



Applications of Parallel Computers

Home ► My courses ► Parallel ► Programming Homework 2 - Parallelizing a Particle ... ► Problem Description

NAVIGATION

Home

 Dashboard


Site pages

My courses

Parallel

Participant
s

 Badges

 Competen
cies

 Grades

General

Institutiona

I Forums

Homework

0 -

Describing
a Parallel
Applicatio
n

Lecture 1 -
Introductio
n

Lecture 2 -
Single
Processor

Problem Description

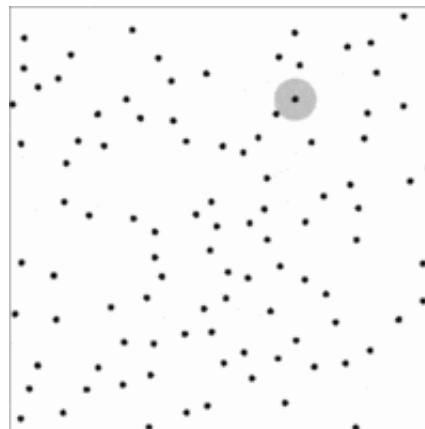
CS267 Assignment 2: Parallelizing a Particle Simulation

Assignment 2: Parallelizing a Particle Simulation

Part 1 Due Date: To be determined by your instructor

Problem statement

Your goal is to parallelize on XSEDE's Bridges supercomputer a toy particle simulator (similar particle simulators are used in mechanics, biology, astronomy, etc.) that reproduces the behaviour shown in the following animation:



Machines
 Lecture 3 -
 Optimizing
 Matrix
 Multiply
 (cont),
 Int...
 Programmi
 ng
 Homework
 1 - Tuning
 Matrix
 Multiply
 Lecture 4 -
 Shared-
 Memory
 Programmi
 ng
 Lecture 5 -
 Roofline
 and
 Performan
 ce
 Modeling
 Lecture 6 -
 Source of
 Parallelism
 and
 Locality in
 ...
 Lecture 7 -
 Sources of
 Parallelism
 in
 Simulation
 -...
 Lecture 8 -
 An
 Introductio
 n to

The range of interaction forces (*cutoff*) is limited as shown in grey for a selected particle. Density is set sufficiently low so that given n particles, only $O(n)$ interactions are expected.

Suppose we have a code that runs in time $T = O(n)$ on a single processor. Then we'd hope to run in time T/p when using p processors. We'd like you to write parallel codes that approach these expectations.

Correctness and Performance

A simple correctness check which computes the minimal distance between 2 particles during the entire simulation is provided. A correct simulation will have particles stay at greater than 0.4 (of cutoff) with typical values between 0.7-0.8. A simulation where particles don't interact correctly will be less than 0.4 (of cutoff) with typical values between 0.01-0.05. More details as well as an average distance are described in the source file.

The code we are providing will do this distance check based on the calls to the interaction function but the full autograder will do the checks based on the outputted txt file. We'd recommend keeping the correctness checker in your code (for the OpenMP, Pthreads and MPI codes the overhead isn't significant) but depending on performance desires it can be deleted as long as the simulation can still output a correct txt file.

The performance of the code is determined by doing multiple runs of the simulation with increasing particle numbers and comparing the performance with the autograder.cpp provided. This can be done automatically with the *auto-** scripts.

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

- In **strong scaling** we keep the problem size constant but increase the number of processors
- In **weak scaling** we increase the problem size proportionally to the number of processors so the work/processor stays the same (*Note that for the purposes of this assignment we will assume a linear scaling between work and processors*)

For more details on the options provided by each file you can use the *-h* flag on the compiled executables.

Important note for Performance:

GPGPU
 Programmi
 ng
 Lecture 9:
 Distributed
 Memory
 Machines
 and
 Program...
 Lecture 10
 -
 Advanced
 MPI and
 Collective
 Communic
 a...
 Programmi
 ng
 Homework
 2 -
 Parallelizin
 g a
 Particle ...



**Proble
 m
 Descrip
 tion**



Starting
 code



HW2
 Assign
 ment
 submis
 sion



HW2
 Discuss
 ion



Proble
 m

While the scripts we are providing have small numbers of particles (500-1000) to allow for the $O(n^2)$ algorithm to finish execution, the final codes will be tested with values 100 times larger (50000-100000) to better see their performance.

Grading

Your grade will depend on the performance and efficiency sustained by your codes on Stampede and will be broken into 2 parts.


For Part 1 of Assignment 2 you will be optimizing the Serial, OpenMP or Pthreads and MPI versions.


Your final grading scheme for Part 1 and Part 2 is the following: (Serial - 20%, OpenMP or Pthreads - 30%, MPI - 30% and GPU - 20%).

- **Serial Code** performance will be tested via fitting multiple runs of the code to a line on a log/log plot and calculating the slope of that line. This will determine whether your code is attaining the $O(n)$ desired complexity versus the starting $O(n^2)$. With an achieved result of $O(n^x)$ you will receive
 - If x is between 2 and 1.5 you will receive a serial score between 0 and 75 proportional to x . (Ex: 1.75 gives a score of 37.5)
 - If x is between 1.5 and 1.3 you will receive a serial score between 75 and 100 proportional to x . (Ex: 1.34 gives a score of 95)
 - If x is below 1.3 you will receive a serial score of 100.
- **Shared Memory performance and MPI Performance** will be tested via calculating the strong and weak scaling average efficiencies for 1,2,4,8,16 processor/thread runs.
 - The strong and weak average efficiencies (eff_{ss} , eff_{ws}) will each contribute 50% of a grade to either OpenMP, Pthreads or MPI codes.
 - The grade for the "Shared Memory" part of the Total grade will be selected as the maximum between the OpenMP and Pthreads Grades so only one implementation needs to be provided.
 - An efficiency (strong or weak) will give a grade according to the following rules:
 - If the efficiency is between 0 and 0.5 you will receive a score between 0 and 75 proportional to it (Ex: 0.2 gives a score of 30)
 - If the efficiency is between 0.5 and 0.8 you will receive a score between 75 and 100 proportional to it (Ex: 0.62 gives a score of 85)
 - If the efficiency is 0.8 or above you will receive a score of 100

Example grade

Descrip
tion
(GPU)

 Starting
code
(GPU)

 HW2
Assign
ment
submis
sion
(GPU)

Lecture 11
- PGAS
and
UPC++

Lecture 12
- Cloud
Computing
and Big
Data
Processing
Programmi
ng
Homework
3 -
Parallelize
Graph
Algorit...

Lecture 13
- Parallel
Matrix
Multiply

Lecture 14
- Dense
Linear
Algebra

Lecture 15
- Sparse
Matrix-

Lets assume that John Student submits the assignment and has:

- a Serial Code with $x = 1.25$
- an OpenMP code with $\text{eff}_{\text{ss}}=0.8$ and $\text{eff}_{\text{ws}}=0.7$
- an MPI code with $\text{eff}_{\text{ss}}=0.4$ and $\text{eff}_{\text{ws}}=0.5$

His grades per each section would be:

- Serial = 100
- OpenMP = $0.5 * 100 + 0.5 * 91.66 = 95.83$
- MPI = $0.5 * 60 + 0.5 * 75 = 67.5$

His total grade (for Part 1) = Serial * 0.2 + {OpenMP, Pthreads} * 0.3 + MPI * 0.3 =
 $100 * 0.2 + 95.83 * 0.3 + 67.5 * 0.3 = \mathbf{70 \text{ (out of 80)}}$

Source files

The starting files with their descriptions can be found in the Starting Code folder.

To download all the files directly to Stampede you can use the following command:

```
wget https://people.eecs.berkeley.edu/~brock/XSEDE-267/xsede_hw2_2019.tar.gz
```

Login, Compilation and Job submission

The easiest way to access the machines is to login directly with your own ssh client to **login.xsede.org** and from there **gsissh** into the correct machine. More information is available here on the single login system. For this assignment we will be using the Stampede supercomputer.

Another easy way to login to XSEDE resources is via the Accounts tab in XSEDE User Portal. To reach this page login to XSEDE User Portal, navigate to MY XSEDE tab and from there select the Accounts page. All machines you have access to will have a **login** option next to them that will launch OpenSSH via a Java Applet.

Please be aware that most XSEDE resources are not available via direct ssh and all requests must go through the XSEDE single login system.

Vector
Multiplicati
on a...
Lecture 16
-
Structured
Grids
Lecture
17:
Machine
Learning -
Part 1
(Supervise
d ...
Lecture 18
- Machine
Learning -
Part 2
(Unsupervi
s...
Lecture 19
- Parallel
Graph
Algorithms
Lecture 20
- Fast
Fourier
Transform
Lecture 21
- Climate
Modeling
Lecture
22: Sorting
and
Searching
Lecture 23
- Dynamic
Load
Balancing

You can extract the downloaded directory from the tgz archive with:

```
tar -xvf xsede_hw2_2018.tgz
```

The default Makefile is built with the default intel compiler; therefore you can simply type **make** to compile the code.

To submit jobs on the Bridges system you will use the SLURM Batch Environment interface for example:

```
sbatch job-bridges-mpi16
```

To check the status of running jobs you can use the following command:

```
squeue -u <username>
```

Once you have code that gives correct output you should check it's efficiency with the autograder provided by running the command:

```
sbatch auto-bridges-mpi16
```

This example will run your MPI code with the "no output" (-no) option and run 1,2,4,8,16 processor runs of your parallel code with varying sizes of problems to determine strong and weak scaling.

Note that at the moment the Bridges system is very fast in processing jobs; therefore depending on your problem size the job might finish even before running the squeue command; check the folder for the .stdout output file if you had a successful submission

For more details on sbatch commands please see Bridges documentation page.

Resources

- Bridges Computing Environment
- Shared memory implementations may require using locks that are available as `omp_lock_t` in OpenMP (requires `omp.h`) and `pthread_mutex_t` in pthreads (requires `pthread.h`).
- Alternatively, you may consider using atomic operations as described here for the Intel compilers and `__sync_lock_test_and_set` in the newer versions of the GNU compiler.
- Distributed memory implementation may benefit from overlapping

Lecture 24

-

Hierarchical Methods for the N-Body P...

Lecture 25

-

Computational Biology

Lecture 26

- Big

Bang, Big Data, and Big Iron

Lecture 27

-

Supercomputers and SuperIntelligence

Lecture 28

- Quantum Computing

communication and computation that is provided by nonblocking MPI routines such as `MPI_Isend` and `MPI_Irecv`.

- Other useful resources: pthreads tutorial, OpenMP tutorial, OpenMP specifications and MPI specifications.

Optional: Other improvements

- Implementing a hybrid code that uses shared memory on each node and MPI between nodes
- Parallelizing the particle creation and output
- Comparing scalability on other XSEDE platforms

Last modified: Monday, 14 January 2019, 5:13 PM

◀ Quiz 10

Jump to...

Starting code ▶



ADMINISTRATION

Course administration

You are logged in as Anthony Bugatto (Log out)

Parallel

Get the mobile app