KWAME NKRUAMH UNIVERSITY OF SCIENCE AND TECHNOLOGY, KUMASI

COLLEGE OF SCIENCE

FACULTY OF PHYSICAL AND COMPUTATIONAL SCIENCES
DEPARTMENT OF MATHEMATICS

PROJECT REPORT: PART ONE (SERIAL)

Contents

1 Introduction

This paper details the work done in the serial programming project for Scientific Computing II utilizing C Programming, as well as the results, approach, and implementation. The project comprises evaluating and developing methods for dense matrices and vectors that can be used to find the outcome of vector-vector multiplication, vector-matrix multiplication, and matrix-matrix multiplication. Because these methods for dense matrices and vectors need a large number of repeated computations, these procedures are timed for varied numbers of rows and columns of a matrix, and their results are presented for analysis.

2 Background Concepts

2.1 Vector-Vector Multiplication

Dot product is the multiplication of two vectors that results in a scalar integer that contains details about the behavior of two vectors towards each other (e.g. the angle between them). Its output is obtained by multiplying the elements of both vectors pairwise and then adding them.

2.2 Matrix-Vector Multiplication

Matrix-vector multiplication is a fundamental operation in computational mathematics that is widely employed in scientific and engineering applications. This procedure includes multiplying a 2D matrix by a 1D vector, resulting in a new vector. The matrix-vector multiplication process can be thought of as a linear combination of the matrix's columns, with the vector's components determining the results given to each column.

2.3 Matrix-Matrix Multiplication

Matrix-matrix multiplication is a fundamental operation in linear algebra and numerical computing, widely employed in various scientific and engineering fields. This operation involves the multiplication of two matrices, resulting in a new matrix. To multiply two matrices, the number of columns in the first matrix must equal the number of rows in the second matrix.

3 Algorithms and Mathematical Descriptions

3.1 Vector-Vector Multiplication

3.1.1 Mathematical Description

Input: Pair of one-dimensional arrays containing numerical values, both having the same size. Output: A single numerical result obtained by summing the element-wise products of the corresponding elements from the arrays. For two arrays, A[i] and B[i], each containing 'n' elements, the dot product can be mathematically represented as follows:

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

3.1.2 Complexity Analysis

The algorithm demonstrates linear complexity, denoted as $O(N)$, where 'N' represents the number of elements in each array. The computational effort increases linearly with the size of the arrays.

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: $\quad R \leftarrow R + (A[i] * B[i])$
5: **end for**

3.2 Matrix-Vector Multiplication

3.2.1 Mathematical Description

Input: A two-dimensional array of size (M × N) and a one-dimensional array of size (N × 1).

Output: A one-dimensional array of size (M × 1).

Given a two-dimensional array $A[i][j]$ of size m × n, a one-dimensional array B[j] of size n × 1 and a one-dimensional results array C[i] the matrix-vector multiplication can be expressed mathematically as:

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

3.2.2 Complexity Analysis

The algorithm demonstrates quadratic complexity, denoted as $O(N^2)$, where 'N' represents the number of elements in each array. 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: $\quad C[i] \leftarrow 0$
5: $\quad$ **for** $j = 0;\ j < N;\ i++$ **do**
6: $\quad\quad C[i] \leftarrow C[i] + (A[i][j] * B[j])$
7: $\quad$ **end for**
8: **end for**

3.3 Matrix-Matrix Multiplication

3.3.1 Mathematical Description

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

Output: 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:

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

3.3.2 Complexity Analysis

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: $\quad$ **for** $j = 0;\ j < P;\ j++$ **do**
5: $\quad\quad C[i][j] = 0$
6: $\quad\quad$ **for** $k = 0;\ k < N;\ k++$ **do**
7: $\quad\quad\quad C[i][j] \leftarrow C[i][j] + (A[i][k] * B[k][j])$
8: $\quad\quad$ **end for**
9: $\quad$ **end for**
10: **end for**

4 Implementation

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

```
double vector_vector (int m, int n, double *vecA, double *vecB){
        double results = 0;
        for (int i = 0; i < m; i++){
                results += (vecA[i]*vecB[i]);
        }
        return results;
}
```

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

```
void matrix_vector(int m, int n, double *matA, double *vecB, double *matC){
        for (int i = 0; i < m; i++){
                for (int j = 0; j < n; j++){
                        matC[i] += matA[i*n+j]*vecB[j];
                }
        }
}
```

Matrix-Matrix Multiplication
The following is a code for a C Programming implementation for a function that does Matrix-matrix multiplication.

```
void matrix_matrix(int m, int n, double *matA, double *matB, double *matC){
        for (int i = 0; i < m; i++){
                for (int j = 0; j < n; j++){
                        for (int k = 0; k < m; k++){
                                matC[(i*n)+j] += matA[(i*n)+k]*matB[(k*n)+j];
                        }
                }
        }
}
```

5 Test and Results

The code was fed arbitrary vectors and square matrices, and the execution duration was varied by the number of rows and columns. Table 1 shows that as the number of N increases, the run increases linearly for vector-vector multiplication, quadratically for matrix-vector multiplication, and cubically for matrix-matrix multiplication, confirming their complexity.

N	Core	V.V/s	M.V/s	M.M/s
1000	1	0.000012	0.007973	2.425972
2000	1	0.000040	0.030380	43.075583
3000	1	0.000046	0.083248	183.039097
4000	1	0.000065	0.144165	513.050412
5000	1	0.000090	0.234529	1056.531317

Table 1: Run time for processes with different N values

6 Analysis

- Vector-vector multiplication: Because its approach for dense matrices has a very short computing time, serial computation is appropriate for it.

- Matrix-vector multiplication: The algorithm's run time is reasonable, falling in between matrix-matrix and vector-vector. A serial implementation of the procedure may be insufficient for extremely dense matrices and vectors.

- Matrix-matrix multiplication: The algorithm's run time rapidly grows as the size of the matrices increases. As a result, for dense matrices, parallel implementation of its approach is advised..

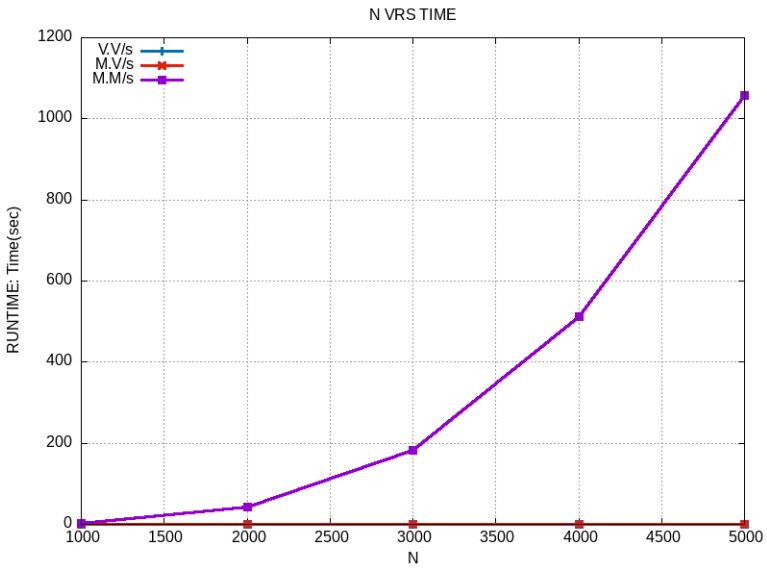

Figure 1: N Against Time(sec)

7 Conclusion

The algorithm's run time for Vector-vector, Matrix-vector, and Matrix-matrix multilocation is shown to be efficient in a serial implementation. The results demonstrate that the algorithm for matrix-matrix multiplication has a dramatic increase in run time as the density of the matrix increases, indicating that parallel implementation is best suited for it.