Kwame Nkrumah University of Science and Technology

College of Science
Faculty of Physical and Computational Sciences
Department of Mathematics

Project Report - Part 2

MATH 458 - Scientific Computing II

August 12, 2024

Tei Mensah Andrews - 4170120

1 Introduction

1.1 Overview

This report discusses the results, methodology and implementation of the work done in parallel pro- gramming project for Scientific Computing II using C Programming. The project contains reviewing algorithms that can be used to find the result of a vector-vector multiplication for dense vectors and implementing them for parallel running with varying number of processes in C programming and OpenMPI. Since the solution for the algorithm for dense vectors involves a lot of iterative com- putations, the process are timed for varying number of processes and their results are plotted for analysis.

1.2 Concepts

Vector-vector Multiplication Vector-vector Multiplication is the same as dot product; is the multiplication of two vectors which results in a scalar number that gives details about the behaviour of two vector towards each(eg. the angle between them). Its results is found by pairwise multiplication of elements of both vectors and subsequently adding them.

Parallelization

As the size of datasets and the complexity of calculations increase, the sequential execution of vector- vector multiplication can become a performance bottleneck. In today's computing landscape, where CPUs are equipped with multiple cores, it is essential to harness the power of parallel processing to achieve optimal performance. Parallelization allows multiple calculations to be performed simulta- neously, leading to faster computation times and improved overall efficiency.

2 Algorithms

2.1 vector-vector multiplication

Mathematical Description

Input data: Two of one-dimensional numeric arrays of equal size.

Output data: A single numeric value of the sum of the pairwise products of corresponding elements of both arrays.

For two one-dimensional arrays A[i] and B[i] each of size n, the dot product can be expressed mathematically as:

$$A \cdot B = \sum_{i=1}^{n} A[i]B[i] \tag{1}$$

Complexity

The algorithm has linear complexity, That is $O(N)$ for arrays of length N.

Algorithm 1 Vector-vector Multiplication

```
1: Data: $A[N],\ B[N]$
2: Result: $R \leftarrow 0$
3: for $i = 0;\ i < N;\ i++$ do
4:     $R \leftarrow R + (A[i] * B[i])$
5: end for
```

2.2 Matrix-Vector Multiplication

Mathematical Description

Input data: A two-dimensional array of a size M × N and a one-dimensional array of size N.

Output data: A one-dimensional array of size.

Given a two-dimensional array A[i][j] of size m × n, a one-dimensional array B[j] of size n and a one-dimensional results array C[i] the matrix-vector multiplication can be expressed mathematically as:

$$C[i] = \sum_{i=1}^{n} A[i][j]\, B[j], \quad \forall\ i \in [1, m] \tag{2}$$

Complexity
The algorithm has quadratic complexity, that is $O(N^2)$ for arrays of length N.

Algorithm 2 Matrix-vector Multiplication

```
1: Data: $A[M][N],\ B[N]$
2: Result: $C[N]$
3: for $i = 0;\ i < M;\ i++$ do
4:     $C[i] \leftarrow 0$
5:     for $j = 0;\ j < N;\ i++$ do
6:         $C[i] \leftarrow C[i] + (A[i][j] * B[j])$
7:     end for
8: end for
```

2.3 Matrix-Matrix Multiplication

Mathematical Description
Input data: Two two-dimensional array of one of size M × N and the other of size N × P.

Output data: A two-dimensional array of size M × P.

Given two two-dimensional array A[i][j] of size m × n, B[i][j] of size n × p and a two-dimensional results array C[i][j] of size m × p the matrix-vector multiplication can be expressed mathematically as:

$$C[i][j] = \sum_{k=1}^{n} A[i][j]\, B[j][i], \quad \forall\ i \in [1, m] \tag{3}$$

Complexity
The algorithm has cubic complexity, that is $O(N^3)$ for arrays of length N.

Algorithm 3 Matrix-vector Multiplication

```
1: Data: $A[M][N],\ B[N][P]$
2: Result: $C[M][P]$
3: for $i = 0;\ i < M;\ i++$ do
4:     for $j = 0;\ j < P;\ j++$ do
5:         $C[i][j] = 0$
6:         for $k = 0;\ k < N;\ k++$ do
7:             $C[i][j] \leftarrow C[i][j] + (A[i][k] * B[k][j])$
8:         end for
9:     end for
10: end for
```

3 Implementation

Vector-vector Multiplication

The following is the code for a C Programming implementation for a function that does vector-vector multiplication.

```
double mpi_vector_vector ( int rank , int local n , double   vecA   , double
// declare local variables
int root = 0;
double local_result = 0 , global_result = 0 ;
// vector multiplication for every rank
for( int i = 0 ; i < localn ; i++){
    local_result+= ( vecA[i]     vecB[i] ) ;
}
// complil e results in root
if ( rank == root ){
    global_result = local_result ;

MPI_Reduce ( MPI IN PLACE,(void     )& global_resul t , 1,MPI_DOUBLE,MPI_SUM, roo
}else {
MPI_Reduce ((void *)& local_result , ( void     )& global_result , 1 , MPI_DOUBLE
}
MPI_Barrier (MPI_COMM_WORLD) ;
if ( rank == root )
return global_result ;
}
```

4 Test and Results

A vector of size 2.510^8 was fed to the code and the run time for varying number of processes was taken. From table 1 we can see that as the number of cores increase the run time decrease steadily for the vector-vector multiplication until after 4 core where the time become roughly uniform.

N	Core	V.V/s
250000000	1	0.000012
250000000	2	0.000016
250000000	3	0.000018
250000000	4	0.000018
250000000	5	0.000018
250000000	6	0.000018
250000000	7	0.000018
250000000	8	0.000018

Table 1: Run time for processes with different N values

5 Analysis

- Vector-vector multiplication: The computational time for its algorithm for dense matrix steadily decreases as the number of process increase. Hence and inverse relationship is observed.

- Speed up time for each core was calculate and results is shown in table 2.

N	Core	V.V/s	Speed Up time
250000000	1	0.984012	0.984012
250000000	2	0.600263	1.639301
250000000	3	0.424152	2.319951
250000000	4	0.354092	2.778973
250000000	5	0.430506	2.285710
250000000	6	0.430907	2.283583
250000000	7	0.330126	2.980716
250000000	8	0.375263	2.622193

Table 2: Run time for processes with different N values

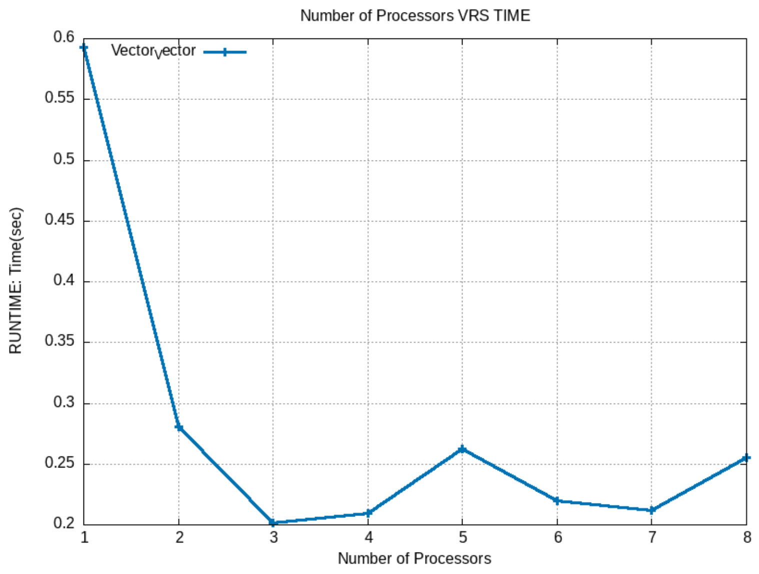

Figure 1: Number of row/col(N) Against Time(sec)

6 Conclusion

The presented run time of the algorithm for Vector-vector using the different number of cores shows their efficient in a parallel implementation. The results show that the algorithm for exhibits a stead decrease in run time for increasing Number of core hence more process increase the efficiency of the algorithm as expected.