Threading Matrix Multiply

61519322 Yang Zherui

Date: 2021-5-15

> Targets:
>
> 1. A `matmul` using multiple thread acceleration.($C = A \cdot B$, we divide up rows of A for different threads)
> 2. Assign last few extra rows to different rows instead of assigning these rows to the last row only.
> 3. Specific command line arguments
> 4. Solution should be timed. (for multiple thread). and estimate the time for I/O and matrix multiplication.

Target 1: Matrix Multiply

Matrix Multiplication Function For Each Thread

In this section, I developed a matrix multiplication with low cache miss rate:

```
void * mm(void* arg)
{
    ...
    /* extract A, B, C, n, m, k from arg */
    for (int i = 0; i < n; ++i)
        for (int k_ = 0; k_ < m; ++k_)
            for (int j = 0; j < k; ++j)
                C[i * k + j] += A[i * m + k_] * B[k_ * k + j];
}
```

The matrix multiplication always happens here. To divide up the rows of input matrix A and the output matrix C, we can give a bias to `A` and `C` when using multiple thread: if we use `start` to represent the number of first line for each thread, and `end` to represent the number of last line for each thread, we have:

1. `A = matA + start * m`
2. `B = matB + start * m`

Secondly, when it comes to passing arguments to each thread, a pointer (`void*`) should be passed to each thread. To pass multiple arguments to each thread (for function `mm`), a struct as follow should be implemented:

```
struct mm_info {
    float *A, *B, *out;
    int n, m, k;
};
```

In this case, we can implement the function `mm` as follows:

```
void *mm(void *info)
{
    /* 提取需要的信息 */
    struct mm_info *pinfo = (struct mm_info *)info;
    float *A = pinfo->A;
    float *B = pinfo->B;
    int n = pinfo->n,
```

```
        m = pinfo->m,
        k = pinfo->k;
    float *C = pinfo->out;
    /* 清空该线程需要的 C （子）矩阵 */
    for (int i = 0; i < n * k; ++i)
        C[i] = 0.0f;

    for (int i = 0; i < n; ++i)
        for (int k_ = 0; k_ < m; ++k_)
            for (int j = 0; j < k; ++j)
                C[i * k + j] += A[i * m + k_] * B[k_ * k + j];
}
```

Assign last few threads with extra rows

In this section, use:

1. `nthread` to denote the number of threads we use in the progress.
2. `rows_each_thread` to denote the number of rows for each normal threads to do `mm()`.
3. `extra_rows_to_assign` to denote the number of threads to do `mm()` with extra lines.
4. `n` to denote the number of rows of matrix A
5. `it` to denote the rank of each thread

To obtain a better performance for the whole progress, we can use:

1. `int rows_each_thread = n / nthread;`
2. `int extra_rows_to_assign =  nthread - n + rows_each_thread * nthread;`

Therefore, to generate each thread:

```
/* 存储各个线程矩阵乘法需要的信息 */
struct mm_info *t_info = malloc(sizeof(struct mm_info) * nthread);
/* 存储各个线程的 tid */
pthread_t *tid = malloc(sizeof(pthread_t) * nthread);
for (int it = 0; it < nthread; ++it){
    int start = rows_each_thread * it;
    if (it >= extra_rows_to_assign)
        start += it - extra_rows_to_assign;
    int end = start + rows_each_thread;
    if (it >= extra_rows_to_assign)
        end += 1;
```

```
    if (end > n_A) end = n_A;
    /* 填写线程参数 */
    t_info[it].A = A + start * m_A;
    t_info[it].B = B;
    t_info[it].n = end - start;
    t_info[it].m = m_A;
    t_info[it].k = m_B;
    t_info[it].out = C + start * m_C;
    /* 创建线程 */
    pthread_create(tid + it, NULL, mm, t_info + it);
}
```

Moreover: Bind thread to specific CPU core

To maximize the performance, we can manually bind threads to specific CPU cores by:

```
cpu_set_t cpu;
CPU_ZERO(&cpu);
CPU_SET(it % 8, &cpu);
...
pthread_setaffinity_np(tid[it], sizeof(cpu_set_t), &cpu);
```

Then, each thread will be allocated averagely to each CPU core.

Target 2: Command Line Arguments and Timer

Command Line

In this program, there is only one format of command line arguments can be passed.

```
./mm -a Amat.txt -b Bmat.txt -t 4
```

Therefore, the preprocess section can be written as follows:

```
int main(int argc, char **argv)
```

```
{
    if (argc != 7)
    {
        printf("[Error] Illegal Param For matmul.");
        return 1;
    }

    int m_A, m_B, n_A, n_B;
    float *A, *B, *C;
    float start,
          mm_start,
          end;
    A = read_matrix(&m_A, &n_A, argv[2]);
    B = read_matrix(&m_B, &n_B, argv[4]);
    int nthread = atoi(argv[6]);
    int m_C = m_B,
        n_C = n_A;
    C = (float *)malloc(sizeof(float) * n_C * m_C);

    /* Do mm ...*/

    /* Free mm spaces */
    return 0;
}
```

Timer

> *There is **always a bunch of problems** when it comes to implement a timer when using multiple thread programming!*

There is several ways to get the running time of a program:

1. `time(NULL)` (in time.h) get the time measured in seconds: the output for a multiple thread program is just the same as a single thread program;
2. `clock()` (in time.h) get the CPU clock of this program, divide `CLOCKS_PER_SEC` to get the CPU time elapsed during the program. However, this will make a difference under the environment of multi-thread programming. This function will return the TOTAL clock of this progress, including every thread's clock ticks.
3. `clock_gettime(clockid_t clk_id, struct timespec *res)` : finds the resolution (precision) of the specified clock `clk_id` , and, if `res` is non-NULL, stores it in the struct `timespec` pointed to by res. If we use `clk_id = CLOCK_MONOTONIC` , we can

> get the precise time corresponds to the number of seconds that the system has been running since it was booted.

Therefore, we can call `clock_gettime(CLOCK_MONOTONIC, &t)` to get the number of seconds elapsed during my program, regardless that the program is running under a multi-thread environment.

```
#define TIME_MS(t) (((float) t.tv_sec) * 1000.f + ((float) t.tv_nsec) / 1e6f)
...
// in main function
struct timespec t;
clock_gettime(CLOCK_MONOTONIC, &t);
start = TIME_MS(t);
...
```

Experiments:

Setup

To run the whole test, I use a shell script as follows:

```
gcc main.c -O2 -lpthread -pthread -o mm
for size in 2 4 8 10 100 200 500 1000 2000 4000 8000
do
    echo "##########################" $size "##########################"
    python gm.py $size $size "Amat$size.txt"
    python gm.py $size $size "Bmat$size.txt"
    for nt in 1 2 4 8 16
    do
        echo "mm -a Amat$size.txt -b Bmat$size.txt -t $nt"

        time -p ./mm -a Amat$size.txt -b Bmat$size.txt -t $nt
    done
done
echo "[DONE.]"
```

and run it using the following command to get the whole log.

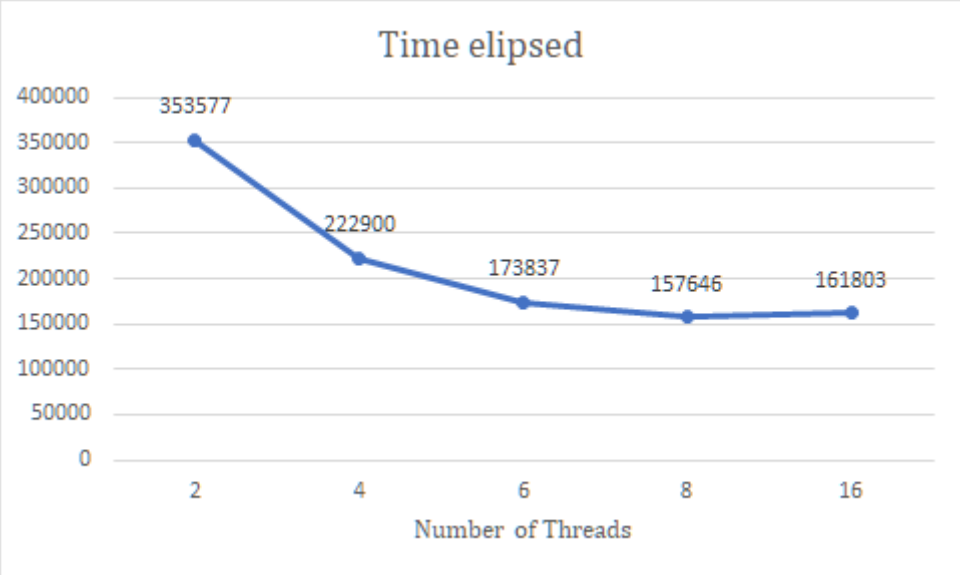
```
$ sh test.sh > log.txt
```

Discussion about the result

-t\size	2	4	8	10	100	200	500	1000	2000	4000	8000
2	0.5	0.5	0.5	5	13.5	44.5	302	1428	8395	50825	353577
4	1	1	1.5	1	14	44.5	286	1184	6355	34971	222900
6	1	1.5	2	1	14	47.5	254	1069	5214	28159	173837
8	1.5	1.5	1.5	1.5	16	41.5	242	1044	4841	24701	157646
16	1.5	1.5	2	1.5	18	48	252	1191	4892	26052	161803

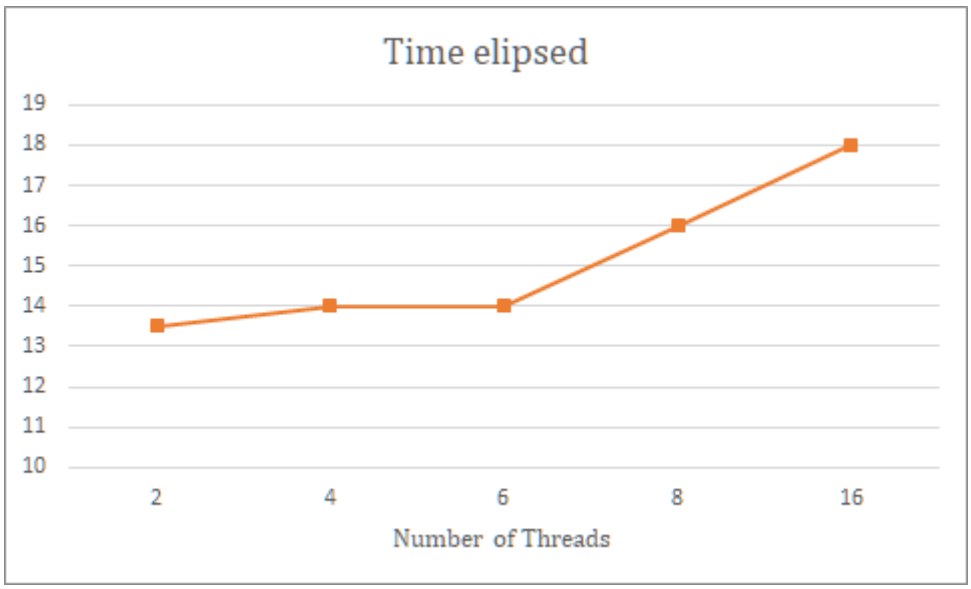
Table: Time elipsed (in ms) using different arguments for the program (-t = number of threads, size = the size of matrices) Tested on a 4-core 8-thread CPU

Take the size = 8000 as an example :



It is obvious that, with multiple threads, there will be a obvious acceleration for matrix multiplication. But when the thread is greater that the number of CPU cores, the performance will be lower than a small thread number.

Another example is size = 100:



With a low size but a high number of threads, more time will be spent on creation and joining of threads.

Codes and Logs

1. `main.c` matrix multiplication program
2. `log.txt` test log.

main.c

```
#define _GNU_SOURCE
#include <stdlib.h>
#include <pthread.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <sched.h>
#define TIME_MS(t) (((float) t.tv_sec) * 1000.f + ((float) t.tv_nsec) /
1e6f)

#define NDEBUG


struct mm_info
{
    float *A;
```

```
    float *B;
    int n, m, k;
    float *out;
};

// do matmul store in out
void *mm(void *info)
{
    struct mm_info *pinfo = (struct mm_info *)info;
    float *A = pinfo->A;
    float *B = pinfo->B;
    int n = pinfo->n,
        m = pinfo->m,
        k = pinfo->k;
    float *C = pinfo->out;

#ifndef NDEBUG
    printf("[Info] I'm going to do matmul...\n");
    printf("\tMat A: address = %p\n", A);
    printf("\tMat B: address = %p\n", B);
    printf("\tMat C: address = %p\n", C);
    printf("\tn, m, k = %d, %d, %d\n", n, m, k);
#endif
    for (int i = 0; i < n * k; ++i)
        C[i] = 0.0f;

    for (int i = 0; i < n; ++i)
    {
        for (int k_ = 0; k_ < m; ++k_)
        {
            for (int j = 0; j < k; ++j)
            {
                C[i * k + j] += A[i * m + k_] * B[k_ * k + j];
            }
        }
    }
}

// read from text
float *read_matrix(int *m, int *n, char *fn)
{
    FILE *file = fopen(fn, "r");
    if (file == NULL)
```

```
    {
        printf("[Error] I cannot read file %s.\n", fn);
    }
    fscanf(file, "%d", n);
    fscanf(file, "%d", m);
    float *mat = (float *)malloc(sizeof(float) * (*n) * (*m));
    for (int i = 0; i < (*n) * (*m); ++i)
    {
        fscanf(file, "%f", &(mat[i]));
    }

    fclose(file);
    return mat;
}

// write to text
void write_matrix(float *mat, int *m, int *n, char *fn)
{
    FILE *file = fopen(fn, "w");
    fprintf(file, "%d %d\n", *m, *n);
    for (int i = 0; i < *n; ++i)
    {
        for (int j = 0; j < *m; ++j)
        {
            fprintf(file, "%f ", mat[i * (*m) + j]);
        }
        fprintf(file, "\n");
    }
    fclose(file);
}

int main(int argc, char **argv)
{
    // mm -a mata.txt -b matb.txt -t nthreads
    if (argc != 7)
    {
        printf("[Error] Illegal Param For matmul.");
        return 1;
    }

    int m_A, m_B, n_A, n_B;
    float *A, *B, *C;
    float start,
```

```
            mm_start,
            end;

    struct timespec t;
    clock_gettime(CLOCK_MONOTONIC, &t);
    start = TIME_MS(t);

#ifndef NDEBUG
    printf("[Info] I wanna read A from %s and B from %s...\n", argv[2],
argv[4]);
#endif

    A = read_matrix(&m_A, &n_A, argv[2]);
    B = read_matrix(&m_B, &n_B, argv[4]);
#ifndef NDEBUG
    printf("[Info] I got mat A <at %p> and mat B <at %p>\n", A, B);
#endif

    int nthread = atoi(argv[6]);
    if (m_A != n_B)
    {
        printf("[Error] Illegal shape For matmul, got m1, m2 = [%d, %d]",
m_A, n_B);
    }

    int m_C = m_B,
        n_C = n_A;
    C = (float *)malloc(sizeof(float) * n_C * m_C);


    int rows_each_thread = n_A / nthread;
    int extra_rows_to_assign =  nthread - n_A + rows_each_thread *
nthread;

    struct mm_info *t_info = malloc(sizeof(struct mm_info) * nthread);
    pthread_t *tid = malloc(sizeof(pthread_t) * nthread);
    clock_gettime(CLOCK_MONOTONIC, &t);
    mm_start = TIME_MS(t);
    printf("[Info] Done IO in %lf ms\n", (((double) (mm_start -
start))));
    for (int it = 0; it < nthread; ++it)
    {
        cpu_set_t cpu;
```

```
        CPU_ZERO(&cpu);
        CPU_SET(it % 8, &cpu);
        int start = rows_each_thread * it;
        if (it >= extra_rows_to_assign)
            start += it - extra_rows_to_assign;
        int end = start + rows_each_thread;
        if (it >= extra_rows_to_assign)
            end += 1;
        if (end > n_A) end = n_A;
        t_info[it].A = A + start * m_A;
        t_info[it].B = B;
        t_info[it].n = end - start;
        t_info[it].m = m_A;
        t_info[it].k = m_B;
        t_info[it].out = C + start * m_C;
#ifndef NDEBUG
        printf("n, m, k = %d, %d, %d\n", t_info[it].n, t_info[it].m,
t_info[it].k);
#endif
        pthread_create(tid + it, NULL, mm, t_info + it);
        pthread_setaffinity_np(tid[it], sizeof(cpu_set_t), &cpu);
    }

    for (int it = 0; it < nthread; ++it){
        pthread_join(tid[it], NULL);
    }
    clock_gettime(CLOCK_MONOTONIC, &t);
    end = TIME_MS(t);
    printf("[Info] Done MM in %lf ms...\n", (((double) (end -
mm_start))));

    write_matrix(C, &m_C, &n_C, "result.txt");

    clock_gettime(CLOCK_MONOTONIC, &t);
    end = TIME_MS(t);
    printf("[Info] Done in %lf ms...\n", ((double) (end - start)));



    free(A); A = NULL;
    free(B); B = NULL;
    free(C); C = NULL;
    free(tid); tid = NULL;
    free(t_info); t_info = NULL;
```

```
    return 0;
}
```

log.txt

```
########################## 2 ##########################
mm -a Amat2.txt -b Bmat2.txt -t 1
[Info] Done IO in 0.500000 ms
[Info] Done MM in 0.000000 ms...
[Info] Done in 5.000000 ms...
mm -a Amat2.txt -b Bmat2.txt -t 2
[Info] Done IO in 0.000000 ms
[Info] Done MM in 0.500000 ms...
[Info] Done in 1.000000 ms...
mm -a Amat2.txt -b Bmat2.txt -t 4
[Info] Done IO in 0.000000 ms
[Info] Done MM in 0.500000 ms...
[Info] Done in 1.000000 ms...
mm -a Amat2.txt -b Bmat2.txt -t 8
[Info] Done IO in 0.500000 ms
[Info] Done MM in 0.500000 ms...
[Info] Done in 1.500000 ms...
mm -a Amat2.txt -b Bmat2.txt -t 16
[Info] Done IO in 0.000000 ms
[Info] Done MM in 1.000000 ms...
[Info] Done in 1.500000 ms...
########################## 4 ##########################
mm -a Amat4.txt -b Bmat4.txt -t 1
[Info] Done IO in 0.000000 ms
[Info] Done MM in 0.500000 ms...
[Info] Done in 4.500000 ms...
mm -a Amat4.txt -b Bmat4.txt -t 2
[Info] Done IO in 0.500000 ms
[Info] Done MM in 0.000000 ms...
[Info] Done in 1.000000 ms...
mm -a Amat4.txt -b Bmat4.txt -t 4
[Info] Done IO in 0.500000 ms
[Info] Done MM in 0.000000 ms...
[Info] Done in 1.500000 ms...
```

```
mm -a Amat4.txt -b Bmat4.txt -t 8
[Info] Done IO in 0.500000 ms
[Info] Done MM in 0.500000 ms...
[Info] Done in 1.500000 ms...
mm -a Amat4.txt -b Bmat4.txt -t 16
[Info] Done IO in 0.000000 ms
[Info] Done MM in 0.500000 ms...
[Info] Done in 1.500000 ms...
########################## 8 ##########################
mm -a Amat8.txt -b Bmat8.txt -t 1
[Info] Done IO in 0.500000 ms
[Info] Done MM in 0.000000 ms...
[Info] Done in 5.000000 ms...
mm -a Amat8.txt -b Bmat8.txt -t 2
[Info] Done IO in 0.500000 ms
[Info] Done MM in 0.000000 ms...
[Info] Done in 1.500000 ms...
mm -a Amat8.txt -b Bmat8.txt -t 4
[Info] Done IO in 0.500000 ms
[Info] Done MM in 0.500000 ms...
[Info] Done in 2.000000 ms...
mm -a Amat8.txt -b Bmat8.txt -t 8
[Info] Done IO in 0.500000 ms
[Info] Done MM in 0.500000 ms...
[Info] Done in 1.500000 ms...
mm -a Amat8.txt -b Bmat8.txt -t 16
[Info] Done IO in 0.500000 ms
[Info] Done MM in 0.500000 ms...
[Info] Done in 2.000000 ms...
########################## 10 ##########################
mm -a Amat10.txt -b Bmat10.txt -t 1
[Info] Done IO in 0.500000 ms
[Info] Done MM in 0.000000 ms...
[Info] Done in 5.000000 ms...
mm -a Amat10.txt -b Bmat10.txt -t 2
[Info] Done IO in 0.500000 ms
[Info] Done MM in 0.000000 ms...
[Info] Done in 1.000000 ms...
mm -a Amat10.txt -b Bmat10.txt -t 4
[Info] Done IO in 0.500000 ms
[Info] Done MM in 0.000000 ms...
[Info] Done in 1.000000 ms...
mm -a Amat10.txt -b Bmat10.txt -t 8
```

```
[Info] Done IO in 0.000000 ms
[Info] Done MM in 0.500000 ms...
[Info] Done in 1.500000 ms...
mm -a Amat10.txt -b Bmat10.txt -t 16
[Info] Done IO in 0.000000 ms
[Info] Done MM in 1.000000 ms...
[Info] Done in 1.500000 ms...
########################## 100 ##########################
mm -a Amat100.txt -b Bmat100.txt -t 1
[Info] Done IO in 4.500000 ms
[Info] Done MM in 0.500000 ms...
[Info] Done in 13.500000 ms...
mm -a Amat100.txt -b Bmat100.txt -t 2
[Info] Done IO in 5.000000 ms
[Info] Done MM in 0.500000 ms...
[Info] Done in 14.000000 ms...
mm -a Amat100.txt -b Bmat100.txt -t 4
[Info] Done IO in 4.000000 ms
[Info] Done MM in 0.500000 ms...
[Info] Done in 14.000000 ms...
mm -a Amat100.txt -b Bmat100.txt -t 8
[Info] Done IO in 5.000000 ms
[Info] Done MM in 0.500000 ms...
[Info] Done in 16.000000 ms...
mm -a Amat100.txt -b Bmat100.txt -t 16
[Info] Done IO in 5.000000 ms
[Info] Done MM in 1.000000 ms...
[Info] Done in 18.000000 ms...
########################## 200 ##########################
mm -a Amat200.txt -b Bmat200.txt -t 1
[Info] Done IO in 17.000000 ms
[Info] Done MM in 5.000000 ms...
[Info] Done in 44.500000 ms...
mm -a Amat200.txt -b Bmat200.txt -t 2
[Info] Done IO in 17.500000 ms
[Info] Done MM in 2.500000 ms...
[Info] Done in 44.500000 ms...
mm -a Amat200.txt -b Bmat200.txt -t 4
[Info] Done IO in 19.500000 ms
[Info] Done MM in 2.000000 ms...
[Info] Done in 47.500000 ms...
mm -a Amat200.txt -b Bmat200.txt -t 8
[Info] Done IO in 17.000000 ms
```

```
[Info] Done MM in 2.000000 ms...
[Info] Done in 41.500000 ms...
mm -a Amat200.txt -b Bmat200.txt -t 16
[Info] Done IO in 20.500000 ms
[Info] Done MM in 2.000000 ms...
[Info] Done in 48.000000 ms...
########################## 500 ##########################
mm -a Amat500.txt -b Bmat500.txt -t 1
[Info] Done IO in 114.000000 ms
[Info] Done MM in 70.000000 ms...
[Info] Done in 302.000000 ms...
mm -a Amat500.txt -b Bmat500.txt -t 2
[Info] Done IO in 117.500000 ms
[Info] Done MM in 48.000000 ms...
[Info] Done in 286.000000 ms...
mm -a Amat500.txt -b Bmat500.txt -t 4
[Info] Done IO in 107.500000 ms
[Info] Done MM in 26.000000 ms...
[Info] Done in 254.500000 ms...
mm -a Amat500.txt -b Bmat500.txt -t 8
[Info] Done IO in 106.500000 ms
[Info] Done MM in 18.500000 ms...
[Info] Done in 242.000000 ms...
mm -a Amat500.txt -b Bmat500.txt -t 16
[Info] Done IO in 116.000000 ms
[Info] Done MM in 18.500000 ms...
[Info] Done in 252.500000 ms...
########################## 1000 ##########################
mm -a Amat1000.txt -b Bmat1000.txt -t 1
[Info] Done IO in 440.500000 ms
[Info] Done MM in 524.000000 ms...
[Info] Done in 1428.000000 ms...
mm -a Amat1000.txt -b Bmat1000.txt -t 2
[Info] Done IO in 421.500000 ms
[Info] Done MM in 292.000000 ms...
[Info] Done in 1184.000000 ms...
mm -a Amat1000.txt -b Bmat1000.txt -t 4
[Info] Done IO in 421.000000 ms
[Info] Done MM in 187.000000 ms...
[Info] Done in 1069.500000 ms...
mm -a Amat1000.txt -b Bmat1000.txt -t 8
[Info] Done IO in 430.500000 ms
[Info] Done MM in 148.500000 ms...
```

```
[Info] Done in 1044.000000 ms...
mm -a Amat1000.txt -b Bmat1000.txt -t 16
[Info] Done IO in 445.500000 ms
[Info] Done MM in 169.500000 ms...
[Info] Done in 1191.000000 ms...
######################### 2000 #########################
mm -a Amat2000.txt -b Bmat2000.txt -t 1
[Info] Done IO in 1749.000000 ms
[Info] Done MM in 4744.000000 ms...
[Info] Done in 8395.000000 ms...
mm -a Amat2000.txt -b Bmat2000.txt -t 2
[Info] Done IO in 1690.500000 ms
[Info] Done MM in 2847.000000 ms...
[Info] Done in 6355.000000 ms...
mm -a Amat2000.txt -b Bmat2000.txt -t 4
[Info] Done IO in 1674.000000 ms
[Info] Done MM in 1606.500000 ms...
[Info] Done in 5214.500000 ms...
mm -a Amat2000.txt -b Bmat2000.txt -t 8
[Info] Done IO in 1702.000000 ms
[Info] Done MM in 1274.500000 ms...
[Info] Done in 4841.000000 ms...
mm -a Amat2000.txt -b Bmat2000.txt -t 16
[Info] Done IO in 1790.500000 ms
[Info] Done MM in 1276.500000 ms...
[Info] Done in 4862.000000 ms...
######################### 4000 #########################
mm -a Amat4000.txt -b Bmat4000.txt -t 1
[Info] Done IO in 6706.500000 ms
[Info] Done MM in 36953.500000 ms...
[Info] Done in 50825.500000 ms...
mm -a Amat4000.txt -b Bmat4000.txt -t 2
[Info] Done IO in 6621.000000 ms
[Info] Done MM in 21158.500000 ms...
[Info] Done in 34971.000000 ms...
mm -a Amat4000.txt -b Bmat4000.txt -t 4
[Info] Done IO in 6584.500000 ms
[Info] Done MM in 14384.500000 ms...
[Info] Done in 28159.500000 ms...
mm -a Amat4000.txt -b Bmat4000.txt -t 8
[Info] Done IO in 6634.000000 ms
[Info] Done MM in 10832.000000 ms...
[Info] Done in 24701.000000 ms...
```

```
mm -a Amat4000.txt -b Bmat4000.txt -t 16
[Info] Done IO in 6758.500000 ms
[Info] Done MM in 12039.500000 ms...
[Info] Done in 26052.000000 ms...
########################## 8000 ##########################
mm -a Amat8000.txt -b Bmat8000.txt -t 1
[Info] Done IO in 30754.000000 ms
[Info] Done MM in 292763.000000 ms...
[Info] Done in 353577.500000 ms...
mm -a Amat8000.txt -b Bmat8000.txt -t 2
[Info] Done IO in 26304.000000 ms
[Info] Done MM in 166687.000000 ms...
[Info] Done in 222900.000000 ms...
mm -a Amat8000.txt -b Bmat8000.txt -t 4
[Info] Done IO in 26186.000000 ms
[Info] Done MM in 117384.000000 ms...
[Info] Done in 173837.500000 ms...
mm -a Amat8000.txt -b Bmat8000.txt -t 8
[Info] Done IO in 26474.500000 ms
[Info] Done MM in 100375.000000 ms...
[Info] Done in 157646.500000 ms...
mm -a Amat8000.txt -b Bmat8000.txt -t 16
[Info] Done IO in 26634.500000 ms
[Info] Done MM in 104828.000000 ms...
[Info] Done in 161803.500000 ms...
[DONE.]
```