Problem 1. I have 100 CPU cores. My code is 0.4 % serial code. What is maximum speed-up I can obtain?

Proof. Speedup from parallel is limited by

$$S(p, N) = \frac{1}{f + \frac{(1-f)}{p}}$$

In our question. f = 0.4 and p =100. So,

$$S(100, N) = \frac{1}{0.4 + \frac{(1-0.4)}{100}} \approx 2.46$$

□

Problem 2.

- Go to your home directory on MIDWAY (e.g. /home/simonsch)
- Create a folder named Exercises day1
- in there, create a sub-folder denoted Fortran as well as a sub-folder named CPP
- go to your favourite programming language's folder.
- Write a little program in that language that reads in your name from the terminal and write "Hello YOURNAME, how are you".
- copy this file to your local laptop/desktop by scp.
- compile the code, create an executable that is called "hidiho.exec".
- Hard-code YOURNAME into your *.cpp/*.f90 file.
- Adjust a slurm job.sh file such that you can submit the executable in batch mode.

Proof.

- makedir exercises_day1
- cd exercises_day1
- makedir Fortran
- makedir CPP
- scp weinberga@midway1.rcc.uchicago.edu: /home/weinberga/exercises_day1/CPP/hello_name.cpp /Users/alexweinberg/Desktop/BootCamp2018/ProbSets/Comp/ProbSet4
- g++ hello_name.cpp -o hidiho.exec
- **NEED HELP MAKING SLURM SH FILE**

□

Listing 1: Code for Hello-name

```
#include <iostream>
using namespace std;

int main(){
  string name;
  cout << "Enter your name: ";
  cin >> name;
  cout << "Hello, " << name << "\tfrom Alex! " << endl;

  return 0;
}
```

Problem 3. Write a program in Fortran or CPP that reads arbitrary values a, b, c from the terminal (stdin) and prints the solution to the quadratic Equation.

$$ax^2 + bx + c = 0$$

Proof.

Listing 2: Quadratic Solver

```
#include <iostream>
#include <math.h>
#include <map>
#include <string>
using namespace std;


pair<float,float> solve_quad(float a, float b, float c)
{
  float x1 = (- b + sqrt(b * b - (4 * a * c))) / (2 * a);
  float x2 = (- b - sqrt(b * b - (4 * a * c))) / (2 * a);

  return make_pair<float,float>(x1, x2);
}

int main()
{
  cout << "\n-----------------------------------------------\n";
  float a, b, c;
  cout << "Let's solve a quadratic equation!\n f(x) = ax^2 + bx + c = 0\n";
  cout << "This only works for equations with real solutions :(\n";
  cout << "Enter a: ";
  cin >> a;
  cout << "Enter b: ";
  cin >> b;
  cout << "Enter c: ";
  cin >> c;

  pair<double,double> result = solve_quad(a,b,c);
  cout << "x = " << result.first << " or " << result.second << endl;

  cout << "-----------------------------------------------\n";
  return 0;
}
```

□

Problem 4. Compute pi and write a makefile

Listing 3: Estimate π

```
#include <iostream>
using namespace std; //no std:: prefix is needed!

static long num_steps = 1000000; double step;

int main ()
{
        int i;
        double x, pi, sum=0.0;

        step = 1.0 / (double) num_steps;
        for (i=0; i<num_steps; i++)
        {
          x = (i + 0.5)*step;
          sum = sum + 4.0 /(1.0 + x*x);
        }
        pi = step * sum;
        cout << "Pi = " << pi << endl;
        return 0;
}
```

Listing 4: Makefile for pi.cpp

```
all: pi.exec
#########################################################################

## Example 1
pi.exec : pi.cpp
        g++ pi.cpp -o pi.exec

#########################################################################

clean :
        rm -f *.exec
```

HOW DO I COMBINE MAKEFILES????

Listing 5: Submit batch to pi

```
#!/bin/bash
# job submission script to submit an MPI job to the sandyb partition on Midway1

# set the job name to hello
#SBATCH --job-name=pi
```

```
# send output to hello-world.out
#SBATCH --output=pi.out

# receive an email when job starts, ends, and fails
#SBATCH --mail-type=BEGIN,END,DAIL

#SBATCH --account=osmlab

# this job requests 1 core. Cores can be selected from various nodes.
#SBATCH --ntasks=1

# there are many partitions on Midway1 and it is important to specify which
# partition you want to run your job on. Not having the following option, the
# sandby partition on Midway1 will be selected as the default partition
#SBATCH --partition=sandyb

# Run the process
./pi.exec
```

Problem 5. Compute Pi using Monte Carlo.
Experiment with the number of random number you create (N = 100, 1,000, 10,000).
Run the code both in interactive as well as in batch mode.

Listing 6: MonteCarlo estimate pi

```
/*
A function to estimate pi. Uses the monte carlo method to draw random samples
(x,y) from [0,1]x[0,1] and checks if (x,y) is in unit circle.

estimate_pi = #inCircle / #totaldraws
*/

#include <random>
#include <stdio.h>

#include <iostream>
using namespace std;

bool inCircle(double x, double y)
{
        double z;
        z = x * x + y * y;
        return (z <= 1);
}


int main()
```

```
{
        unsigned seed_unif1 = 3; // unsigned means non-negative, declare seed

        std::default_random_engine generator_unif(seed_unif1);
        std::uniform_real_distribution<double> distribution_unif(0.0,1.0);

        int total_draws = 19000;
        int N_inCircle = 0;
        double randX;
        double randY;


// Generate (x,y) points
        for(int numbers = 1; numbers <= total_draws; numbers++)
        {
          randX = distribution_unif(generator_unif); // draw from unif
                randY = distribution_unif(generator_unif); // draw from unif

                bool inside = inCircle(randX, randY); // check in unit circle

                if (inside)
                                N_inCircle += 1;
        }

// Compute estimate of pi
        double estimate;
        estimate = 4 * (double) N_inCircle / (double)total_draws;
        cout << "My estimate of pi is: " << estimate << "\t with samples = " <<
           total_draws<<endl;
        cout << "Answer should be near: 3.141\n";
        cout << "Requires 19_000 samples to get first two decimals accurate\n";

  return 0;
}
```

WHY ONLY WORK ON LOCAL, NOT MIDWAY????

Problem 6. Run the project in midway

Proof. Hello □