodes nid[005305-005306] are ready for job
student@nid005305:~/hw2-2/build>
```

You now have a shell into one of the two allocated nodes. We recommend that you allocate only a single node and test on multiple MPI ranks with that node until you are ready to conduct a full scaling benchmark.

Unlike earlier assignments, you cannot directly run the executable from the command prompt! You must use `srun` or `sbatch` with the sample jobscript that we provide you. You can modify the jobscript to benchmark different runtime configurations.

If you choose to run the binary using `srun` within an interactive session, you should set any environment variables in your interactive session to match the environment variables set by the jobscript. After you have done so, here's how to run the simulation for two nodes, 6 million particles, and 64 MPI ranks per node, for a total of 128 total MPI ranks:

```
student@nid005305:~/hw2-2/build> srun -N 2 --ntasks-per-node=64 ./mpi -n
6000000 -s 1
Simulation Time = 19.504 seconds for 6000000 particles.
```

To test on only a single node with 68 total MPI ranks, you can run:

```
student@nid005305:~/hw2-2/build> srun -N 1 --ntasks-per-node=64 ./mpi -n
6000000 -s 1
Simulation Time = 51.1204 seconds for 6000000 particles.
```

Before you try writing any parallel code, you should make sure that you have a correct serial implementation. To benchmark the program in a serial configuration, run on a single node with `--ntasks-per-node=1`. You should measure the strong and weak scaling of your implementation by varying the total number of MPI ranks from 1 to 128.

Important notes for Performance:

There will be two types of scaling that are tested for your parallel codes:

the work/processor stays the same *(Note that for the purposes of this assignment we will assume a linear scaling between work and processors)*

While the scripts we are providing have small numbers of particles 1000 to allow for the $O(n^2)$ algorithm to finish execution, the final codes should be tested with values much larger to better see their performance. **Test up to 2 nodes and 6,000,000 particles.**

Grading

We will grade your assignment by reviewing your assignment write-up, measuring the scaling of the implementation, and benchmarking your code's raw performance. To benchmark your code, we will compile it with the exact process detailed above, with the GNU compiler.

Submission Details

Supposing your custom group name is **XYZ**, follow these steps to create an appropriate submission archive:

- Ensure that your write-up is located in your source directory, next to **serial.cpp**. It should be named **cs267XYZ_hw2_2.pdf**
- From your **build** directory, run:

```
student@perlmutter:login11:~/hw2-2/build> cmake -DGROUP_NAME=XYZ ..
student@perlmutter:login11:~/hw2-2/build> make package
```

This second command will fail if the PDF is not present.

- Confirm that it worked using the following command. You should see output like:

```
student@perlmutter:login11:~/hw2-2/build> tar tfz cs267XYZ_hw2_2.tar.gz
cs267XYZ_hw2_2/cs267XYZ_hw2_2.pdf
cs267XYZ_hw2_2/mpi.cpp
```

- Submit your .tar.gz through bCourses.

Write-up Details

- Your write-up should contain:
  - The names of the people in your group and each member's contribution.
  - A plot in log-log scale that shows that your parallel codes performance and a description of the data structures that you used to achieve it.
  - A description of the communication you used in the distributed memory implementation.

speedup and a discussion on whether it is possible to do better. Both strong and weak scaling.
- Where does the time go? Consider breaking down the runtime into computation time, synchronization time and/or communication time. How do they scale with p?

Notes:

- Your grade will mostly depend on three factors:
  - Scaling sustained by your codes on the Perlmutter supercomputer (varying n).
  - Performance sustained by your codes on the Perlmutter supercomputer.
  - Explanations of your methodologies and the performance features you observed (including what didn't work).
- You must use the GNU C Compiler for this assignment. If your code does not compile and run with GCC, it will not be graded.
- If your code produces incorrect results, it will not be graded.

For info on running the rendering output, refer to the HW2-1 page.

For info on checking output correctness, refer to the HW2-1 page.

Resources

- Programming in distributed memory models are introduced in Lectures.
- You may consider using atomic operations such as __sync_lock_test_and_set with the GNU compiler.
- Other useful resources: MPI specifications.
- It can be very useful to use a performance measuring tool in this homework. Parallel profiling is a complicated business but there are a couple of tools that can help.
  - TAU (Tuning and Analysis Utilities) is a source code instrumentation system to gather profiling information. You need to "module load tau" to access these capabilities. This system can profile MPI, OpenMP and PThread code, and mixtures, but it has a learning curve.
  - HPCToolkit Is a sampling profiler for parallel programs. You need to "module load hpctoolkit". You can install the hpcviewer on your own computer for offline analysis, or use the one on NERSC by using the NX client to get X windows displayed back to your own machine.
- If you are using TAU or HPCToolkit you should run in your $SCRATCH directory which has faster