COSC3500/7502 Assignment : Parallel programming techniques

Summary : The goal of this assignment is to implement three different matrix multiply functions for three different hardware configurations; the CPU (AVX/openMP), the GPU (CUDA), and a cluster of two nodes (MPI). The matrices are real-only square matrices. The performance of your matrix multiplication imp benchmarked relative to a 'gold standard' reference implementation pro library (CPU and MPI), or CUBLAS (GPU), and marks assigned on the bas mentations are for a matrix of dimensions 2048×2048. The MPI implementatio sed implementation of *matrixMultiply.cpp*.

Rubric :

Marks	CPU (A	GPU (CUDA) 1 GPU	MPI 2 nodes, 4 cores each
7	2.3	4.9	1.4
6	16.5	8	8.3
5	36.2	24.8	18.5
4	64.3	43	31.6
3	121	100	60.8
2	1000 (and gives correct answer and job doesn't timeout)		
1	Compiles and runs to completion, but gives wrong answer		
0	Doesn't compile or wasn't submitted or timeout		

Table 1 : Marks versus performance of your implementation relative to the reference implementation (Intel MKL or CUBLAS). The values in the table indicate how many times longer your runtime is relative to the reference (lower values are better.) Marks are capped at 7 for each implementation.

Hardware :

Your final performance will be assessed on the *vgpu10-0* and *vgpu10-1* nodes of the *rangpur.compute.eait.uq.edu.au* cluster. However, please ***do not*** hammer those two nodes for debugging or it will create a logjam of jobs. All nodes on *rangpur* will have very similar, if not identical, performance for the CPU and GPU jobs, and for all but the most optimised MPI implementations. Only if your matrix multiply is highly optimised, will you start to notice the communication overhead between the nodes. This effect can be seen in Table 1, whereby the CPU speed is ½ the MPI speed for all but the fastest (grade 7) implementation. The MPI communication between the *vgpu10* nodes appears to be faster than between the *vgpu20/40* nodes, likely because the *vgpu10* nodes are simply virtualisations of the same physical hardware. For development, you can also submit jobs to *getafix.smp.uq.edu.au*.

The benchmarks :

To benchmark the code, a set of random unitary square real-only matrices are created, and then successively multiplied together to get the final solution, which is then checked to ensure it gives the correct answer to within some allowable floating point error, and is measured for speed relative to the reference implementations (Intel MKL or CUBLAS). The benchmark keeps multiplying matrices together, getting the result, and then multiplying the next matrix by the result from the previous matrix multiplication. i.e. it does a sequence of **A** = **A.B**. For the CPU and GPU implementations, this is relatively straightforward as the entire matrix multiplication occurs on the same machine in the same memory space. For MPI, each individual node will have access to it's own identical copy of the full set of matrices that need to be multiplied from the start. So there's no need for you to distribute that test data amongst the nodes, it will already be there. However you *will* need to make sure that all nodes maintain a copy of the current matrix product answer, as each node will need this full matrix so it can calculate the next matrix product. That is, each node can calculate it's own portion of the matrix multiply operation, but you will then have to make sure that all nodes end up with a full copy of the solution.

Turning on/off CPU, GPU, and/or MPI code :

If you wish to run your code on a machine that does not have a certain type of hardware available (e.g. no nvidia GPU, or MPI not installed), then you can completely remove those features by commenting out the '#define ENABLE_CPU', '#define ENABLE_GPU' and/or '#define ENABLE_MPI' lines in the *Assignment1_GradeBot.cpp*, as well as removing the relevant lines in the MakeFile (e.g. for no GPU, no nvcc, no c[illegible]h, -lcudart, -lcublas, matrixMultiplyGPU.cu, etc)

The only code files [illegible]ur final submission :

There are only 3 fi[illegible]change, ***matrixMultiply.cpp***, ***matrixMultiplyGPU.cu***, and ***matrixMultiplyMP[illegible]*** [illegible]e not allowed to use any header files or outside libraries, other than the [illegible]les (*matrixMultiply.h*, *matrixMultiplyGPU.cuh*, and *matrixMultiplyMPI*[illegible] not write to *stdout* or to file in the final submission.

The script that will be used to assign your final grade is *goslurm_COSC3500Assignment_RangpurJudgementDay*. However for the love of nvidia *PLEASE* do not just submit all your jobs using this script, as you will overwhelm the *vgpu10* nodes. Please use variations on the *goslurm_COSC3500Assignment_RangpurDebug* or *goslurm_COSC3500Assignment_GetafixDebug* scripts.

When calculating your final grade, as the final grade script will run on two nodes, with the CPU and GPU benchmarks run once on each node, the fastest node result (most favourable to you) will be used although in practice the values tend to be almost identical and vary by less than 0.2 marks.

You are welcome, and indeed encouraged, to debug your implementations on your own computer, but remember your code *must* compile and run successfully on the *rangpur* cluster, and ultimately it is the performance on those nodes that will set your marks.

Software interface and the GradeBot

Assignment1_GradeBot.cpp is a remorseless marking machine. It can't be bargained with. It can't be reasoned with. It doesn't feel pity, or remorse, or fear, and it absolutely will not stop, ever, until your COSC3500 assignment has been assigned a mark out of 21.

The GradeBot runs your benchmarks and assigns your marks. You can use it to get instant feedback.


```
./Assignment1_GradeBot {matrix dimension} {threadCount}
{runBenchmarkCPU} {runBenchmarkGPU} {runBenchmarkMPI} {optional
integer} {optional integer}…
```

For example,

```
./Assignment1_GradeBot 2048 4 1 0 1
```

Would run benchmarks of the 2048×2048 matrix multiply routines, using 4 threads per node for the CPU and MPI, but no GPU benchmark would be performed.

It is possible to supply an additional optional list of integer flags after the end of those 5 parameters that will be passed through to the *matrixMultiply* routines and can be used for debugging and tweaking.

For example,

```
./Assignment1_GradeBot 2048 4 1 0 1 64 128 256
```

Would pass the array [64 128 256] as the *args* parameter (*argCount*=3) to the *matrixMultiply* routines. How that array of integers is interpreted is entirely up to you, but remember they will not exist when the final *JudgementDay* script is called.

Text output :

The *Assignment1_GradeBot* will output to *stdout*, as well as individual text files for each benchmark on each node *COSC3500Assignment_{benchmark type}_{node}.txt*.

The text files will include 6 columns

Info. : {CPU|GP[illegible]piWorldSize}|{threadCount, gpuID, or mpiRank},{tota[illegible] number of physical [illegible]PUs or mpiWorldSize]({hardware run on, e.g. CPU name, GPU name, or node[illegible]

N : Matrix dimensi[illegible]

Matrices/second (M[illegible] [illegible] matrix multiplies performed per second by the reference software MKL (CP[illegible]PU)

Matrices/second (You) : The number of matrix multiplies performed per second by your implementation.

Error : The sum of squares different between your final output matrix, and the reference implementation. Some small (<1e-7) floating point error is allowed, but larger errors are counted as a failed implementation.

Grade : The total number of marks assigned for this result.

Final Submission :

Your submission _must_ include the following files all zipped together, in a file named {your 8 digit student number}.zip

```
matrixMultiply.cpp

matrixMultiplyGPU.cu

matrixMultiplyMPI.cpp
```


Plus a zip file (within the master zip file containing the matrixMultiply files), slurm.zip, containing all your slurm job output files (slurm-xxxx.out).

If you have not implemented any of the required files (e.g. you've only implemented the CPU, but not the GPU, or MPI), then just submit the original blank files provided on blackboard.

The final submission *must* strictly have this format. Any deviation will not be forgiven by the cold uncaring scripts associated with *Assignment1_GradeBot.cpp.*