

P145484 DCC Assignment 2

by Cerys Bethan Nickels

FILE	2645_CERYS_BETHAN_NICKELS_P145484_DCC_ASSIGNMENT_2_113617_490493847.DOCX		
TIME SUBMITTED	11-MAY-2018 11:21PM (UTC+0100)	WORD COUNT	1131
SUBMISSION ID	89031505	CHARACTER COUNT	9288

Portfolio Report

Name: Cerys Bethan Nickels

Student Number: P145484

Course: Software Engineering

Module Name: Distributed &
Cluster Computing (Level 6)

Module Code: M6X01093

Assignment Number: 2

Portfolio Report Table of Contents

Table of Contents

The Development Process	3
Identifying Areas for Improvement	3
Threading (C++/Boost)	4
Message Passing Interface	5
Timing Results	6
Limitations of Each Method	6
Threading (C++/Boost)	6
Message Passing Interface	7
CUDA	7
Conclusion	8
References	8

Portfolio Report Table of Figures

Figure 1: Identifying Areas for Improvement.....	3
Figure 2: Function Run Time Information.....	3
Figure 3: Threading - SetCD Method.....	4
Figure 4: Threading Main Method	5
Figure 5: Threading - Additional Method	5
Figure 6: MPI Changes.....	5

The Development Process

Identifying Areas for Improvement

```
input_10000.txt
input_200.txt
input_2000.txt
input_20000.txt
input_500.txt
input_5000.txt

Enter file name to read:
input_200.txt
Time taken to complete "File Read": 0 milliseconds.
Time taken to complete "Initialising variables": 0 milliseconds.
Time taken to complete "Setting A": 0 milliseconds.
Time taken to complete "Setting B": 0 milliseconds.
Time taken to complete "Inverting A": 23 milliseconds.
Time taken to complete "Inverting B": 26 milliseconds.
Time taken to complete "Setting ADash and BDash": 60 milliseconds.
Time taken to complete "Setting C and D": 73 milliseconds.
Time taken to complete "Setting E": 43 milliseconds.
Time taken to complete "Setting Result": 0 milliseconds.
Time taken to complete "Computation": 195 milliseconds.
Time taken to complete "File write": 80 milliseconds.
Time taken to complete "Deallocating variables": 1 milliseconds.
Time taken to complete "Entire Program": 287 milliseconds.
Complete. Press <Enter> to exit.
```

Figure 1: Identifying Areas for Improvement

To identify the areas where the greatest areas of improvement are required, the built-in performance tool analyser in Microsoft Visual Studio 2015 was used. This was tested with the input_200.txt file to quickly identify the classes and functions to optimise. This tool is beneficial for all the three methods of optimising the code, as this information can be used to reduce the execution time as much as possible. The longest time is caused by the 'Computation' class at 195 milliseconds, followed by 'File write' at 80ms, and 'Setting C & D' at 73ms. This can be broken up by analysing the functions for CPU intensity and time:

Function Name	Total CPU (%) ▼	Self CPU (%)	Total CPU (ms)	Self CPU (ms)
▲ _scrt_common_main_seh	100.00%	0.00%	127	0
▲ main	100.00%	0.00%	127	0
▲ ComputationEngine::SetCD	39.37%	37.80%	50	48
▸ ScopedTimer::~ScopedTimer	0.79%	0.00%	1	0
▸ std::operator<< <<std::char_traits<char> >	0.79%	0.00%	1	0
[External Code]	0.79%	0.79%	1	1
▲ ComputationEngine::SetE	32.28%	30.71%	41	39
▸ ScopedTimer::~ScopedTimer	0.79%	0.00%	1	0
[External Code]	0.79%	0.79%	1	1
▸ ComputationEngine::SetABDash	22.05%	0.00%	28	0
▸ FileIO::WriteFile	2.36%	0.00%	3	0
▸ ComputationEngine::Deallocate	0.79%	0.00%	1	0
▸ ComputationEngine::SetB	0.79%	0.00%	1	0
▸ ComputationEngine::SetA	0.79%	0.00%	1	0
▸ ScopedTimer::~ScopedTimer	0.79%	0.00%	1	0
▸ FileIO::ReadFile	0.79%	0.00%	1	0

Figure 2: Function Run Time Information

Threading (C++/Boost)

Due to the times calculated by the performance tool analyser, the first class threaded was the 'Computation Engine'. The method 'SetCD' had the longest runtime, and was parallelised by using an additional array to reduce time.

```
void ComputationEngine::SetCD(void)
{
    ScopedTimer *st = new ScopedTimer("Setting C and D");
    // Create b, k1 and k2, set nC and nD to 0, and create an array
    size_t b, k1, k2;
    memset(nC, 0, sizeof(nC));
    memset(nD, 0, sizeof(nD));
    double* arrayMult = new double[arrSize];

    // Loop through the elements to populate the matrices with a number
    for (int L = 0; L < arrSize; L++)
    {
        arrayMult[L] = L * arrSize;
        for (size_t I = 0; I < arrSize; I++)
        {
            b = arrayMult[L] + I;
            k2 = arrayMult[I] + L;
            k1 = arrayMult[L] + (k2-I) / arrSize;
            for (size_t J = 0; J < arrSize; J++)
            {
                nC[b] += nA[k1] * nB[k2]* arrSize;
                nD[b] += nADash[k1] * nBDash[k2]* arrSize;
            }
        }
    }

    // Delete the scoped timer and the array to prevent memory leaks
    delete(st);
    delete arrayMult;
}
```

Figure 3: Threading - SetCD Method

The main method was also updated to speed up the overall computation and file write time, by adding in threading (with a thread manager as an array), and an additional method for each thread to run. They are deleted at the end of the main, with the exception of the thread array, as it considers itself to still be in scope, despite the threads being joined.

```

ScopedTimer *cp = new ScopedTimer("Computation");
double** douThreadArray = new double*[noOfThreads];
ComputationEngine **ce = new ComputationEngine*[noOfThreads];

int multOuterLoop = 0;
for (int outerLoop = 0; outerLoop < noOfThreads; outerLoop++)
{
    douThreadArray[outerLoop] = new double[size/noOfThreads];
    multOuterLoop = size / noOfThreads * outerLoop;

    for (int loop = 0; loop < (size / noOfThreads); loop++)
    {
        douThreadArray[outerLoop][loop] = arr[multOuterLoop + loop];
    }
    ce[outerLoop] = new ComputationEngine(&douThreadArray[outerLoop], size / noOfThreads);

    threadManager[outerLoop] = thread(RunCompEngine, &ce[outerLoop]);
}

for (int joinLoop = 0; joinLoop < size/noOfThreads; joinLoop++)
{
    threadManager[joinLoop].join();
}
delete(cp);

```

Figure 4: Threading Main Method

```

void RunCompEngine(ComputationEngine **compEngineIn)
{
    ComputationEngine *ce = *compEngineIn;
    ce->Initialise();
    ce->SetA();
    ce->SetB();
    ce->SetABDesh();
    ce->SetE();
    ce->SetResults();
}

```

Figure 5: Threading - Additional Method

Message Passing Interface

The MPI project was created using the original project, with the 'SetCD' method copied from the threading project. The additional variables for the MPI rank and size were created and the relevant methods were used to create the MPI project.

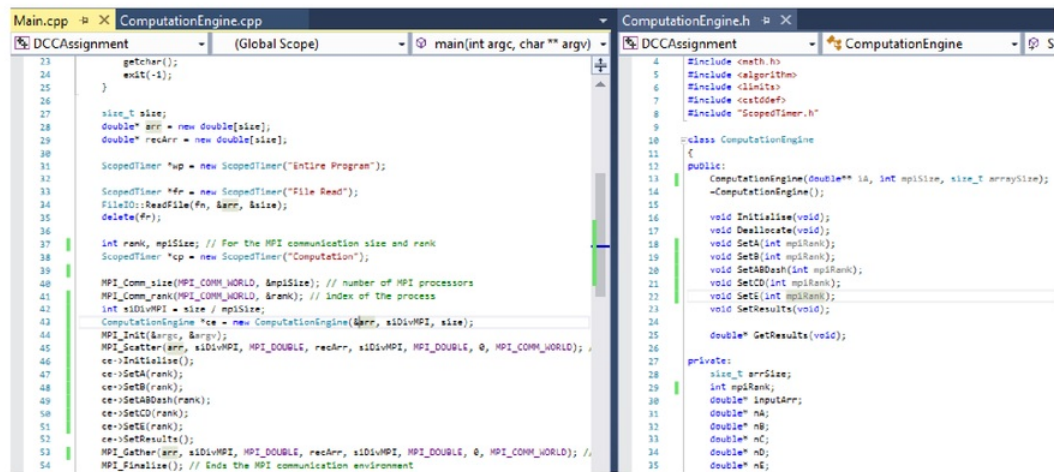


Figure 6: MPI Changes

Timing Results

	Original Program	Threading
File read	0 milliseconds	0 milliseconds
Initialising variables	0 milliseconds	0 milliseconds
Setting A	0 milliseconds	0 milliseconds
Setting B	0 milliseconds	0 milliseconds
Inverting A	13 milliseconds	0 milliseconds
Inverting B	12 milliseconds	0 milliseconds
Setting A' and B'	29 milliseconds	20 milliseconds
Setting C and D	48 milliseconds	35 milliseconds
Setting E	36 milliseconds	0 milliseconds
Setting results	0 milliseconds	0 milliseconds
Computation	126 milliseconds	40 milliseconds
File write	22 milliseconds	4 milliseconds
Deallocating variables	0 milliseconds	0 milliseconds
Entire program	156 milliseconds	90 milliseconds

Limitations of Each Method

Threading (C++/Boost)

There are many limitations to threading, including the following:

Security issues can arise as threads within a program share the same data, therefore unknown threads may overwrite the existing data [1]. This is similar to when synchronisation issues arise, caused when two or more processes are executed simultaneously. These can interact by using the

same resources, causing race conditions. This can be resolved by the use of a mutex or semaphore [2].

Slowing down or blocking the program occurs when the processor is overloaded with threads, or if a single kernel thread stops, such as the main thread of execution [1]. This reduction in execution speed can often be due to rapid construction and synchronisation of threads, which are CPU/memory intensive.

Existing code often requires significant re-architecting to allow multithreading to exist. This includes the removing of static variables, and replacing any function calls, or other code, that is not thread-safe. These changes must be thoroughly tested, increasing the cost of the application [3, Ch. 1-2], [4].

Threading increases the complexity of these applications, making them harder to write. This requires programming experience to code these types of applications effectively. Bad design and programming of threads can lead to 'thread starvation', which occurs when a thread has not been allocated enough CPU time [5].

There is a decrease in the ease of replicating, identifying, and verifying errors within the code source. This can be due to unpredictable results, which are often timing related, which are less common in single-threaded applications. This causes problems during the debugging stages, as these may not be identified previously, or even during this or the testing stages.

Message Passing Interface

There are several limitations to MPI:

1. Changes are required from serial to parallel versions of code, which are required for effective MPI code. These changes, along with MPI being harder to debug than other communication libraries, is sometimes not worth the effort of the complex code changes required.
2. When new versions are released, there is a need to ensure all workstations/ clusters are up to date with these changes, and are all running the same version [3, Ch. 1-2], [4].
3. Cluster computing and MPI require specialist hardware to implement, and its performance is limited by the communication network between the nodes [3, Ch. 1-2], [4], [6].

CUDA

This method of reducing execution time has a major limitation, in which it requires a compatible NVIDIA graphics card, with a compatible version of CUDA installed for operations.

Conclusion

In conclusion, I believe that the use of threading is the best for speed, using a single device. If there are multiple, then MPI can be utilised to the best of its capabilities, providing it is set up correctly. Threading is better utilised for the smaller sized input values, due to its small start-up costs, compared to the use of MPI, where the high start-up costs, and the queue on the service can prevent the results from being accessed quickly. Threading is also more versatile, and can easily be expanded with more threads (up to a point), and it can also be used within MPI to speed up the tasks.

References

- [1] Admin, 'Advantages and disadvantages of multi-threading systems', *IT Release*, 2018. [Online]. Available: <http://www.itrelease.com/2017/12/advantages-disadvantages-multi-threading-systems/>. [Accessed: 11-May-2018].
- [2] Microsoft, 'Synchronization Objects', *Microsoft*, 2018. [Online]. Available: [https://msdn.microsoft.com/en-us/library/windows/desktop/ms686364\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms686364(v=vs.85).aspx). [Accessed: 11-May-2018].
- [3] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message Passing Interface*, 2nd ed. Cambridge, MA: MIT Press, 1999.
- [4] Dartmouth College, 'Pros and Cons of OpenMP and MPI', *Dartmouth College*, 2011. [Online]. Available: https://www.dartmouth.edu/~rc/classes/intro_mpi/parallel_prog_compare.html. [Accessed: 28-Dec-2017].
- [5] J. Jenkov, 'Java Concurrency / Multithreading Tutorial', *Jenkov.com*, 2014. [Online]. Available: <http://tutorials.jenkov.com/java-concurrency/index.html>. [Accessed: 20-Jan-2018].
- [6] T. Bashford, K. Donne, A. Marotin, and A. Al-Hussany, 'Parallelisation Techniques for the Dual Reciprocity and Time-Dependent Boundary Element Method Algorithms', *Int. J. Comput. Methods Exp. Meas.*, vol. 5, no. 3, pp. 395–403, 2017.

P145484 DCC Assignment 2

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

51 /100

Instructor

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

GRADING FORM: SMU GENERIC_GJ

CERY'S BETHAN NICKELS

51

RESEARCH-INFORMED LITERATURE



A reasonable number of sources used, however a chunk of material comes directly from the first assignment! This is dangerously close to self-plagiarism!

2

KNOWLEDGE AND UNDERSTANDING

- | | | |
|---|---|-----------|
|  | Reasonable evidence of knowledge and understanding evident throughout the report. A tad brief overall, however. | 14 |
|---|---|-----------|

ANALYSIS

- | | | |
|---|---|----------|
|  | A good, scientific approach taken. Evidence of synthesis, with valid conclusions drawn. | 8 |
|---|---|----------|

PRACTICAL APPLICATION AND DEP.

- | | | |
|--|--|-----------|
|  | A fairly good implementation, though far less developed than it could've been - it'd be nice to see a more robust parallel implementation. | 24 |
|--|--|-----------|

SKILLS FOR PROFESSIONAL PRAC.

- | | | |
|---|---|----------|
|  | A fairly well-written report, nicely presented. | 3 |
|---|---|----------|