High Performance Computing
CS5013
Lab 03

R Abinav
ME23B1004

January 28, 2026

1 Implementation

1.1 Matrix Addition

```c
#include<stdio.h>
#include<stdlib.h>
#include<omp.h>
#include<time.h>

#define rows 12000
#define cols 12000

int main(){
    printf("====== Start of programme ======\n");
    srand(time(NULL));

    int threads[] = {1, 2, 4, 6, 8, 10, 12, 16, 20, 24, 32, 64};
    int size = sizeof(threads)/sizeof(threads[0]);

    //the two matrices to be added
    double ** a = malloc(rows * sizeof(double *));
    for(int i=0; i<rows; i++) a[i] = malloc(cols * sizeof(double));

    double ** b = malloc(rows * sizeof(double *));
    for(int i=0; i<rows; i++) b[i] = malloc(cols * sizeof(double));
    double t_1 = -1;

    //initialise the matrices a and b
    for(int i=0; i<rows; i++){
        for(int j=0; j<cols; j++){
            a[i][j] = (double)rand() * 10000;
            b[i][j] = (double)rand() * 10110;
        }
    }

    for(int i=0; i<size; i++){
        //the resultant matrix
        double ** c = malloc(rows * sizeof(double *));
        for(int a=0; a<rows; a++) c[a] = malloc(cols * sizeof(double));

        omp_set_num_threads(threads[i]);
        double s_time = omp_get_wtime();

        #pragma omp parallel for
        for(int j=0; j<rows; j++){
            for(int k=0; k<cols; k++){
                c[j][k] = a[j][k] + b[j][k];
            }
        }

        double e_time = omp_get_wtime();

        //free matrix c
        for(int a=0; a<rows; a++) free(c[a]);
        free(c);

        double exec_time = e_time - s_time;
        printf("Execution time with %d thread(s): %lf\n",
               threads[i], exec_time);

        //speedup
        double speedup = -1.00;
        if(i == 0){
            t_1 = exec_time;
            speedup = (double)t_1/exec_time;
        }else{
            speedup = (double)t_1/exec_time;
        }

        printf("The speedup for %d thread(s) is: %lf\n",
               threads[i], speedup);

        //Parallelisation factor
        double f = (threads[i] * (speedup - 1)) /
                   ((threads[i] - 1) * speedup);
        printf("The parallelisation fraction for %d thread(s) is: %lf\n",
               threads[i], f);
        printf("\n\n");
    }

    for(int i=0; i<rows; i++) free(a[i]);
    free(a);
    for(int i=0; i<rows; i++) free(b[i]);
    free(b);

    printf("====== End of programme ======\n");
    return 0;
}
```

Listing 1: OpenMP Parallel Matrix Addition

1.2 Matrix Multiplication

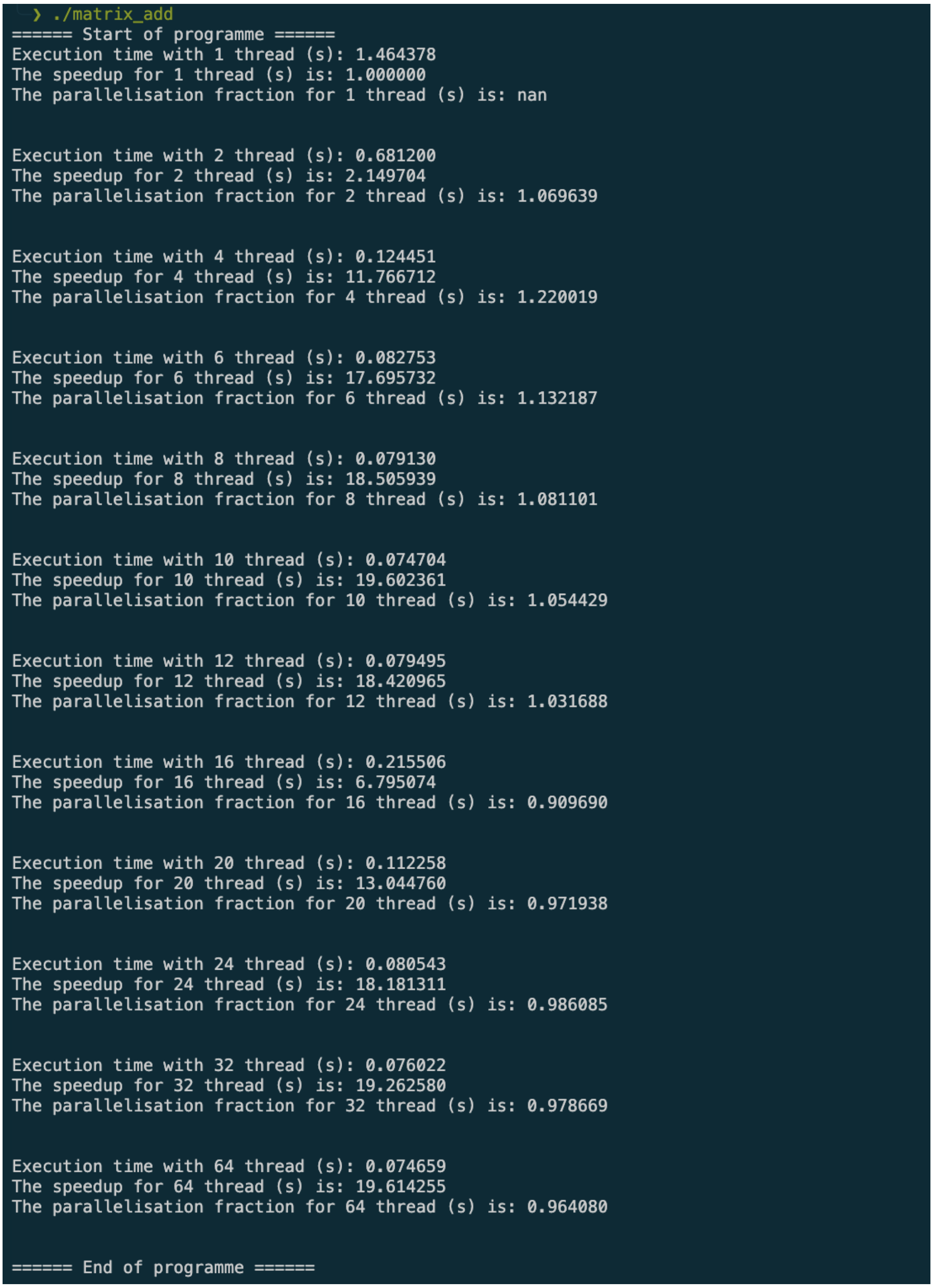
```c
#include<stdio.h>
#include<stdlib.h>
#include<omp.h>
#include<time.h>

#define rows 3500
#define cols 3500

int main(){
    printf("====== Start of programme ======\n");
    srand(time(NULL));

    int threads[] = {1, 2, 4, 6, 8, 10, 12, 16, 20, 24, 32, 64};
    int size = sizeof(threads)/sizeof(threads[0]);

    //the two matrices to be added
    double ** a = malloc(rows * sizeof(double *));
    for(int i=0; i<rows; i++) a[i] = malloc(cols * sizeof(double));

    double ** b = malloc(rows * sizeof(double *));
    for(int i=0; i<rows; i++) b[i] = malloc(cols * sizeof(double));
    double t_1 = -1;

    //initialise the matrices a and b
    for(int i=0; i<rows; i++){
        for(int j=0; j<cols; j++){
            a[i][j] = (double)rand() * 10000;
            b[i][j] = (double)rand() * 10110;
        }
    }

    for(int i=0; i<size; i++){
        //the resultant matrix
        double ** c = malloc(rows * sizeof(double *));
        for(int a=0; a<rows; a++) c[a] = malloc(cols * sizeof(double));

        omp_set_num_threads(threads[i]);
        double s_time = omp_get_wtime();

        #pragma omp parallel for
        for(int j=0; j<rows; j++){
            for(int k=0; k<cols; k++){
                c[j][k] = 0;
                for(int w=0; w<rows; w++){
                    c[j][k] += a[j][w] * b[w][k];
                }
            }
        }

        double e_time = omp_get_wtime();

        //free matrix c
        for(int a=0; a<rows; a++) free(c[a]);
        free(c);

        double exec_time = e_time - s_time;
        printf("Execution time with %d thread (s): %lf\n",
               threads[i], exec_time);

        //speedup
        double speedup = -1.00;
        if(i == 0){
            t_1 = exec_time;
            speedup = (double)t_1/exec_time;
        }else{
            speedup = (double)t_1/exec_time;
        }

        printf("The speedup for %d thread (s) is: %lf\n",
               threads[i], speedup);

        //Parallelisation factor
        double f = (threads[i] * (speedup - 1)) /
                   ((threads[i] - 1) * speedup);
        printf("The parallelisation fraction for %d thread (s) is
           : %lf\n",
               threads[i], f);
        printf("\n\n");
    }

    for(int i=0; i<rows; i++) free(a[i]);
    free(a);
    for(int i=0; i<rows; i++) free(b[i]);
    free(b);

    printf("====== End of programme ======\n");
    return 0;
```

```
}
```

Listing 2: OpenMP Parallel Matrix Multiplication

2 Results and Analysis

2.1 Matrix Addition Output



```
 ❯ ./matrix_add
====== Start of programme ======
Execution time with 1 thread (s): 1.464378
The speedup for 1 thread (s) is: 1.000000
The parallelisation fraction for 1 thread (s) is: nan


Execution time with 2 thread (s): 0.681200
The speedup for 2 thread (s) is: 2.149704
The parallelisation fraction for 2 thread (s) is: 1.069639


Execution time with 4 thread (s): 0.124451
The speedup for 4 thread (s) is: 11.766712
The parallelisation fraction for 4 thread (s) is: 1.220019


Execution time with 6 thread (s): 0.082753
The speedup for 6 thread (s) is: 17.695732
The parallelisation fraction for 6 thread (s) is: 1.132187


Execution time with 8 thread (s): 0.079130
The speedup for 8 thread (s) is: 18.505939
The parallelisation fraction for 8 thread (s) is: 1.081101


Execution time with 10 thread (s): 0.074704
The speedup for 10 thread (s) is: 19.602361
The parallelisation fraction for 10 thread (s) is: 1.054429


Execution time with 12 thread (s): 0.079495
The speedup for 12 thread (s) is: 18.420965
The parallelisation fraction for 12 thread (s) is: 1.031688


Execution time with 16 thread (s): 0.215506
The speedup for 16 thread (s) is: 6.795074
The parallelisation fraction for 16 thread (s) is: 0.909690


Execution time with 20 thread (s): 0.112258
The speedup for 20 thread (s) is: 13.044760
The parallelisation fraction for 20 thread (s) is: 0.971938


Execution time with 24 thread (s): 0.080543
The speedup for 24 thread (s) is: 18.181311
The parallelisation fraction for 24 thread (s) is: 0.986085


Execution time with 32 thread (s): 0.076022
The speedup for 32 thread (s) is: 19.262580
The parallelisation fraction for 32 thread (s) is: 0.978669


Execution time with 64 thread (s): 0.074659
The speedup for 64 thread (s) is: 19.614255
The parallelisation fraction for 64 thread (s) is: 0.964080


====== End of programme ======
```

Figure 1: Matrix Addition Program Output

2.2 Performance Data - Matrix Addition

Table 1: Execution Time, Speedup and Parallelisation Fraction - Matrix Addition

Threads	Execution Time (s)	Speedup	Parallelisation Fraction
1	1.464378	1.000000	–
2	0.681200	2.149704	1.069639
4	0.124451	11.766712	1.220019
6	0.082753	17.695732	1.132187
8	0.079130	18.505939	1.081101
10	0.074704	19.602361	1.054429
12	0.079495	18.420965	1.031688
16	0.215506	6.795074	0.909690
20	0.112258	13.044760	0.971938
24	0.080543	18.181311	0.986085
32	0.076022	19.262580	0.978669
64	0.074659	19.614255	0.964080

2.3 Visualizations - Matrix Addition

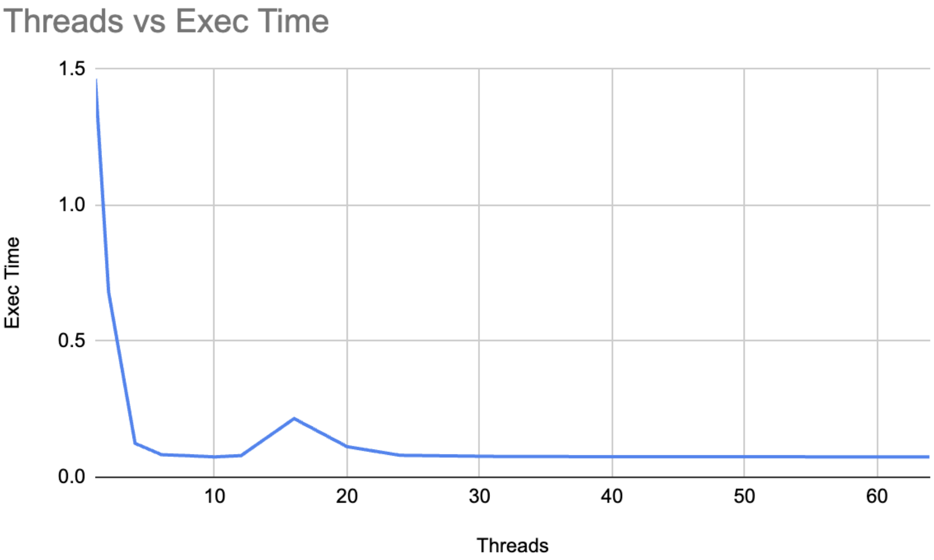


Figure 2: Threads vs Execution Time - Matrix Addition

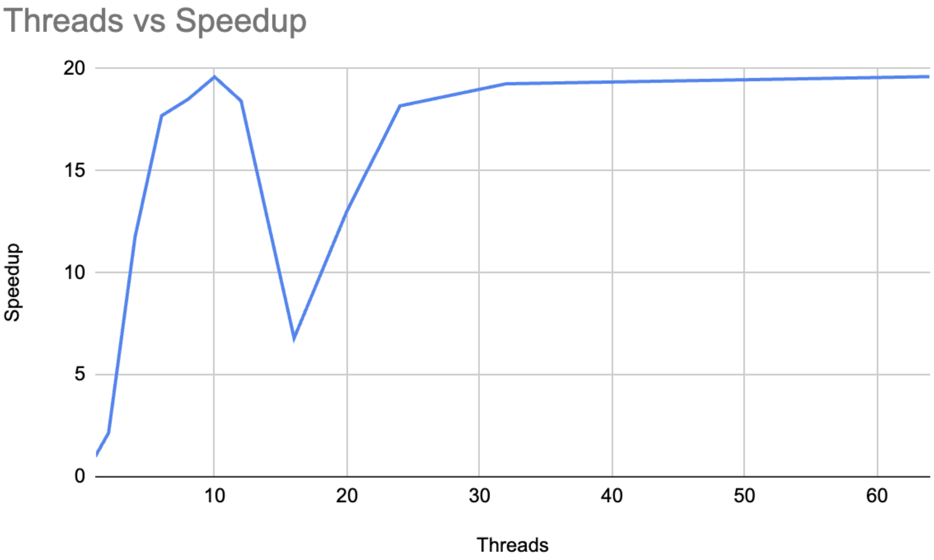


Figure 3: Threads vs Speedup - Matrix Addition

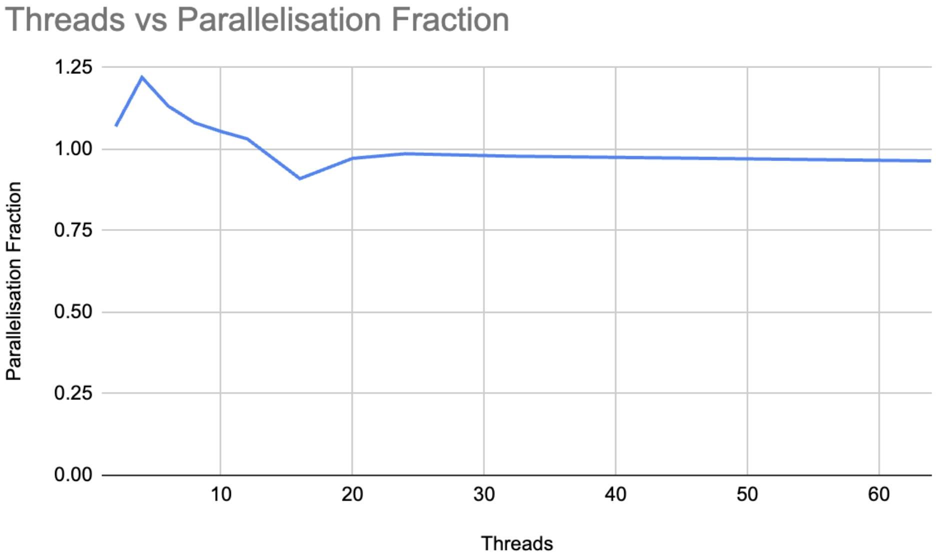
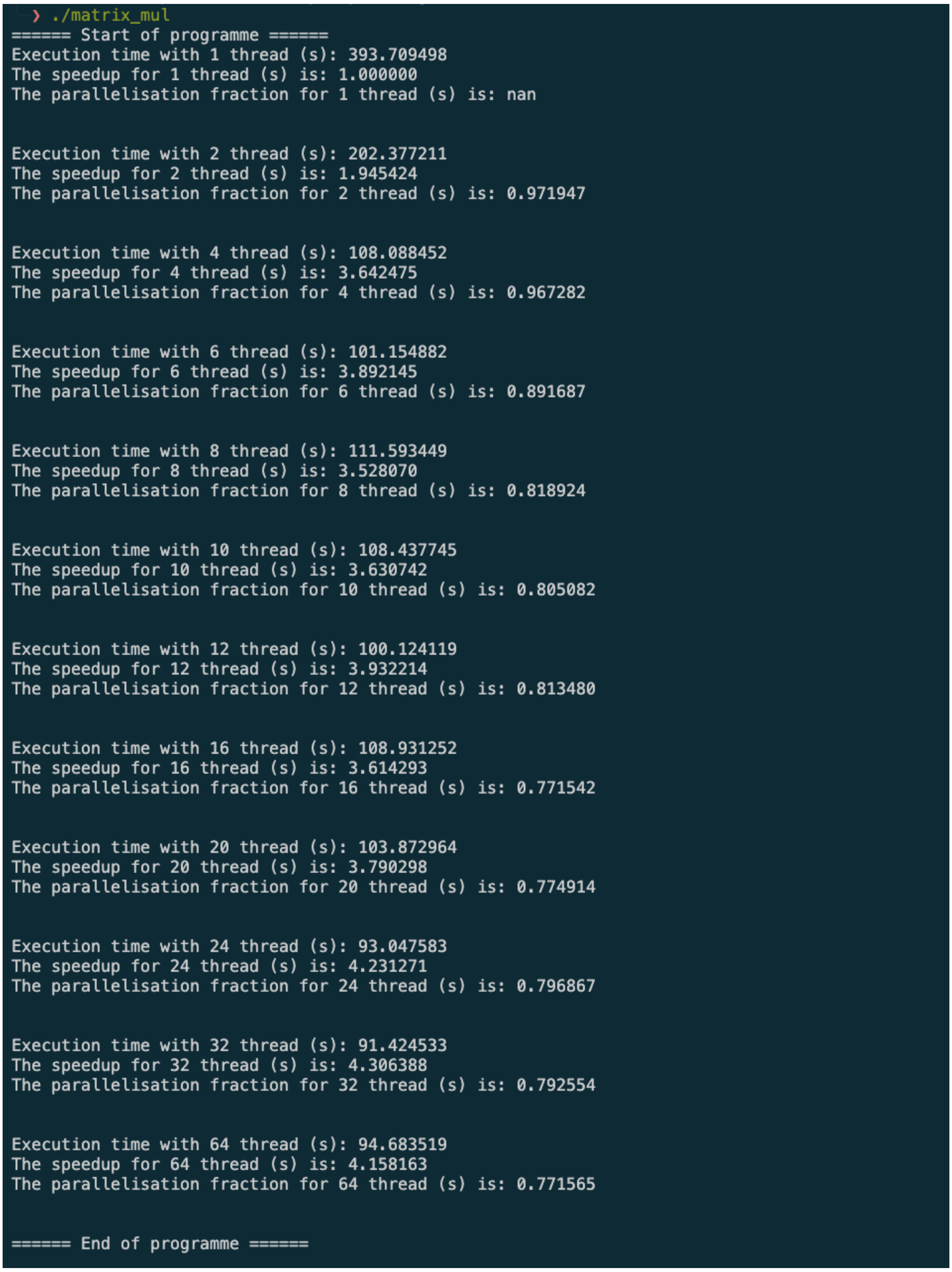


Figure 4: Threads vs Parallelisation Fraction - Matrix Addition

2.4 Matrix Multiplication Output



```
╰─❯ ./matrix_mul
====== Start of programme ======
Execution time with 1 thread (s): 393.709498
The speedup for 1 thread (s) is: 1.000000
The parallelisation fraction for 1 thread (s) is: nan


Execution time with 2 thread (s): 202.377211
The speedup for 2 thread (s) is: 1.945424
The parallelisation fraction for 2 thread (s) is: 0.971947


Execution time with 4 thread (s): 108.088452
The speedup for 4 thread (s) is: 3.642475
The parallelisation fraction for 4 thread (s) is: 0.967282


Execution time with 6 thread (s): 101.154882
The speedup for 6 thread (s) is: 3.892145
The parallelisation fraction for 6 thread (s) is: 0.891687


Execution time with 8 thread (s): 111.593449
The speedup for 8 thread (s) is: 3.528070
The parallelisation fraction for 8 thread (s) is: 0.818924


Execution time with 10 thread (s): 108.437745
The speedup for 10 thread (s) is: 3.630742
The parallelisation fraction for 10 thread (s) is: 0.805082


Execution time with 12 thread (s): 100.124119
The speedup for 12 thread (s) is: 3.932214
The parallelisation fraction for 12 thread (s) is: 0.813480


Execution time with 16 thread (s): 108.931252
The speedup for 16 thread (s) is: 3.614293
The parallelisation fraction for 16 thread (s) is: 0.771542


Execution time with 20 thread (s): 103.872964
The speedup for 20 thread (s) is: 3.790298
The parallelisation fraction for 20 thread (s) is: 0.774914


Execution time with 24 thread (s): 93.047583
The speedup for 24 thread (s) is: 4.231271
The parallelisation fraction for 24 thread (s) is: 0.796867


Execution time with 32 thread (s): 91.424533
The speedup for 32 thread (s) is: 4.306388
The parallelisation fraction for 32 thread (s) is: 0.792554


Execution time with 64 thread (s): 94.683519
The speedup for 64 thread (s) is: 4.158163
The parallelisation fraction for 64 thread (s) is: 0.771565


====== End of programme ======
```

Figure 5: Matrix Multiplication Program Output

2.5 Performance Data - Matrix Multiplication

Table 2: Execution Time, Speedup and Parallelisation Fraction - Matrix Multiplication

Threads	Execution Time (s)	Speedup	Parallelisation Fraction
1	393.709498	1.000000	–
2	202.377211	1.945424	0.971947
4	108.088452	3.642475	0.967282
6	101.154882	3.892145	0.891687
8	111.593449	3.528070	0.818924
10	108.437745	3.630742	0.805082
12	100.124119	3.932214	0.813480
16	108.931252	3.614293	0.771542
20	103.872964	3.790298	0.774914
24	93.047583	4.231271	0.796867
32	91.424533	4.306388	0.792554
64	94.683519	4.158163	0.771565

2.6 Visualizations - Matrix Multiplication

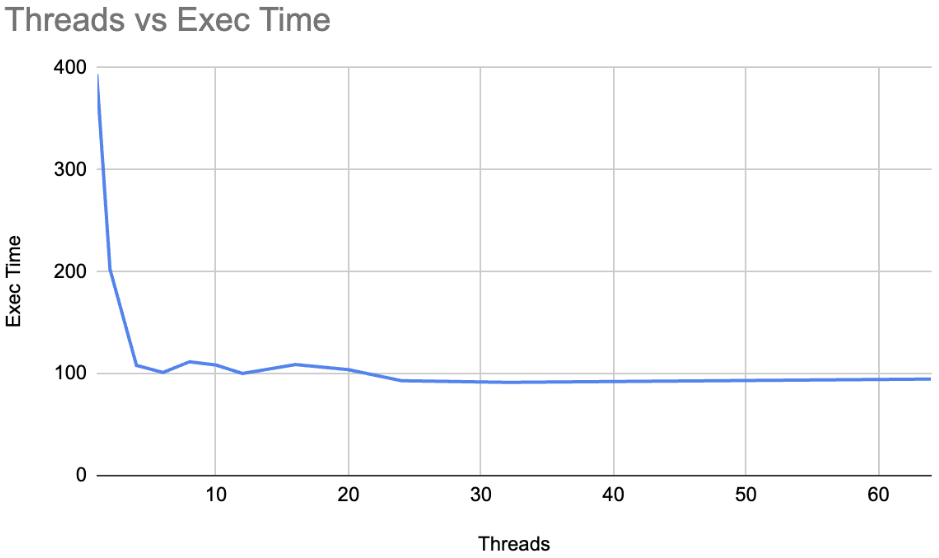


Figure 6: Threads vs Execution Time - Matrix Multiplication

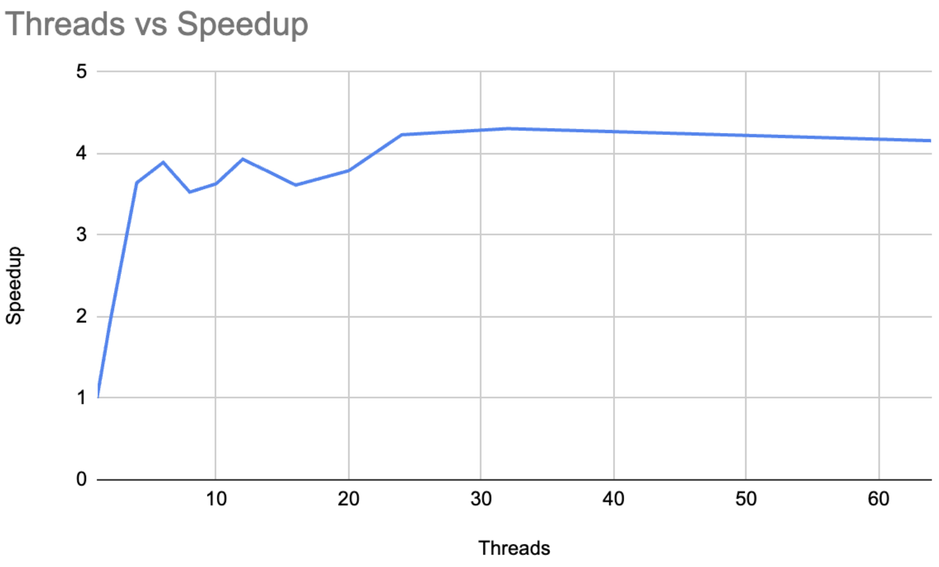


Figure 7: Threads vs Speedup - Matrix Multiplication

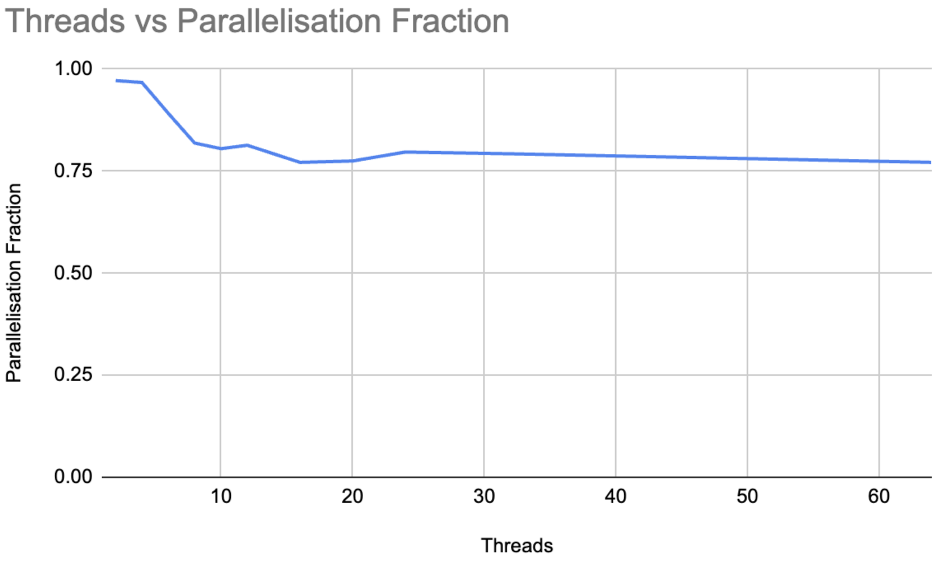


Figure 8: Threads vs Parallelisation Fraction - Matrix Multiplication