DAPA ASSIGNMENT REPORT

Matrix Multiplication using Cube Interconnection Model and CRCW Model

SUBMITTED BY: Vidhya Lakshmi B (106117107), Rebecca Suraksha (106117075) & Hemakshi Janyani (106117031)

DAPA ASSIGNMENT REPORT

Problem Definition:

1. To develop code for a parallel algorithm using a cube interconnection model to multiply two matrices and comparing its performance with a sequential matrix multiplication algorithm.
2. To develop code for a parallel algorithm for matrix multiplication on a CRCW SIMD model and comparing its performance with a sequential matrix multiplication algorithm.

Assumptions:

1. Assumptions for cube interconnection model: The processors in the cube are simulated using different threads of control. This simulates an environment with a cube of processors with their own local memory and own tasks to perform, parallel w.r.t each other.
2. Assumptions for CRCW Model: The processors in the CRCW are simulated with threads sharing a common memory. The threads process data in parallel which simulates processors working in parallel.

Algorithms:

1. Sequential Algorithm:

```
procedure MATRIX MULTIPLICATION (A, B, C)
        for i = 1 to m do
                for j= 1 to k do
                        (1) c_ij <- 0
                        (2) for s = 1 to n do
                                c_ij <- c_ij + (a_is x b_sj)
                        end for
                end for
        end for.
```

2. Cube Algorithm:

```
procedure CUBE MATRIX MULTIPLICATION (A, B, C)
        Step 1: for m = 3q - 1 down to 2q do
                for all r in {N, r_m = 0} do in parallel
                        (1.1) A_r(m) <-  A_r
                        (1.2) B_r(m) <- B_r
```

Submitted by: Vidhya Lakshmi B (106117107), Rebecca Suraksha (106117075) & Hemakshi Janyani (106117031)

end for
end for.
Step 2: for m = q - 1 down to 0 do
for all r in {N, $r_m = r_{2q+m}$} do in parallel
$A_{r(m)}$ <- A_r
end for
end for.
Step 3: for m = 2q - 1 down to q do
for all r in {N, $r_m = r_{q+m}$} do in parallel
$B_{r(m)}$ <- B_r
end for
end for.
Step 4: for r = 1 to N do in parallel
C_r <- A_r * B_r
end for.
Step 5: for m = 2q to 3q - 1 do
for r = 1 to N do in parallel
C_r <- C_r + $C_{r(m)}$
end for
end for.

3. CRCW Algorithm:

procedure CRCW MATRIX MULTIPLICATION (A, B, C)
for i = 1 to m do in parallel
for j = 1 to k do in parallel
for s = 1 to n do in parallel
(1) c_{ij} <- 0
(2) c_{ij} <- a_{is} x b_{Sj}
end for
end for
end for.

Code:

1. Sequential Code:

```
#include <iostream>
using namespace std;
#define N 4
```

```
void multiply(int mat1[][N],
              int mat2[][N],
              int res[][N])
{
    int i, j, k;
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < N; j++)
        {
            res[i][j] = 0;
            for (k = 0; k < N; k++)
                res[i][j] += mat1[i][k] * mat2[k][j];
        }
    }
}

int main()
{
    int i, j;
    int res[N][N]; // To store result
    int mat1[N][N] = {{1, 1, 1, 1},
                      {2, 2, 2, 2},
                      {3, 3, 3, 3},
                      {4, 4, 4, 4}};

    int mat2[N][N] = {{1, 1, 1, 1},
                      {2, 2, 2, 2},
                      {3, 3, 3, 3},
                      {4, 4, 4, 4}};

    multiply(mat1, mat2, res);

    cout << "Result matrix is \n";
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < N; j++)
        cout << res[i][j] << " ";
        cout << "\n";
    }

    return 0;
}
```

2. Cube Model Code:

```
#include<iostream>
#include<pthread.h>
using namespace std;

int n, A[100][100][100], B[100][100][100], C[100][100][100], part=0, a[100][100], b[100][100];

void* multiply(void* arg)
{
	int i=part/(n*n), j=(part%(n*n))/n, k=part%n;
	C[i][j][k]=A[i][j][k]*B[i][j][k];
	part++;
}

int main()
{
	cout<<"\nEnter n : ";
	cin>>n;

	cout<<"\nEnter elements of first matrix \n";
	for(int i=0;i<n;i++)
		for(int j=0;j<n;j++)
			cin>>a[i][j];

	cout<<"\nEnter elements of second matrix \n";
	for(int i=0;i<n;i++)
		for(int j=0;j<n;j++)
			cin>>b[i][j];

	for(int i=0;i<n;i++)
		for(int j=0;j<n;j++)
			for(int k=0;k<n;k++)
				A[j][i][k]=a[i][j];

	for(int i=0;i<n;i++)
		for(int j=0;j<n;j++)
			for(int k=0;k<n;k++)
				B[i][k][j]=b[i][j];

	pthread_t threads[64];
```

```
	for(int i=0;i<64;i++)
		pthread_create(&threads[i],NULL,multiply,(void*)NULL);


	for(int i=1;i<n;i++)
		for(int j=0;j<n;j++)
			for(int k=0;k<n;k++)
				C[0][j][k]+=C[i][j][k];

	cout<<"\n \n \nElements of result matrix ";
	for(int j=0;j<n;j++)
	{
		cout<<"\n";
		for(int k=0;k<n;k++)
			cout<<C[0][j][k]<<" ";
	}
	cout<<"\n";

	return 0;
}
```

3. CRCW Model Code:

```
#include<iostream>
#include<pthread.h>
using namespace std;

int m,n,k,C[100][100]={0},part=0,a[100][100],b[100][100];

void* multiply(void* arg)
{
	int i=part/k, j=part%k;
	for(int temp=0;temp<n;temp++)
		C[i][j]+=a[i][temp]*b[temp][j];
	part++;
}

int main()
{

	cout<<"\nEnter m, n, k values : ";
	cin>>m;
	cin>>n;
	cin>>k;
```

```
        cout<<"\nEnter elements of first matrix \n";
        for(int i=0;i<m;i++)
                for(int j=0;j<n;j++)
                        cin>>a[i][j];

        cout<<"\nEnter elements of second matrix \n";
        for(int i=0;i<n;i++)
        for(int j=0;j<k;j++)
                cin>>b[i][j];


        int t=m*k;
        pthread_t threads[t];

        for(int i=0;i<t;i++)
                pthread_create(&threads[i],NULL,multiply,(void*)NULL);


        cout<<"\n \n \nElements of result matrix ";
        for(int i=0;i<m;i++)
        {
                cout<<"\n";
                for(int j=0;j<k;j++)
                        cout<<C[i][j]<<" ";
        }
        cout<<"\n";

        return 0;
}
```

Analysis:

1. Sequential Algorithm:
   Assuming that m < n and k < n, it is clear that procedure MATRIX MULTIPLICATION runs in $O(n^3)$ time. Since the innermost loop runs $O(n^3)$ times and multiplication is assumed to take constant time.

2. Cube Model:
   Steps 1, 2, 3, and 5 consist of q constant time iterations, while step 4 takes constant time. Thus, procedure CUBE MATRIX MULTIPLICATION runs in O(q) time, that is, t(n) = O(logn). We now show that this running time is the fastest achievable by any parallel algorithm for multiplying two n x n matrices on the cube.

First note that each c_{ij} is the sum of n elements. It takes Omega(log n) steps to compute this sum on any interconnection network with n (or more) processors. To see this, let s be the smallest number of steps required by a network to compute the sum of n numbers. During the final step, at most one processor is needed to perform the last addition and produce the result. During step s - 1 at most two processors are needed, during step s - 2 at most four processors, and so on. Thus, after s steps, the maximum number of useful additions that can be performed is

$$\sum_{i=0}^{s-1} 2^i = 2^s - 1.$$

Given that exactly n - 1 additions are needed to compute the sum of n numbers, we have n - 1 <= 2^n - 1, that is, s 3 log n.
Since p(n) = n^3, procedure CUBE MATRIX MULTIPLICATION has a cost of
c(n) = O(n^3 logn), which is higher than the running time of sequential procedure MATRIX MULTIPLICATION. Thus, although matrix multiplication on the cube is faster than on the mesh, its cost is higher due to the large number of processors it uses.

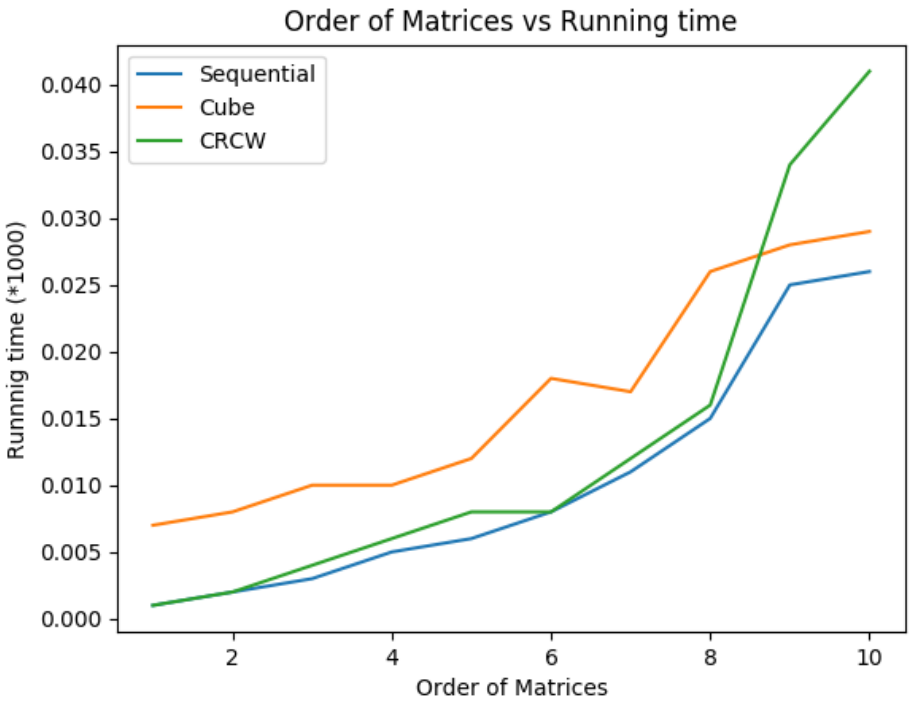
3. CRCW Model:
   It is clear that procedure CRCW MATRIX MULTIPLICATION
   runs in constant time.
   Since p(n) = n^3
   c(n) = p(n) * t(n)
   = n^3 * O(1)
   = O(n^3),
   which matches the running time of sequential procedure MATRIX MULTIPLICATION.

Results & Comparison:

Table:

Input Size (n):	Execution time on Sequential Code:	Execution time on Cube Model:	Execution time on CRCW Model:
1	0.001000	0.007000	0.001000
2	0.002000	0.008000	0.002000
3	0.003000	0.010000	0.004000
4	0.005000	0.010000	0.006000
5	0.006000	0.012000	0.008000
6	0.008000	0.018000	0.008000
7	0.011000	0.017000	0.012000
8	0.015000	0.026000	0.016000
9	0.025000	0.028000	0.034000
10	0.026000	0.029000	0.041000

Graph:
X- Axis: Order Of Matrices
Y- Axis: Execution Time



Conclusion:

Speedup and efficiency are calculated for both the models with respect to the sequential model:

1. Cube interconnection model is not optimal for multiplication as:
   Speedup = $(n^3/\log n)$
   Efficiency = $(n^3/ (\log n * n^3)) = 1/\log n$

2. CRCW Model is optimal as:
   Speedup = $(n^3/1) = n^3$
   Efficiency = $(n^3/ n^3) = 1$

Submitted by: Vidhya Lakshmi B (106117107), Rebecca Suraksha (106117075) & Hemakshi Janyani (106117031)