Cairo University
Faculty of Computers and AI

Parallel Processing Research Matrix Multiplication using Fine-Grained Parallelism 2020

PID23875668

Abdallah Mohamed Naguib 20170160

abdonaguib99@gmail.com

Mohamed Ahmed Saad 20170212

mohamedsaad17841@gmail.com

Hussien Tarek Ismail 20170094

sehes333@stud.fci-cu.edu.eg

Note :
you can find better formatted tables in this drive folder :

https://drive.google.com/drive/folders/1tm0vzZ844mEO3J-qEXtr2_uWPB1noDQH?usp=sharing

Serial Program :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

int *generate1Darray(int n){
    return (int *) malloc(n * sizeof(int));
}

int **generate2Darray(int r, int c) {
    int **arr = (int **) malloc(r * sizeof(int *));
    for (int i = 0; i < r; i++)
        arr[i] = generate1Darray(c);
    return arr;
}
int main(int argc, char *argv[]) {
    freopen("/media/abdo/New Volume1/My
work/serial.ods","w",stdout);
    printf("matrix sz,time\n");
    for(int i=24 ; i<=1024 ;i+=100) {
        int **mat1 = generate2Darray(i, i);
        int **mat2 = generate2Darray(i, i);
        for (int j = 0; j < i; j++) {
            for (int k = 0; k < i; k++) {
                mat1[j][k] = mat2[j][k] = j * i + k;
            }
        }

        clock_t begin = clock();
        int **res = generate2Darray(i, i);
```

```
        for (int j = 0; j < i; j++) {
            for (int k = 0; k < i; k++) {
                res[j][k] = 0;
                for (int l = 0; l < i; l++) {
                    res[j][k] += mat1[j][l] * mat2[l][k];
                }
            }
        }
        clock_t end = clock();
        double time_spent = (double)(end - begin) /
CLOCKS_PER_SEC;
        printf("%d,%f\n",i,time_spent*1000);
    }

}
```

matrix sz	time
24	0.146
124	15.298
224	77.104
324	209.224
424	508.008
524	963.851
624	1823.422
724	2989.713
824	6284.355
924	7530.048
1024	14090.208

better formatted table can be found in here :

https://drive.google.com/file/d/1kuFAISxGeIR6JEHP_FEOByj9MqbDh7Us/view?usp=sharing

Parallel program using openMP :

Algorithm :

for each process :

 # find its segment of elements that it will calculate , they
 # will be consecutive in the matrix for example from (1,1)
 # (2,3) this means this process take the first row complete

 # and the first three elements from the second
 # then for each element in its segment calculate the result
 # for that index by the known matrix multiplication formula
 # and put the result in a shared 2-D array
 # note that the initial two matrices will be saved In shared
 # memory

```
#include <stdio.h>
#include "omp.h"
#include <stdlib.h>
#include <sys/time.h>
int *generate1Darray(int n){
    return (int *) malloc(n * sizeof(int));
}
int **generate2Darray(int r, int c) {
    int **arr = (int **) malloc(r * sizeof(int *));
    for (int i = 0; i < r; i++)
        arr[i] = generate1Darray(c);
    return arr;
}
int main (){
    freopen("/media/abdo/New Volume1/My
work/openMP.ods","w",stdout);
    printf("Nom of procs/mat sz,");
    FILE *f=fopen("/media/abdo/New Volume1/My
work/openMPSpeedUp.ods","w");
    FILE *f2=fopen("/media/abdo/New Volume1/My
work/openMPEff.ods","w");
    fprintf(f,"Nom of procs/mat sz,");
    fprintf(f2,"Nom of procs/mat sz,");
    int start=24,end=1024,stp=100;
    for(int i=1 ; i<=101 ; i+=10){
        printf("%d%c",i,",\n"[i==101]);
        fprintf(f,"%d%c",i,",\n"[i==101]);
        fprintf(f2,"%d%c",i,",\n"[i==101]);
    }
    for(int i=start ; i<=end  ; i+=stp) {
        int **mat1 = generate2Darray(i, i);
        int **mat2 = generate2Darray(i, i);
```

```
        for (int j = 0; j < i; j++) {
            for (int k = 0; k < i; k++) {
                mat1[j][k] = mat2[j][k] = j * i + k;
            }
        }
        printf("%d,",i);
        fprintf(f,"%d,",i);
        fprintf(f2,"%d,",i);
        int **res = generate2Darray(i,i);
        double init=1;
        for (int pCnt = 1; pCnt <= 101; pCnt += 10) {
            omp_set_num_threads(pCnt);
            struct timeval startTime, endTime;
            int eachProc = i*i / pCnt;
            gettimeofday(&startTime, NULL);
#pragma omp parallel
            {
                int id = omp_get_thread_num();
                int st = id*eachProc;
                int en = st+eachProc-1;
                for(int row = st/i ; row<=en/i ; row++){
                    int colStart = 0 , colEnd=i-1;
                    if(row == st/i){
                        colStart = st%i;
                    }
                    if(row == en/i){
                        colEnd = en%i;
                    }
                    for(int col=colStart ; col<=colEnd ; col++){
                        res[row][col] = 0;
                        for(int k=0 ; k<i ; k++){
                            res[row][col] += mat1[row][k]*mat2[k]
[col];
                        }
                    }
                }
            }
            gettimeofday(&endTime, NULL);
            double delta = ((endTime.tv_sec  - startTime.tv_sec) *
1000000u +
                     endTime.tv_usec - startTime.tv_usec) / 1.e6;
```

```
            printf("%f,",delta*1000);
            if(pCnt==1){
                init=delta;
            }
            double spdUp=init/delta;
            fprintf(f,"%.3f,",init/delta);
            fprintf(f2,"%.3f,",spdUp/pCnt);
        }
        printf("\n");
        fprintf(f,"\n");
        fprintf(f2,"\n");
    }
    fclose (f);
    fclose (f2);
}
```

running time :

mat sz/Nom of procs	1	11	21	31	41	51	61	71	81	91	101
24	0.374	0.895	0.666	0.959	1.031	1.038	3.909	1.637	1.664	1.449	1.619
124	36.483	11.678	8.609	7.107	9.39	6.552	7.526	8.272	6.615	7.634	7.172
224	72.187	39.626	37.209	38.942	36.793	38.402	38.073	37.734	38.749	40.225	38.355
324	234.08	119.837	119.709	120.623	123.19	119.963	126.416	156.787	140.769	126.163	150.432
424	691.463	298.286	320.865	412.971	478.722	465.602	512.317	762.459	791.874	496.575	526.027
524	1958.786	604.056	546.591	549.42	554.851	754.81	982.575	1050.333	1017.372	1021.791	973.654
624	4011.048	1451.4	1655.508	1853.011	1681.356	1601.98	1353.002	1198.794	1560.993	2195.658	1942.388
724	5037.389	1900.547	1731.035	1696.001	1816.044	1979.822	4736.537	3831.492	2823.659	2894.524	3576.301
824	8180.184	2757.352	2658.479	3669.742	3782.02	3811.429	4623.004	5607.842	3724.149	4398.243	3319.619
924	14907.939	6863.259	8863.654	9279.761	6678.208	6624.051	7674.901	7000.598	10240.275	6105.373	6205.49
1024	19259.447	12444.569	7604.423	10193.139	11386.451	10736.349	11269.063	10928.874	11862.979	9982.85	9920.457

better formatted table can be found in here :

https://drive.google.com/file/d/1wdPhAUVT70PoyHdHv7et09IFABl1Lv6j/view?usp=sharing

Speed Up :

mat sz/Nom of procs	1	11	21	31	41	51	61	71	81	91	101
24	1	0.418	0.562	0.39	0.363	0.36	0.096	0.228	0.225	0.258	0.231
124	1	3.124	4.238	5.133	3.885	5.568	4.848	4.41	5.515	4.779	5.087
224	1	1.822	1.94	1.854	1.962	1.88	1.896	1.913	1.863	1.795	1.882
324	1	1.953	1.955	1.941	1.9	1.951	1.852	1.493	1.663	1.855	1.556
424	1	2.318	2.155	1.674	1.444	1.485	1.35	0.907	0.873	1.392	1.315
524	1	3.243	3.584	3.565	3.53	2.595	1.994	1.865	1.925	1.917	2.012
624	1	2.764	2.423	2.165	2.386	2.504	2.965	3.346	2.57	1.827	2.065
724	1	2.65	2.91	2.97	2.774	2.544	1.064	1.315	1.784	1.74	1.409
824	1	2.967	3.077	2.229	2.163	2.146	1.769	1.459	2.197	1.86	2.464
924	1	2.172	1.682	1.607	2.232	2.251	1.942	2.13	1.456	2.442	2.402
1024	1	1.548	2.533	1.889	1.691	1.794	1.709	1.762	1.623	1.929	1.941

better formatted table can be found in here :

https://drive.google.com/file/d/11xBfFAeim6WiXT98qM2xIdtzrGiynsqq/view?usp=sharing

Efficiency:

Nom of procs/mat sz	1	11	21	31	41	51	61	71	81	91	101
24	1	0.038	0.027	0.013	0.009	0.007	0.002	0.003	0.003	0.003	0.002
124	1	0.284	0.202	0.166	0.095	0.109	0.079	0.062	0.068	0.053	0.05
224	1	0.166	0.092	0.06	0.048	0.037	0.031	0.027	0.023	0.02	0.019
324	1	0.178	0.093	0.063	0.046	0.038	0.03	0.021	0.021	0.02	0.015
424	1	0.211	0.103	0.054	0.035	0.029	0.022	0.013	0.011	0.015	0.013
524	1	0.295	0.171	0.115	0.086	0.051	0.033	0.026	0.024	0.021	0.02
624	1	0.251	0.115	0.07	0.058	0.049	0.049	0.047	0.032	0.02	0.02
724	1	0.241	0.139	0.096	0.068	0.05	0.017	0.019	0.022	0.019	0.014
824	1	0.27	0.147	0.072	0.053	0.042	0.029	0.021	0.027	0.02	0.024
924	1	0.197	0.08	0.052	0.054	0.044	0.032	0.03	0.018	0.027	0.024
1024	1	0.141	0.121	0.061	0.041	0.035	0.028	0.025	0.02	0.021	0.019

better formatted table can be found in here :

https://drive.google.com/file/d/1uj3PobGKKyeL4ppajFV01n5kXgPKw-JG/view?usp=sharing

Parallel program using MPI :

Algorithm :

 # each process will take two vectors of the same size and will
 # return the sum of (arr1[i] * arr2[i]) for each i , which will be the
 # result of some index in the result matrix
 # process 0 will calculate for each index in the result matrix
 # which process will calculate it , then it sends to each process
 # the number of indices it will calculate , then it will send to
 # them the row number and the column number followed by
 # the row from the first matrix and the column from the
 # second matrix as two arrays
 # then process 0 will collect the results from each process and
 # put them in the result matrix

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "mpi.h"
int *generate1Darray(int n){
    return (int *) malloc(n * sizeof(int));
}
int **generate2Darray(int r, int c) {
    int **arr = (int **) malloc(r * sizeof(int *));
    for (int i = 0; i < r; i++)
        arr[i] = generate1Darray(c);
    return arr;
}
int main(int argc, char *argv[]) {
    int my_rank;       /* rank of process */
    int totalNumberOfProcesses;          /* number of process
*/
    MPI_Status status;
    double t1,t2;
    /* Start up MPI */
    MPI_Init(&argc, &argv);
    /* Find out process rank */
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    /* Find out number of process */
    MPI_Comm_size(MPI_COMM_WORLD,
&totalNumberOfProcesses);
    int start=24,end=1024,stp=100;
    double init=1;
    for(int i=start ; i<=end ; i+=stp) {
        t1 = MPI_Wtime();
        //   MPI_Send( message, strlen(message)+1, MPI_CHAR,
0, 0, MPI_COMM_WORLD);
        //   MPI_Recv(message, 100, MPI_CHAR, 0, 0,
MPI_COMM_WORLD, &status );
        if (my_rank == 0) {
            int **mat1 = generate2Darray(i, i);
            int **mat2 = generate2Darray(i, i);
            int *processCnt =
generate1Darray(totalNumberOfProcesses);
```

```
            // taskProcess[x][y] = the process id that will solve for
mat[x][y]
            int **taskProcess = generate2Darray(i, i);
            for (int j = 0; j < totalNumberOfProcesses; j++) {
                processCnt[j] = 0;
            }
            int pId = 0;
            for (int j = 0; j < i; j++) {
                for (int k = 0; k < i; k++) {
                    mat1[j][k] = mat2[j][k] = j * i + k;
                    processCnt[pId]++;
                    taskProcess[j][k] = pId++;
                    pId %= totalNumberOfProcesses;
                }
            }
            for (int j = 0; j < totalNumberOfProcesses; j++) {
                MPI_Send(&processCnt[j], 1, MPI_INT, j, 0,
MPI_COMM_WORLD);
                // sending the size of a row*col
                MPI_Send(&i, 1, MPI_INT, j, 0, MPI_COMM_WORLD);
            }
            // reversing mat2 so that each column will be a row
which will make it
            // consecutive in the memory
            for (int j = 0; j < i; j++) {
                for (int k = j; k < i; k++) {
                    int tmp = mat2[j][k];
                    mat2[j][k] = mat2[k][j];
                    mat2[k][j] = tmp;
                }
            }
            for (int j = 0; j < i; j++) {
                for (int k = 0; k < i; k++) {
                    // tell the process which index it's currently
working on
                    MPI_Send(&j, 1, MPI_INT, taskProcess[j][k], 0,
MPI_COMM_WORLD);
                    MPI_Send(&k, 1, MPI_INT, taskProcess[j][k], 0,
MPI_COMM_WORLD);
                    MPI_Send(mat1[j], i, MPI_INT, taskProcess[j][k],
0, MPI_COMM_WORLD);
```

```
                    MPI_Send(mat2[k], i, MPI_INT, taskProcess[j][k],
0, MPI_COMM_WORLD);
                }
            }
        }
        int myTasks, sz;
        MPI_Recv(&myTasks, 1, MPI_INT, 0, 0,
MPI_COMM_WORLD, &status);
        MPI_Recv(&sz, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
&status);
        int *row = generate1Darray(sz);
        int *col = generate1Darray(sz);
        while (myTasks--) {
            int j, k;
            MPI_Recv(&j, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
&status);
            MPI_Recv(&k, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
&status);
            MPI_Recv(row, sz, MPI_INT, 0, 0, MPI_COMM_WORLD,
&status);
            MPI_Recv(col, sz, MPI_INT, 0, 0, MPI_COMM_WORLD,
&status);
            int ans = 0;
            for (int x = 0; x < sz; x++) {
                ans += row[x] * col[x];
            }
            MPI_Send(&j, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
            MPI_Send(&k, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
            MPI_Send(&ans, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
        }
        if (my_rank == 0) {
            int **taskProcess = generate2Darray(sz, sz);
            int pId = 0;
            for (int j = 0; j < sz; j++) {
                for (int k = 0; k < sz; k++) {
                    taskProcess[j][k] = pId++;
                    pId %= totalNumberOfProcesses;
                }
            }
            int **res = generate2Darray(sz, sz);
            for (int f = 0; f < sz; f++) {
```

```
                for (int l = 0; l < sz; l++) {
                    int j, k, ans;
                    MPI_Recv(&j, 1, MPI_INT, taskProcess[f][l], 0,
MPI_COMM_WORLD, &status);
                    MPI_Recv(&k, 1, MPI_INT, taskProcess[f][l], 0,
MPI_COMM_WORLD, &status);
                    MPI_Recv(&ans, 1, MPI_INT, taskProcess[f][l], 0,
MPI_COMM_WORLD, &status);
                    res[j][k] = ans;
                    //   printf("%d ",ans);
                }
                //puts("");
            }
        }
        if(my_rank==0) {
            FILE *f2=fopen("/media/abdo/New Volume1/My work/
MPISpeedUp.ods","a");
            FILE *f=fopen("/media/abdo/New Volume1/My
work/MPI.ods","a");
            FILE *f3=fopen("/media/abdo/New Volume1/My work/
MPIEff.ods","a");
            // TODO remove after the first run
            t2 = MPI_Wtime();
            double passed = (t2 - t1) * 1000;
            if(i==start){
                init=passed;
                fprintf(f,"Nom of procs/mat sz,");
                fprintf(f2,"Nom of procs/mat sz,");
                fprintf(f3,"Nom of procs/mat sz,");
                for(int j=start ; j<=end ; j+=stp){
                    printf("%d%c",i,",\n"[i==101]);
                    fprintf(f,"%d%c",i,",\n"[i==101]);
                    fprintf(f2,"%d%c",i,",\n"[i==101]);
                }
            }
            fprintf(f,"%d,",totalNumberOfProcesses);
            fprintf(f2,"%d,",totalNumberOfProcesses);
            fprintf(f3,"%d,",totalNumberOfProcesses);
            double spdUp=init/passed;
            fprintf(f,"%.3f,",passed);
            fprintf(f2,"%.3f,",spdUp);
```

```
            fprintf(f3,"%.3f,",spdUp/totalNumberOfProcesses);
            fclose(f);
            fclose(f2);
            fclose(f3);
        }
    }
    /* shutdown MPI */
    MPI_Finalize();
    return 0;
}
```

running time :

mat sz/Nom of procs	1	11	21	31	41	51	61	71	81	91	101
24	0.5984	1.7005	1.1988	1.5344	2.062	1.6608	7.4271	2.4555	2.9952	2.3184	2.7523
124	58.3728	19.8526	13.7744	14.214	17.841	9.828	11.289	15.7168	12.5685	15.268	12.1924
224	129.937	71.3268	63.2553	66.2014	62.5481	61.4432	60.9168	60.3744	77.498	60.3375	57.5325
324	351.12	239.674	215.476	192.997	234.061	191.941	240.19	282.217	253.384	227.093	300.864
424	1175.49	477.258	545.471	702.051	957.444	698.403	1024.63	1448.67	1267	844.178	894.246
524	3525.81	1026.9	929.205	1043.9	887.762	1358.66	1768.64	1785.57	1831.27	1737.04	1947.31
624	7620.99	2612.52	2648.81	3706.02	2522.03	2883.56	2029.5	2157.83	2653.69	3952.18	3302.06
724	8563.56	3420.98	2942.76	2713.6	2724.07	2969.73	8052.11	6513.54	4235.49	5210.14	7152.6
824	13906.3	4411.76	3987.72	5504.61	6429.43	5717.14	7396.81	10654.9	7075.88	6597.36	4979.43
924	29815.9	12353.9	16840.9	15775.6	12688.6	13248.1	13814.8	14001.2	17408.5	10989.7	9308.24
1024	36592.9	21155.8	12167.1	17328.3	19357	17178.2	20284.3	21857.7	22539.7	19965.7	18848.9

better formatted table can be found in here :
https://drive.google.com/file/d/1mdgzCw7TJsJul8r-OVOyX_1gyTn8JU7U/view?usp=sharing

speed-up :

mat sz/Nom of procs	1	11	21	31	41	51	61	71	81	91	101
24	1	0.351897	0.499166	0.38999	0.290204	0.360308	0.0805698	0.243698	0.199786	0.258109	0.217418
124	1	2.94031	4.23777	4.10671	3.27183	5.93944	5.17077	3.71404	4.64437	3.82321	4.78764
224	1	1.82171	2.05417	1.96275	2.07739	2.11475	2.13302	2.15219	1.67665	2.1535	2.2585
324	1	1.46499	1.62951	1.8193	1.50012	1.82931	1.46184	1.24415	1.38572	1.54615	1.16704
424	1	2.46301	2.155	1.67437	1.22774	1.68311	1.14723	0.811427	0.927774	1.39247	1.3145
524	1	3.43345	3.79444	3.37754	3.97157	2.59506	1.99351	1.97461	1.92534	2.02978	1.81061
624	1	2.9171	2.87714	2.05638	3.02177	2.64291	3.75511	3.53178	2.87185	1.9283	2.30795
724	1	2.50325	2.91004	3.15579	3.14366	2.88362	1.06352	1.31473	2.02186	1.64363	1.19727
824	1	3.1521	3.48728	2.5263	2.16291	2.43239	1.88004	1.30516	1.96531	2.10786	2.79275
924	1	2.41348	1.77045	1.89	2.34982	2.25058	2.15826	2.12952	1.71272	2.71308	3.20317
1024	1	1.72969	3.00753	2.11174	1.89042	2.13019	1.804	1.67414	1.62349	1.83279	1.94138

better formatted table can be found in here :

https://drive.google.com/file/d/1wQSDL_3g53bvnB7EzZgdceyDQTP1oU_7/view?usp=sharing

effieciency :

mat sz/Nom of procs	1	11	21	31	41	51	61	71	81	91	101
24	1	0.0319906	0.0237698	0.0125803	0.00707814	0.00706487	0.00132082	0.00343236	0.0024665	0.00283636	0.00215266
124	1	0.267301	0.201799	0.132475	0.0798008	0.11646	0.0847667	0.0523104	0.0573379	0.0420133	0.0474024
224	1	0.16561	0.0978175	0.0633146	0.0506681	0.0414657	0.0349676	0.0303125	0.0206994	0.0236649	0.0223614
324	1	0.133181	0.0775956	0.0586872	0.0365883	0.0358689	0.0239646	0.0175232	0.0171077	0.0169907	0.0115548
424	1	0.22391	0.102619	0.0540118	0.0299448	0.0330022	0.0188071	0.0114285	0.011454	0.0153018	0.0130149
524	1	0.312132	0.180687	0.108953	0.0968676	0.0508836	0.0326806	0.0278115	0.0237696	0.0223053	0.0179268
624	1	0.265191	0.137007	0.0663349	0.0737017	0.0518218	0.0615591	0.0497434	0.0354549	0.0211901	0.022851
724	1	0.227568	0.138574	0.1018	0.0766747	0.0565415	0.0174347	0.0185174	0.0249612	0.0180619	0.0118541
824	1	0.286554	0.166061	0.0814936	0.052754	0.0476939	0.0308203	0.0183825	0.0242631	0.0231633	0.027651
924	1	0.219407	0.0843069	0.0609678	0.0573126	0.044129	0.0353813	0.0299933	0.0211447	0.029814	0.0317146
1024	1	0.157244	0.143216	0.0681207	0.0461079	0.0417685	0.0295738	0.0235795	0.020043	0.0201405	0.0192216

better formatted table can be found in here :

https://drive.google.com/file/d/1g4bju-nFH7P8pCiosk-OxK9vO_2g1pa3/view?usp=sharing