Homework #4— DSC 520

Assigned: Monday, November 27, 2023
Due: Monday, December 11, 2023

This is a bonus homework worth up to +15 points on your previously lowest homework grade. Students looking to get an A+ need to put in a solid effort, roughly at least +10; this is in addition to the other requirements for an A+.

Reading & viewing: If you would like additional experience with MPI, please consider consulting http://mpitutorial.com/ and https://computing.llnl.gov/tutorials/mpi/

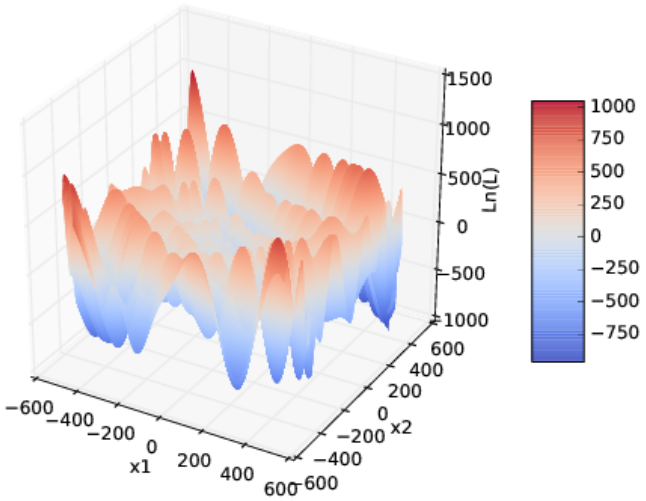
You must use C/C++ or Fortran to solve these problems. Other languages (Python, Matlab, Mathematica, gnuplot, etc) should be used only to visualize the data (ie making plots).

Report: Put all the figures and answers asked for in all the questions into a single well-organized PDF document (prepared using LaTeX, a Jupyter notebook, or something else). **Please email me your report before midnight by the due date**. Your report should also be uploaded to Bitbucket along with all of your code.

Code: On every homework you turn in, be sure to upload the computer code you used to generate your solutions to Bitbucket. Include all code used such as the C/C++ code to solve the problem, any code used to make figures or cary out analysis, and any Unix scripts. Please strive towards automation where there is one script that solves the entire problem: it will compile the code, run the code, and generate any figures or plots.

The main goal for this assignment is to write a parallelized program to solve a numerical optimization task. You should start with the code you wrote on the recent labs, modifying it to solve this problem. In its simplest form, numerical optimization is the study of finding a maximum or minimum of a function. Here we will consider a simple Monte Carlo method of finding the **minimum** of a secret function shown below and defined on a 2D box

$$x_1, x_2 \in [-512, 512]$$



Your work will be broken down into three problems

1. (problem 1) Write code to find the minimum of the secret function using MPI.
2. (problem 2) Carry out a weak scaling test on Anvil
3. (problem 3) Use Anvil to find the secret function's minimum.

Problem 1:

(a) (1 points) Write an MPI code that draws N random samples from a 2D box defined on $[-512, 512]^2$, and for each sample evaluates the secret function **secret_function** defined in the header secret_function.h. To include this function into your code, you will need to download the header (secret_function.h) and object (secret_function.o) files located in the class Bitbucket project. Include the header file secret_function.h at the top of your program like so:

```
#include <stdlib.h>
#include <stdio.h>
#include <mpi.h>
#include "secret_function.h"


int main(int argc, char **argv)
{

  double test = secret_function(100.0,100.0);
  printf("secret_function(100.0,100.0) = %lf\n",test);
  return 0;
}
```

and compile with a command that looks something like

```
>>> mpicc secret_function.o -Wall -std=c99 -o optimizer MyCode.c -lm
```

which tells the compiler to "link in" the object file (which I've built on my laptop). To check you have correctly added the secret function, please test your code by evaluating the secret function and comparing against my code:

```
f(100.0,100.0) = 1799.585607
```

Note: if you cannot get the above to work please let me know. I may need to build the secret_function.o object file specifically for your laptop's hardware.

(b) (2 points) If you launch 4 processes, each process will provide N samples of the secret function (with $4N$ total samples from your program). Have each process make $N = 2$ samples and print the function's minimum value. In your solution please demonstrate that your code is working correctly by showing its output. My code's output looks like (yours will look different since random numbers are involved):

```
Process 3, sample 1,  with f = 1.755e+03
Process 2, sample 1,  with f = 2.615e+03
Process 1, sample 1,  with f = 1.837e+03
Process 0, sample 1,  with f = 1.600e+03
Process 3, sample 2,  with f = 2.062e+03
Process 2, sample 2,  with f = 2.284e+03
Process 1, sample 2,  with f = 2.024e+03
Process 0, sample 2,  with f = 1.820e+03
Process 3 of 4 local min = 1.75536e+03
Process 2 of 4 local min = 2.28394e+03
Process 1 of 4 local min = 1.83671e+03
Process 0 of 4 local min = 1.60040e+0
```

(c) (2 points) Use the MPI minimum reduction to find the global minimum over all processes. Have the main process (ID 0), and for full credit *only this process*, report the minimum. Again, provide output from your code to show that it is working by using 4 processes.

(d) (2 points) Add a wall timer (you can use the function omp_get_wtime) to your code. Time two separate regions of your code: (i) The entire portion of your code from the first line to the last line, and (ii) the portion of code that implements the MPI reduction. Have only the main process (ID 0) report these timing numbers. Provide evidence that your code correctly outputs the timing numbers by showing an example output.

Problem 2:

(a) (3 points) Set $N = 10$ and run your code on Anvil using cores = 64 (1 node), 128 (1 node), 256 (2 nodes), 512 (4 nodes)*Important*: Please set the requested job time to 2 minutes (the job will take under 2 minutes):

```
#SBATCH -t 00:02:00
```

Notice that by fixing $N = 10$ you have performed a weak scaling test since the total number of random samples will be cores $\times$ N. Make a figure showing cores (x-axis) vs walltime (y-axis) for the entire program and just the reduction part. Comment on whether or not these timing numbers indicate the parallelization is efficient. Please compare to what would be perfect parallelization.

Problem 3

Now lets find the minimum of the secret function.

(a) (2 points) In part b you will run your code on Anvil using 256 cores (2 nodes). Here, in part a, you will estimate the required walltime. I found it takes about 10^7 random samples to correctly find the secret function's minimum. So we will use this number in the steps below. To estimate how long your code will need to run for, please consider the following:

1. How long does it take to run your code for 1 random sample in units of *hours*? Note that your answer will depend on the number of cores being used, as your code's scalability will affect the total time. In problem 2, you already ran your code using 256 cores (2 nodes) with $N = 10$. So take the time you found, convert it to hours, and divide by 10. Call this number x.
2. Estimate how long it will take to run your code for 10^7 random samples in hours? Call this number $y = x \times 10^7$; y is cputime (not walltime).
3. y is the number of cpu-hours you need – its the number of hours your code would need to run on a single core. For comparison, my code takes about 200 cpu-hours. On a single core this would require more than 8 days!
4. In your job script you will request the walltime, not cputime. To convert from cputime to walltime, you divide by the number of cores. Call this number $w = y/256$. **If you find the walltime to be larger than 4 hours please let me know. Something has gone wrong!**

For full credit, please document your work in your report.

(b) (2 points) Write an Anvil job script. In your job script set $N = 10^7/256$, since with 256 cores being use the total number of samples performed by your program will be 10^7 (recall that unlike OpenMP, the Monte Carlo for-loop is not shared; each process executes its own independent for-loop). Run your code on Anvil using 256 cores (2 nodes). Using your work from part a, request the required amount of walltime. I would add about 30 minutes to your request as a buffer. For example, if you found $w = 1.5$ then request 2 hours

```
#SBATCH -t 02:00:00
```

(c) (1 point) Provide a sample of the output from your program after its finished running. From your data, provide a guess for the secret function's minimum value is. You can simply copy and paste the output from your job's output file into your homework solution.