HPC Experiment 7 Report

Block Based Matrix Multiplication

Name: Reuben Skariah Mathew
Roll No: CED18I042
Programming Environment: OpenMP
Problem: Block Based Matrix Multiplication
Date: 2nd September 2021

Hardware Configuration:
Processor: Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz
Sockets: 1
Cores per Socket: 4
Threads per Core: 2
L1 Cache: 32 kB
L2 Cache: 256 kB
L3 Cache: 6 MB
RAM: 8 GB

Serial Code:

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

#define MIN(X, Y) (((X) < (Y)) ? (X) : (Y))

int block[] = {5, 10, 50, 60, 70, 80, 90, 100};
int main()
{
    double st, et, tt[8];

    int L = 500, M = 500, N = 500;

    double matrix_x[500][500], matrix_y[500][500], matrix_z[500][500];

     for (int i = 0; i < 500; i++)
     {
         for (int j = 0; j < 500; j++)
         {
            matrix_x[i][j] = (i * 5.123);
            matrix_y[i][j] = (j * 3.123);
            matrix_z[i][j] = 0;
         }
     }
    for(int bl = 0; bl < 4; bl++)
    {
        st = omp_get_wtime();

            #pragma omp for
```

```
            for(int jj = 0; jj < N; jj = jj + block[bl])
            {
                for(int kk = 0; kk < N; kk = kk + block[bl])
                {
                    for(int i = 0; i < N; i++)
                    {
                        for(int j = jj; j < MIN(jj + block[bl], N); j++)
                        {
                            double r = 0;
                            for(int k = kk; k < MIN(kk + block[bl], N); k++)
                            {
                                r += matrix_x[i][k] * matrix_y[k][j];
                            }
                            matrix_z[i][j] = matrix_z[i][j] + r;
                        }
                    }
                }
            }

        et = omp_get_wtime();
        tt[bl] = et - st;
    }
    for (int i = 0; i < 8; i++)
    {
        printf("\n\nBlock Size: %d", block[i]);
        printf("\nrun time = %f\n", tt[i]);
    }
    return 0;
}
```

Parallel Code:

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

#define MIN(X, Y) (((X) < (Y)) ? (X) : (Y))

int thread[] = {1, 2, 4, 6, 8, 10, 12, 16, 20, 32, 64, 128, 150};

int block[] = {5, 10, 50, 60, 70, 80, 90, 100};
int main()
{
    double st, et, tt[8][13];

    int omp_rank, omp_threads, L = 500, M = 500, N = 500;

    double matrix_x[500][500], matrix_y[500][500], matrix_z[500][500];

    #pragma omp for collapse(2)
     for (int i = 0; i < 500; i++)
     {
         for (int j = 0; j < 500; j++)
         {
            matrix_x[i][j] = (i * 5.123);
            matrix_y[i][j] = (j * 3.123);
            matrix_z[i][j] = 0;
         }
     }
    for(int bl = 0; bl < 4; bl++)
    {
        for (int k = 0; k < 13; k++)
        {
            omp_set_num_threads(thread[k]);
            st = omp_get_wtime();
            #pragma omp parallel private(omp_rank)
            {
                omp_rank = omp_get_thread_num();
                omp_threads = omp_get_num_threads();

                #pragma omp for
                for(int jj = 0; jj < N; jj = jj + block[bl])
                {
                    for(int kk = 0; kk < N; kk = kk + block[bl])
                    {
                        for(int i = 0; i < N; i++)
                        {
                            for(int j = jj; j < MIN(jj + block[bl], N); j++)
                            {
                                double r = 0;
                                for(int k = kk; k < MIN(kk + block[bl], N); k++)
                                {
                                    r += matrix_x[i][k] * matrix_y[k][j];
                                }
```

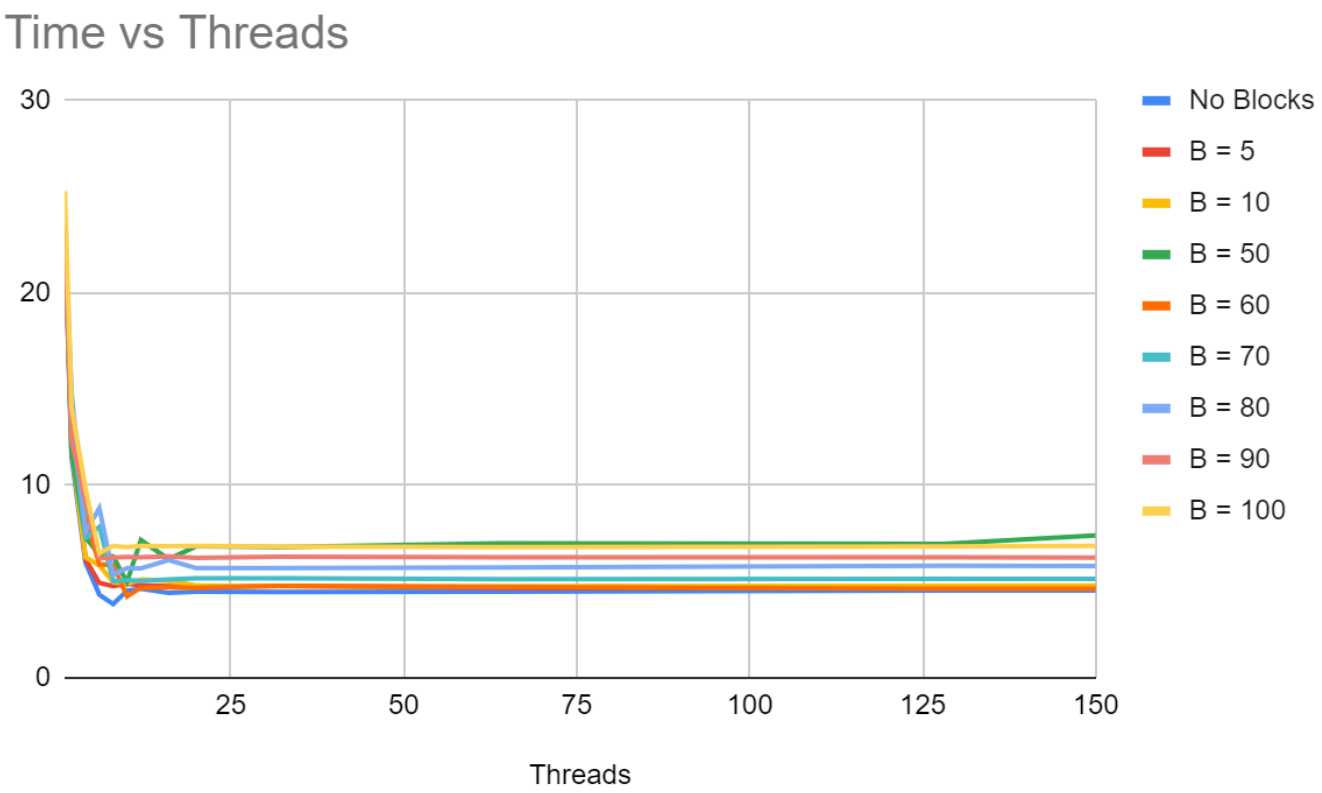
```
                                matrix_z[i][j] = matrix_z[i][j] + r;
                            }
                        }
                    }
                }
            }
            et = omp_get_wtime();
            tt[bl][k] = et - st;
        }
    }
    for (int i = 0; i < 8; i++)
    {
        printf("\n\nBlock Size: %d", block[i]);
        for(int j = 0; j < 13; j++)
            printf("\nthreads =%d time = %f\n", thread[j], tt[i][j]);
    }
  return 0;
}
```

Analysis

- **Runtimes:**

Threads	No Blocks	B = 5	B = 10	B = 50	B = 60	B = 70	B = 80	B = 90	B = 100
1	22.834381	23.486433	23.224251	23.009945	22.953027	23.151415	23.005722	22.946544	25.309478
2	11.48587	12.013435	11.856178	11.746762	13.906575	13.101539	14.805051	12.376577	14.050808
4	6.000427	6.157926	6.262981	7.319166	8.407262	7.272827	7.587125	9.161485	9.816426
6	4.318604	4.935832	5.844545	6.397539	5.897808	7.838773	8.810107	**6.22314**	**6.463053**
8	**3.824371**	4.772059	4.971684	6.303351	5.854035	**5.014907**	**5.354247**	6.249338	6.853781
10	4.535858	4.881417	4.868007	**4.973113**	**4.229139**	5.08979	5.702993	6.282035	6.790966
12	4.62796	4.811815	5.120404	7.167268	4.68071	5.04532	5.68634	6.26236	6.851798
16	4.424744	4.766318	5.037133	6.164324	4.699177	5.11005	6.122624	6.316504	6.8426
20	4.478333	4.736443	4.791715	6.834658	4.680541	5.169086	5.701152	6.228981	6.855366
32	4.466217	4.787231	4.7674	6.79396	4.784448	5.173267	5.698317	6.290286	6.823517
64	4.477203	4.676044	**4.766442**	6.98979	4.718529	5.125316	5.734176	6.260243	6.795096

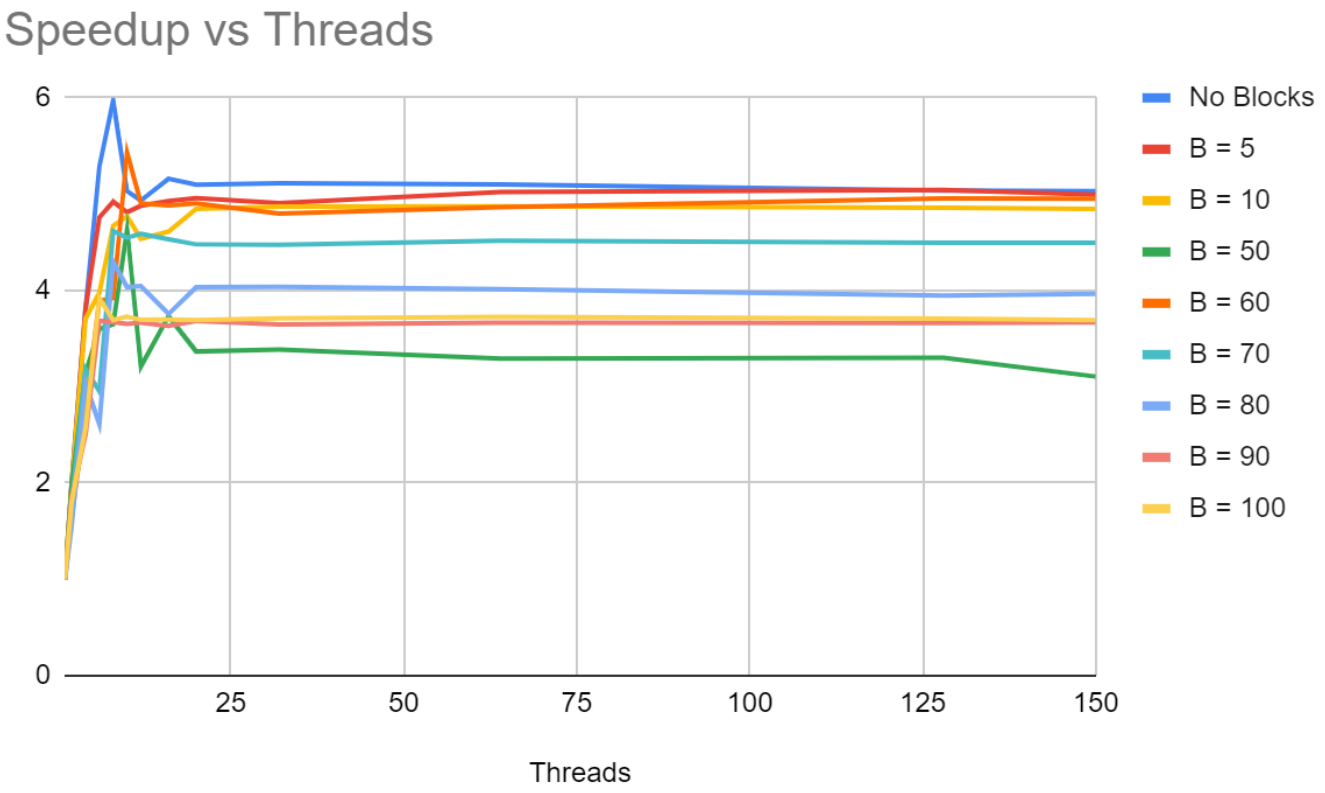
128	4.532837	**4.656539**	4.780197	6.967308	4.632021	5.150665	5.826881	6.265207	6.825074
150	4.538055	4.701207	4.790802	7.40836	4.634609	5.150174	5.803925	6.254129	6.85385



- **Speedup:**

Threads	No Blocks	B = 5	B = 10	B = 50	B = 60	B = 70	B = 80	B = 90	B = 100
1	1	1	1	1	1	1	1	1	1
2	1.988041045	1.955013949	1.958831168	1.958832996	1.650516177	1.767075 99	1.553910351	1.854029915	1.801282745
4	3.805459345	3.814016765	3.708178422	3.143793296	2.730142941	3.183275912	3.032205 48	2.504675 17	2.578278286
6	5.287444971	4.758353404	3.973662792	3.596686945	3.891789458	2.953448837	2.611287468	**3.687293553**	**3.916025135**
8	**5.970754668**	4.921656 04	4.671304733	3.650430541	3.920889 95	**4.616519309**	**4.296724077**	3.671835961	3.692776002
10	5.034192208	4.811396568	4.770792441	**4.626869528**	**5.427352234**	4.548599255	4.033973389	3.652724635	3.726933399
12	4.934005696	4.880992515	4.535628634	3.210420623	4.903749004	4.588691104	4.045787273	3.664200717	3.693844 74
16	5.160610648	4.927584144	4.610609051	3.732760478	4.884478069	4.530565259	3.757493846	3.632791 81	3.698810101

20	5.09885732	4.958664762	4.84675132	3.366656386	4.903926063	4.478821788	4.035276028	3.683835928	3.691922211
32	5.112689554	4.906058011	4.871471032	3.386823738	4.79742428	4.475202034	4.03728364	3.647933337	3.709154385
64	5.10014422	5.022714286	**4.872450142**	3.291936525	4.864445466	4.517070752	4.012036254	3.665439824	3.724668202
128	5.037547346	**5.043753096**	4.858429684	3.302558894	4.955294244	4.494839987	3.948205223	3.662535651	3.708308218
150	5.031755014	4.995830432	4.847674982	3.105943151	4.952527171	4.495268509	3.963821379	3.669023137	3.692738826



Inference:

- Maximum speedup was observed at thread count equal to 8 in the non-blocking method.
- In the case for block based matrix multiplication: for low number of blocks (5 - 60) the maximum speedup was observed at threads counts higher than 8. For B = 80 and 90 maximum speedup was observed at thread count = 8. When B is increased further the maximum speedup was found to be at thread count = 6 (lower than 8).
- When comparing the matrix addition with and without block, lower runtime and higher speedup was observed for the method without block. Although when B = 60 the speedup was comparable at thread count = 8.