Karin Metzgar
ASTE 546
HW 14

Part 1: Serial Code

A note about the HW14 PDF, in the screen shot you provided there is not a N-1 term in the final for loop which causes an error, it is correct in the code you provided but not in the screenshot. Just thought I'd point it out in case no one else did!

ASTE546_HW14_KM.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

[11]
```
%%writefile serial.cpp
#include <iostream>

int main() {
  const int N=11;
  double x[N],y[N];

  //set xs
  for (int i=0; i<N; i++) x[i] = i/(double)(N-1);

  //set ys
  for (int i=0; i<N; i++) y[i] = x[i] * x[i] * x[i];

  //integrate
  double sum = 0;
  for (int i=0; i<N-1; i++) sum += 0.5 * (y[i]+y[i+1]) * (x[i+1]-x[i]);

  std::cout <<"Integral = " << sum << std::endl;
  return 0;
}
```

```
Overwriting serial.cpp
```

```
%%bash
g++ -O2 serial.cpp -o serial
./serial
```

```
Integral = 0.2525
```

[ ] Start coding or generate with AI.

Part 2

```
[13] %%writefile mt.cpp
#include <iostream>
#include <thread>
using namespace std;

void kernel(int i, double *x, double *y) {
    y[i] = x[i]*x[i]*x[i];
}

int main() {
    const int N=11;
    double x[N],y[N];

    // set xs
    for (int i=0;i<N;i++) x[i] = i/(double)(N-1);

   // use mulithreading to compute values
   thread my_threads[N];
    for (int i=0;i<N;i++) my_threads[i] = thread(kernel,i,x,y);
    for (thread &t:my_threads) t.join();  // wait to finish

    // integrate
    double sum = 0;
    for (int i=0;i<N-1;i++) sum+=0.5*(y[i]+y[i+1])*(x[i+1]-x[i]);


    std::cout<<"MT Integral = "<<sum<<std::endl;
     return 0;
}
```

```
Writing mt.cpp
```

```
%%bash
g++ -O2 mt.cpp -o mt -lpthread
./mt
```

```
MT Integral = 0.2525
```

[ ] Start coding or generate with AI.

Part 3:

```
%%writefile mpi.cpp
#include <iostream>
#include <mpi.h>
using namespace std;
int main(int num_args, char *args[]) {
    MPI_Init(&num_args,&args);
    int my_rank;
    int N;
    MPI_Comm_rank(MPI_COMM_WORLD,&my_rank);
    MPI_Comm_size(MPI_COMM_WORLD,&N);
    // compute my dynamically
    double my_x = my_rank/(double)(N-1);
    double my_y = my_x*my_x*my_x;
    // if not root
    if (my_rank>0) {
        // send my x and y to root (rank 0)
        MPI_Send(&my_x,1,MPI_DOUBLE,0,777, MPI_COMM_WORLD);
        MPI_Send(&my_y,1,MPI_DOUBLE,0,888, MPI_COMM_WORLD);

}
else {
    // allocate memory to store results
    double *x = new double[N];
    double *y = new double[N];
    x[0] = my_x;    // store root's own data
    y[0] = my_y;
    for (int r=1;r<N;r++) {  // receive x and y from each rank
        MPI_Recv(&x[r], 1, MPI_DOUBLE, r, 777, MPI_COMM_WORLD,MPI_STATUS_IGNORE);
        MPI_Recv(&y[r], 1, MPI_DOUBLE, r, 888, MPI_COMM_WORLD,MPI_STATUS_IGNORE);
}
    // integrate
    double sum = 0;
    for (int i=0;i<N-1;i++) sum+=0.5*(y[i]+y[i+1])*(x[i+1]-x[i]);
    std::cout<<"MPI Integral = "<<sum<<std::endl;
delete[] x;
        delete[] y;
    }  // if root
    MPI_Finalize();
return 0;
}
```

```
Writing mpi.cpp
```

```
%%bash
mpic++ -O2 mpi.cpp -o mpi
mpirun --allow-run-as-root -np 11 ./mpi
```

```
--------------------------------------------------------------------------
There are not enough slots available in the system to satisfy the 11
slots that were requested by the application:

  ./mpi

Either request fewer slots for your application, or make more slots
available for use.
```

I got the “There are not enough slots” error if I used a number larger than 1, which resulted in sum = 0

I used the “oversubscribe” flag and was able to do more:

```
Overwriting mpi.cpp
```

```bash
%%bash
mpic++ -O2 mpi.cpp -o mpi
mpirun --allow-run-as-root --oversubscribe -np 2 ./mpi
```

```
MPI Integral = 0.5
```

```
[ ] Start coding or generate with AI.
```

```
Overwriting mpi.cpp
```

```bash
%%bash
mpic++ -O2 mpi.cpp -o mpi
mpirun --allow-run-as-root --oversubscribe -np 10 ./mpi
```

```
MPI Integral = 0.253086
```

```
[ ] Start coding or generate with AI.
```

c) Both the send and receive functions go away
The loop that has all process (other than root) send their data to root is not necessary because the Gather function serves this purpose
The “else” code for the root was split up, the memory needs to be allocation before calling the gather function, then the gather function is called, then the rest of the block.

+ Code + Text

```
#include <mpi.h>

using namespace std;

int main(int num_args, char *args[]) {
    // Initialize MPI
    MPI_Init(&num_args, &args);

    int my_rank;
    int N;

    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &N);

    // compute my dynamically
    double my_x = my_rank / (double)(N-1);
    double my_y = my_x * my_x * my_x;

    //allocate memory
    double *x = nullptr;
    double *y = nullptr;

    //define x and y in root?
    if (my_rank == 0) {
        x = new double[N];
        y = new double[N];
    }

    // Gather x and y from all processes to root
    MPI_Gather(&my_x, 1, MPI_DOUBLE, x, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Gather(&my_y, 1, MPI_DOUBLE, y, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    if (my_rank == 0) {
        // integrate
        double sum = 0;
        for (int i = 0; i < N-1; i++) sum += 0.5 * (y[i] + y[i+1]) * (x[i+1] - x[i]);

        cout << "MPI Integral = " << sum << endl;

        delete[] x;
        delete[] y;
    }

    MPI_Finalize();
    return 0;
}
```

```
Overwriting mpi.cpp
```

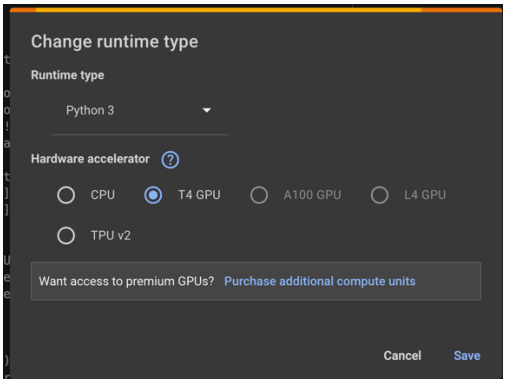
[29] 1s

```
%%bash
mpic++ -O2 mpi.cpp -o mpi
mpirun --allow-run-as-root --oversubscribe -np 11 ./mpi
```

```
MPI Integral = 0.2525
```

Part 4:

The code provided did not seem to work, and gave an answer of 0
I have T4 GPU selected



And I did need to remove the -arch=sm+37 because it generated an error

```
[8] %%writefile cuda.cu
        #include <iostream>
        using namespace std;
        // using float since GPUs generally run much faster with single precision
        __global__ void kernel(float *glob_x, float *glob_y, int N) {
            // compute my thread int
            int i = blockIdx.x * blockDim.x + threadIdx.x;
            if (i<N) {  // may have more threads than N if not even divisible by block size
                float x = i/(float)(N-1);
                float y = x*x*x;
                glob_x[i] = x;
                glob_y[i] = y;  // save into global memory
    } }
        int main() {
            const int N = 11;

            // allocate memory on the GPU
            float *dev_x,*dev_y;
            cudaMalloc(&dev_x,sizeof(float)*N);
            cudaMalloc(&dev_y,sizeof(float)*N);
            if (cudaGetLastError()!=cudaSuccess)
              cout<<"Error: "<<cudaGetErrorString(cudaGetLastError())<<endl;

            // allocate memory on the CPU
            float *x = new float[N];
            float *y = new float[N];


            // copy from GPU to CPU
            cudaMemcpy(x,dev_x,sizeof(float)*N,cudaMemcpyDeviceToHost);
            cudaMemcpy(y,dev_y,sizeof(float)*N,cudaMemcpyDeviceToHost);

            // integrate
            double sum = 0;
            for (int i=0;i<N-1;i++) sum+=0.5*(y[i]+y[i+1])*(x[i+1]-x[i]);
            std::cout<<"CUDA Integral = "<<sum<<std::endl;
            delete[] x;
            delete[] y;
            return 0;
    }
```

```
Overwriting cuda.cu
```

```
%%bash
nvcc cuda.cu -o cuda
./cuda
```

```
CUDA Integral = 0
```

However the following code worked, with some help from the internet, the //launch kernel portion was added and that got the expected answer

```
/ Launch kernel
    int blockSize = 256;
    int numBlocks = (N + blockSize - 1) / blockSize;  // Ceiling of
N/blockSize
    kernel<<<numBlocks, blockSize>>>(dev_x, dev_y, N);
```

```
// Synchronize the device
cudaDeviceSynchronize();
```

ASTE546_HW14_KM.ipynb

File Edit View Insert Runtime Tools Help

+ Code + Text

```
        glob_y[i] = y;  // Save into global memory
    }
}

int main() {
    const int N = 11;

    // Allocate memory on the GPU
    float *dev_x, *dev_y;
    cudaMalloc(&dev_x, sizeof(float) * N);
    cudaMalloc(&dev_y, sizeof(float) * N);
    if (cudaGetLastError() != cudaSuccess)
        cout << "Error: " << cudaGetErrorString(cudaGetLastError()) << endl;

    // Launch kernel
    int blockSize = 256;
    int numBlocks = (N + blockSize - 1) / blockSize;  // Ceiling of N/blockSize
    kernel<<<numBlocks, blockSize>>>(dev_x, dev_y, N);

    // Synchronize the device
    cudaDeviceSynchronize();

    // Allocate memory on the CPU
    float *x = new float[N];
    float *y = new float[N];

    // Copy from GPU to CPU
    cudaMemcpy(x, dev_x, sizeof(float) * N, cudaMemcpyDeviceToHost);
    cudaMemcpy(y, dev_y, sizeof(float) * N, cudaMemcpyDeviceToHost);

    // Integrate
    double sum = 0;
    for (int i = 0; i < N-1; i++)
        sum += 0.5 * (y[i] + y[i+1]) * (x[i+1] - x[i]);

    std::cout << "CUDA Integral = " << sum << std::endl;

    // Free memory
    delete[] x;
    delete[] y;
    cudaFree(dev_x);
    cudaFree(dev_y);

    return 0;
}
```

```
Overwriting cuda.cu
```

```
%%bash
nvcc cuda.cu -o cuda
./cuda
```

```
CUDA Integral = 0.2525
```