Homework 2

Homework 2 will be part of Project 1

EAS520/DSC520

High-Performance Scientific Computing

Problem 1

Redo the numerical experiment in Mini Lecture 2 using **two** other computer languages other than the one you are comfortable writing in. Run the experiment for small, medium, and large number of iterations n.

Record the speedup and average efficiency.

Problem 2: Background processes case.

Suppose you are running several applications with high CPU usage in the background while executing the embarrassingly parallel numerical experiment in Mini Lecture 2.

For example, you play several 4K videos on youtube on several tabs in the browser while running the **mainp.m** (provided in mycourses) with a small, medium, and large number of iterations n.

Does it affect the speedup and average efficiency?

Problem 3: Random running time scenario.

Redo the numerical experiment in Mini Lecture 2 with the case where the **timeconsumingfun** pauses at a random time between 1 to 5 seconds (instead of just a fixed time of 5 seconds). Run the experiment for small, medium, and large number of iterations n.

Record the speedup and average efficiency.

Homework 3

Homework 3 will be part of Project 1

EAS520/DSC520

High-Performance Scientific Computing

Problem 1

Redo the root-finding numerical experiment in Mini Lecture 3 using **two** other computer languages other than the one you are comfortable writing in.

Record the speedup and average efficiency.

Problem 2

Palindromes are phrases that read the same way right-to-left as they do left-to-right, when disregarding capitalization. Find codes from the internet (do not forget to cite the author properly) or even write the codes yourself, which can be used to examine a large set of strings whose subset may contain a 5 letter palindrome.

For example,

AIGFJHPCPHTUZW

contains **HPCPH**.

Test your code for a large set of strings containing 1000 capital letters, with a 5-letter palindrome embedded somewhere in it. If the code finds the palindrome, set the output variable to 1, otherwise 0.

Suppose you have 100 different text files. Each contains 1000 capital letters with a 5-letter palindrome embedded somewhere in it. Use for-loop to check those files serially. Next, check those files in a nearly embarrassingly parallel method way. Record the speedup and efficiency as the number of text files become larger.

Problem 3

Suppose you are tasked to solve a special ordinary differential equation (ODE), called the Van der Pol oscillator

$$\frac{d^2x}{dt^2} - \varepsilon(1 - x^2)\frac{dx}{dt} + x = 0$$

for $\varepsilon = 0.1$ from $t = 0$ until $t = 10$ with an initial condition $x(0) = 0.5$.

Use codes specifically used for solving ODEs and use them to solve the ODE above.

In an embarrasingly parallel way, solve that equation for 5000 different values of ε between $\varepsilon = 0.01$ and $\varepsilon = 4$.

Collect and plot the solutions of the ODE for 5 different values of ε.

Compare the serial and parallel execution time and record the speedup and average efficiency.

Homework 4

Homework 4 will be part of Project 1

EAS520/DSC520

High-Performance Scientific Computing

Problem 1

Using the SPMD technique similar to page 5 on the Mini Lecture 4, redo the integral example for the double integral case on a rectangular domain $[a, b] \times [c, d]$, i.e.

$$\int_a^b \int_c^d f(x, y) dx dy$$

- You can choose your own integral function.
- Break the rectangular domain based on the number of workers.
- Compute integrals independently on those subdomains.
- Add local integration values to obtain the total integral.

Note:

In MATLAB, use **integral2** function for performing a double integral. Other languages may have a similar spmd environment and a double integral subroutine.

Problem 2

Using the SPMD technique similar to page 5 on the Mini Lecture 4, redo the integral example for the triple integral case on a cuboid domain $[a, b] \times [c, d] \times [e, f]$, i.e.

$$\int_a^b \int_c^d \int_e^f f(x, y, z) dx dy dz$$

- You can choose your own integral function.
- Break the cuboid domain based on the number of workers.
- Compute integrals independently on those subdomains.
- Add local integration values to obtain the total integral.

Note:

In MATLAB, use **integral3** function for performing a triple integral. Other languages may have a similar spmd environment and a triple integral subroutine.

Problem 3

Modify the mandelbrot spmd program such that the grid domain 1000 x 1000 is partitioned into chunks of *square* blocks.

For example:

- In the case of 2 x 2 blocks, each block will have 500 x 500 grid points.
- For the 4 x 4 blocks, each block will have 250 x 250 points, and so on.

Use the MATLAB on the UMass Dartmouth Corsair Desktop (choose MATLAB with GPU option) if you need more cores/workers.

Test your code for the grid domain of size 2000 x 2000.

Homework 7

Homework 7 will be part of Project 2

EAS520/DSC520

High-Performance Scientific Computing

Problem 1

Create a function called **mymapreduce**, which accept three variables: **v**, **nc**, and **op** as inputs and **result** as an output variable.

v is a vector, **nc** is the number of cores, and **op** is a string describing an operation to be done to the vector v in a distributed way.

For example: Let v = [1 2 0 1 3 1], nc = 2, op = 'sum'.

- Calling **result** = **mymapreduce(v,nc,op)** should give result = 8

Inside the mymapreduce function, there are two stages.

The map part: the vector v has to be partitioned (split) as a **codistributed array** based on the number of cores nc, and the op has to be done to each partition. **The reduce part:** Results from each partition are gathered to be used for generating the final output.

In the above example (nc=2), in the map part, v is split into [1 2 0] and [1 3 1]. The local sums are 3 and 5, respectively. In the reduce part, the final sum is 8.

Test your mymapreduce with a random vector of length 1000, nc = 2, 4, 8, and the operation op as 'sum', 'average', 'min', and 'max'. Use Corsair Desktop if you need more cores.

Problem 2

Modify mymapreduce in problem 1 such that it accepts the input v as an m x n matrix (not just a vector).

Test it with a random matrix of size 800 x 1000, nc = 2, 4, 8, and the operation op as 'sum', 'average', 'min', and 'max'. Use Corsair Desktop if you need more cores.

Problem 3

Modify mymapreduce in problem 1 such that it accepts the input v as an m x n x p "cuboid matrix".

Test it with a random cuboid matrix of size 50 x 100 x 200, nc = 2, 4, 8, and the operation op as 'sum', 'average', 'min', and 'max'. Use Corsair Desktop if you need more cores.

Homework 8

Homework 8 will be part of Project 2

EAS520/DSC520

High-Performance Scientific Computing

Problem 1

Use the message passing technique to simulate a random walk of n objects on a "periodic" or "circular" integer lattice. The rules are described as the following:

1. The lattice can be discretized as N grid points with a location at 1, 2, ..., N.

2. Initially, the n objects can be placed anywhere randomly in the lattice. One or more objects can occupy the same grid point at any given time.

3. Each object must flip a coin before moving a step: Move 1 step to the right (Head) or Move 1 step to the left (Tail).

4. If an object located at grid point N moves 1 step to the right, it should land in grid point 1. If an object located at grid point 1 moves 1 step to the left, it should land in grid point N.

The lattice should be distributed into np partitions where np is the number of workers (cpus/cores). For example, for the case where N=10 and np=2, worker 1 is taking care of grid 1, ..., 5, and worker 2 is taking care of grid 6, ..., 10.

Utilize the commands **labSend**, **labProbe**, **labReceive**, and **labBarrier** to deal with objects crossing partitions. Stop the simulation when all objects achieve M steps. Plot the initial and final positions of all objects.

Problem 2

Use the message passing technique to simulate online gambling. The rules are described as the following:

1. There is one dealer and n players (bettors). Initially, the dealer has an M amount of money, and each player has $m_1, m_2, \ldots, m_n$ amount of funds in the bank, respectively. You may also assume that the dealer has much more money than all players combined.

2. Each player will make a bet. Initial bet can range from $1 to the maximum available money in the bettor's bank.

3. The chance of winning and losing is 50-50 (like flipping a coin). If one wins, the dealer pays the bettor, and that individual will put half of the amount of winning money for the next bet. If one loses, the bettor pays the dealer and then puts one-fourth of the remaining money in the bank for the next bet.

4. A player or a dealer is declared bankrupt if there is only less than $1 in the bank.

Simulate the gambling process until either the dealer or all players went bankrupt. For example, for the case with 1 dealer and n=3 players, you can use 4 cores. Core 1 is reserved for the dealer, core 2 for bettor 1, core 3 for bettor 2, core 4 for bettor 3.

Homework 9

Homework 9 will be the (last) part of Project 2

EAS520/DSC520

High-Performance Scientific Computing

Problem

Create a 2000x2000 random matrix A and a 2000x1 random vector b.

With MATLAB, use for-loop to perform A*b about 20000 times. Record the computational timing under the following 4 scenarios:

1. With CPU and A and b are single precisions (8-digit accurate).
2. With CPU and A and b are double precisions (16-digit accurate).
3. With GPU and A and b are single precisions (8-digit accurate).
4. With GPU and A and b are double precisions (16-digit accurate).

Rank the computational time from the fastest to the slowest.