ELEN4020A: Data Intensive Computing Laboratory Exercise No 2

Kopantsho Mathafa (849038) Chizeba Maulu (900968) James Phillips (1036603)

March 17, 2019

Abstract

This report presents a summarized description of the solutions developed to solve the problem proposed in the lab brief. The solutions developed involve the use of OpenMP and Pthread, these were used to solve the problem of transposing a matrix using parallel programing methods. The computation times of all methods developed can be found in the table of results. Pseudo code to each method can also be found within the report.

I. Introduction

Threads are independent streams of instructions which can be scheduled to execute, independently and asynchronously, by the operating system. Optimizing performance and efficiency in high performance computing environments are the primary reasons behind threads being implemented and used.

The objective of this lab is to write a program which implements matrix transposition using parallel programming methods. The preferred language used is C++.

This report discusses the use of threads in the solution to solving the problem of matrix transposition. The analysis technique involves the use of timing sequences to compare the speed and performance of the different methods presented in this report. A table of results of results can be found in the below.

II. Matrix Transposition Using PThreads

PThreads or POSIX Threads,

III. Matrix Transposition Using OpenMP

OpenMP or Open Multi-Processing, is a package which forms part of the GCC compiler in Linux environments. This package supports shared memory multiprocessing. Shown below is pseudo illustrating the use of OpenMP to transpose a matrix using parallelism. OpenMp is used to divide iterations of loops into threads which can be executed independently.

A. OpenMP Pseudocode

IV. Critical Analysis and Future Recommendations

$N_0 = N_1$	**Basic**	**Pthreads**		**OpenMP**		
		Diagonal	**Blocked**	**Naive**	**Diagonal**	**Blocked**
128						
1024						
2048						
4096						

V. Conclusion

Appendix

```
function CREATEARRAY($dimension$)
    for $row = 0$; $row \leftarrow dimension$ do
        for $column = 0$; $column \leftarrow dimension$ do $matrix[row][column] = rand()\%21$
        end for
    end for
end function
```

```
function RANK2TENSORADD($A, B, rank$)
    for $row = 0$; $row \leftarrow rank$ do
        for $column = 0$; $column \leftarrow rank$ do $result[row][column] = A[row][column] + B[row][column]$
        end for
    end for
end function
```

```
function RANK2TENSORMULT($A, B, rank$)
    for $row = 0$; $row \leftarrow rank$ do
        for $column = 0$; $column \leftarrow rank$ do
            for $iterator = 0$; $iterator \leftarrow rank$ do $result[row][column] + = A[row][iterator] + B[iterator][column]$
            end for
        end for
    end for
end function
```

```
function PRINTFUNC($matrix, size$)
    for $row = 0$; $row \leftarrow size$ do
        for $column = 0$; $column \leftarrow size$ do $print(matrix[row][column])$
        end for
    end for
end function
```

```
function RANK3TENSORMULT($A$,$B$,$rank$)
    for $depth = 0$; $depth \leftarrow rank$ do
        for $row = 0$; $row \leftarrow rank$ do
            for $column = 0$; $column \leftarrow rank$ do
                for $iterator = 0$; $iterator \leftarrow rank$ do $result[row][column][depth] + = A[row][iterator][depth] + B[iterator][column][depth]$
                end for
            end for
        end for
    end for
end function
```

```
function RANK2TENSORADD($A, B, rank$)
    for $depth = 0$; $depth \leftarrow rank$ do
        for $row = 0$; $row \leftarrow rank$ do
            for $column = 0$; $column \leftarrow rank$ do
                $result[row][column][depth] = A[row][column][depth] + B[row][column][depth]$
            end for
        end for
    end for
end function
```