中山大学计算机院本科生实验报告

（2023 学年秋季学期）

课程名称：高性能计算程序设计　　　　　　　　　　　　**批改人：**

实验	通用矩阵乘法	专业（方向）	信息与计算科学
学号	21311359	姓名	何凯迪
Email	hekd@mail2.sysu.edu.cn	完成日期	2023/8/28

1. 实验目的

（1）通过实验 3 构造的基于 Pthreads 的 parallel_for 函数替换 fft_serial 应用中的某些计算量较大的“for 循环”,实现 for 循环分解、分配和线程并行执行。

（2）将 heated_plate_openmp 应用改造成基于 MPI 的进程并行应用。Bonus:使用 MPI_Pack/MPI_Unpack，或 MPI_Type_create_struct 实现数据重组后的消息传递。

（3）性能分析任务：对任务 1 实现的并行化 fft 应用在不同规模下的性能进行分析，即分析：

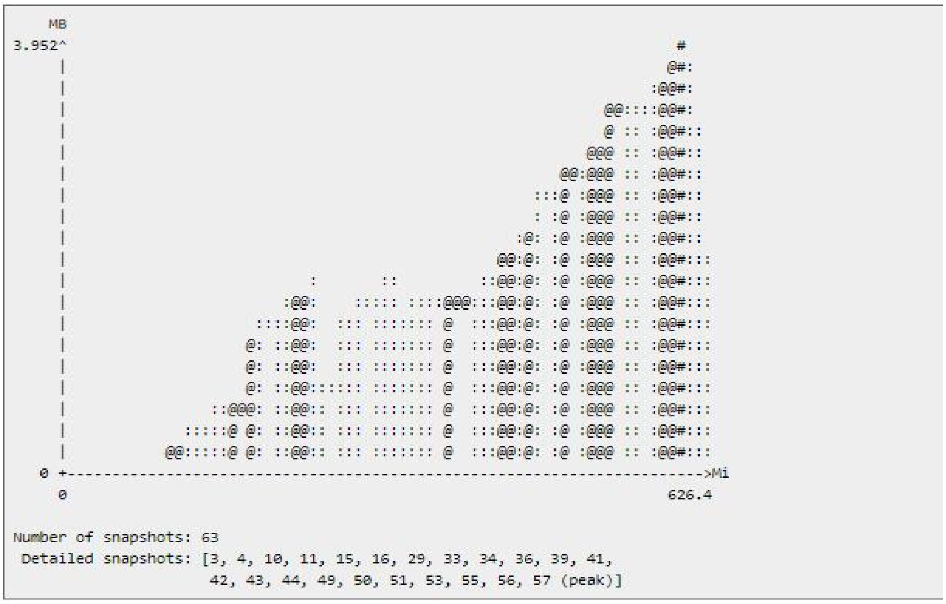
1）不同规模下的并行化 fft 应用的执行时间对比；

2）不同规模下的并行化 fft 应用的内存消耗对比。

本题中,“规模”定义为“问题规模”和“并行规模”;“性能”定义为“执行时间”和“内存消耗”。

其中，问题规模 N，值为 2，4，6，8，16，32，64，128，……，2097152；并行规模，值为 1，2，4，8 进程/线程。

内存消耗采用 “valgrind-tool=massif--time-unit=B ./your_exe”工具采集，注意命令 valgrind 命令中增加--stacks=yes 参数采集程序运行栈内内存消耗。Valgrind –tool=massif 输出日志（massif.out.pid）经过 ms_print 打印后示例如下图，其中 x 轴为程序运行时间，y 轴为内存消耗量：



2. 实验过程和核心代码

（1）通过实验 3 构造的基于 Pthreads 的 parallel_for 函数替换 fft_serial 应用中的某些计算量较大的“for 循环”，实现 for 循环分解、分配和线程并行执行。

以下本小题呈现的运行结果皆在 num_threads 设置为 4 时编译运行。

首先由 fft_serial_test.txt 得到原始运行结果如下：

Accuracy check:

FFT (FFT (X(1:N))) == N * X(1:N)

N	NITS	Error	Time	Time/Call	MFLOPS
2	10000	7.859082e-17	1.140000e-03	5.700000e-08	175.438596
4	10000	1.209837e-16	2.556000e-03	1.278000e-07	312.989045
8	10000	6.820795e-17	4.595000e-03	2.297500e-07	522.306855
16	10000	1.438671e-16	1.239700e-02	6.198500e-07	516.253932
32	1000	1.331210e-16	2.340000e-03	1.170000e-06	683.760684
64	1000	1.776545e-16	6.231000e-03	3.115500e-06	616.273471
128	1000	1.929043e-16	1.104800e-02	5.524000e-06	811.006517
256	1000	2.092319e-16	2.790300e-02	1.395150e-05	733.971258
512	100	1.927488e-16	5.153000e-03	2.576500e-05	894.236367
1024	100	2.308607e-16	1.309000e-02	6.545000e-05	782.276547
2048	100	2.447624e-16	2.927300e-02	1.463650e-04	769.582892
4096	100	2.479782e-16	6.133200e-02	3.066600e-04	801.408726
8192	10	2.578088e-16	1.286300e-02	6.431500e-04	827.925056
16384	10	2.733986e-16	2.714000e-02	1.357000e-03	845.158438
32768	10	2.923012e-16	5.931600e-02	2.965800e-03	828.646571
65536	10	2.829927e-16	1.297160e-01	6.485800e-03	808.362885
131072	1	3.149670e-16	2.688600e-02	1.344300e-02	828.767388
262144	1	3.218597e-16	5.723200e-02	2.861600e-02	824.467431
524288	1	3.281373e-16	1.228870e-01	6.144350e-02	810.620489
1048576	1	3.285898e-16	2.796540e-01	1.398270e-01	749.909531

用于后续对比。

本题共需要三个文件：

g++ -shared -fPIC -o libparallel_for.so parallel_for.cpp lpthread 生成.so 

Export LD_LIBRARY_PATH=/path/to/library:$LD_LIBRARY_PATH

g++ -o fft_serial fft_serial.cpp -I/path/to/parallel -L. -lparallel_for -lpthread -lm 生成 

./fft_serial 运行

下面是实验过程详述：

起初我编写的 parallel_for 如下：

```c
#include <stdio.h>
#include <cstdlib>
#include <pthread.h>
#include "parallel_for.h"

void *worker(void *args) {
    struct for_index *index = (struct for_index *)args;
    for (int i = index->start; i < index->end; i += index->increment) {
        index->functor(index->arg, i);
    }
    return NULL;
}

void parallel_for(int start, int end, int increment, void *(*functor)(void *, int), void *arg, int num_threads) {
    pthread_t threads[num_threads];

    int chunk_size = (end - start + num_threads - 1) / num_threads;
    for (int i = 0; i < num_threads; ++i) {
        struct for_index index;
        index.start = i * chunk_size;
        index.end = (i + 1) * chunk_size;
        index.increment = increment;
        index.functor = functor;
        index.arg = arg;

        if (pthread_create(&threads[i], NULL, worker, (void *)&index) != 0) {
            perror("Error creating thread");
            exit(EXIT_FAILURE);
        }
    }

    for (int i = 0; i < num_threads; ++i) {
        if (pthread_join(threads[i], NULL) != 0) {
            perror("Error joining thread");
            exit(EXIT_FAILURE);
        }
    }
}
```

再将 fft_serial 作相应改造，可见 Error 值一致但性能结果不尽人意：

```
 Accuracy check:

   FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

             N      NITS    Error         Time          Time/Call     MFLOPS

             2     10000  7.859082e-17  7.880000e-04  3.940000e-08    253.807107
             4     10000  1.209837e-16  5.531817e+00  2.765908e-04      0.144618
             8     10000  6.820795e-17  1.339000e-03  6.695000e-08   1792.382375
            16     10000  1.438671e-16  5.578883e+00  2.789441e-04      1.147183
            32      1000  1.331210e-16  3.750000e-04  1.875000e-07   4266.666667
            64      1000  1.776545e-16  5.719560e-01  2.859780e-04      6.713803
           128      1000  1.929043e-16  1.556000e-03  7.780000e-07   5758.354756
           256      1000  2.092319e-16  1.483372e+00  7.416860e-04     13.806382
           512       100  1.927488e-16  8.160000e-04  4.080000e-06   5647.058824
          1024       100  2.308607e-16  3.120105e+00  1.560053e-02      3.281941
          2048       100  2.447624e-16  4.494000e-03  2.247000e-05   5012.906097
          4096       100  2.479782e-16  3.181354e+01  1.590677e-01      1.545003
          8192        10  2.578088e-16  2.201000e-03  1.100500e-04   4838.527942
         16384        10  2.733986e-16  3.335407e+01  1.667704e+00      0.687700
         32768        10  2.923012e-16  8.295000e-03  4.147500e-04   5925.497288
^C
```

尽管我采用了-O2(-O3)编译依然会有这种相邻性能差异极大的情况，以至于 N=1048576 时迟迟没有输出。在对 fft_serial.cpp 不断优化后仍得不到改善，于是我将目光放到底层的 parallel_for 函数上。在初步尝试优化中，我认为其中两部分判断线程创建和结束的部分有些冗余，于是将其删去：

```
#include <stdio.h>
#include <cstdlib>
#include <pthread.h>
#include "parallel_for.h"

void *worker(void *args) {
    struct for_index *index = (struct for_index *)args;
    for (int i = index->start; i < index->end; i += index->increment) {
        index->functor(index->arg, i);
    }
    return NULL;
}

void parallel_for(int start, int end, int increment, void *(*functor)(void *, int), void *arg, int num_threads) {
    pthread_t threads[num_threads];

    int chunk_size = (end - start + num_threads - 1) / num_threads;
    for (int i = 0; i < num_threads; ++i) {
        struct for_index index;
        index.start = i * chunk_size;
        index.end = (i + 1) * chunk_size;
        index.increment = increment;
        index.functor = functor;
        index.arg = arg;
    }
}
```

然后再编译运行，我惊喜地发现性能结果得到极大的改善：

```
Accuracy check:

  FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

             N      NITS    Error         Time          Time/Call     MFLOPS

             2     10000  7.859082e-17  7.780000e-04  3.890000e-08    257.069409
             4     10000  4.856852e-01  2.396000e-03  1.198000e-07    333.889816
             8     10000  6.820795e-17  1.324000e-03  6.620000e-08   1812.688822
            16     10000  7.616240e-01  3.409000e-03  1.704500e-07   1877.383397
            32      1000  1.331210e-16  3.620000e-04  1.810000e-07   4419.889503
            64      1000  7.584636e-01  8.590000e-04  4.295000e-07   4470.314319
           128      1000  1.929043e-16  1.893000e-03  9.465000e-07   4733.227681
           256      1000  7.625348e-01  3.622000e-03  1.811000e-06   5654.334622
           512       100  1.927488e-16  7.350000e-04  3.675000e-06   6269.387755
          1024       100  7.638633e-01  1.609000e-03  8.045000e-06   6364.201367
          2048       100  2.447624e-16  3.480000e-03  1.740000e-05   6473.563218
          4096       100  7.595243e-01  7.581000e-03  3.790500e-05   6483.577364
          8192        10  2.578088e-16  1.610000e-03  8.050000e-05   6614.658385
         16384        10  7.576564e-01  3.583000e-03  1.791500e-04   6401.786213
         32768        10  2.923012e-16  7.992000e-03  3.996000e-04   6150.150150
         65536        10  7.569469e-01  1.631800e-02  8.159000e-04   6425.885525
        131072         1  3.149670e-16  3.354000e-03  1.677000e-03   6643.482409
        262144         1  7.572501e-01  7.593000e-03  3.796500e-03   6214.397471
        524288         1  3.281373e-16  2.098800e-02  1.049400e-02   4746.270250
       1048576         1  7.566535e-01  6.257700e-02  3.128850e-02   3351.314381
```

于是我的 parallel_for 函数便如上方所示。

但此时 Error 值又出现了交替异常的情况，可见 N=2 时正常，N=4 时异常，N=8 时正常...以此类推。

我猜测是删除了线程创建和等待完成部分导致的，于是稍作修改并重新添加回来：

```c
#include <stdio.h>
#include <cstdlib>
#include <pthread.h>
#include "parallel_for.h"

void *worker(void *args) {
    struct for_index *index = (struct for_index *)args;
    for (int i = index->start; i < index->end; i += index->increment) {
        index->functor(index->arg, i);
    }
    return NULL;
}

void parallel_for(int start, int end, int increment, void *(*functor)(void *, int), void *arg, int num_threads) {
    pthread_t threads[num_threads];
    struct for_index indices[num_threads];

    int chunk_size = (end - start + num_threads - 1) / num_threads;
    for (int i = 0; i < num_threads; ++i) {
        indices[i].start = i * chunk_size;
        indices[i].end = (i + 1) * chunk_size;
        indices[i].increment = increment;
        indices[i].functor = functor;
        indices[i].arg = arg;

        // 创建线程并传递相应的结构体
        pthread_create(&threads[i], NULL, worker, (void *)&indices[i]);
    }

    // 等待线程完成
    for (int i = 0; i < num_threads; ++i) {
        pthread_join(threads[i], NULL);
    }
}
```

此时 Error 值正常，性能又异常：

```
 Accuracy check:

   FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

             N      NITS    Error          Time           Time/Call     MFLOPS

             2     10000  7.859082e-17  7.870000e-04  3.935000e-08    254.129606
             4     10000  1.209837e-16  1.978432e+00  9.892160e-05      0.404361
             8     10000  6.820795e-17  1.280000e-03  6.400000e-08   1875.000000
            16     10000  1.438671e-16  2.021694e+00  1.010847e-04      3.165662
            32      1000  1.331210e-16  3.830000e-04  1.915000e-07   4177.545692
            64      1000  1.776545e-16  2.139970e-01  1.069985e-04     17.944177
           128      1000  1.929043e-16  1.595000e-03  7.975000e-07   5617.554859
           256      1000  2.092319e-16  4.071990e-01  2.035995e-04     50.294819
           512       100  1.927488e-16  7.430000e-04  3.715000e-06   6201.884253
          1024       100  2.308607e-16  6.753510e-01  3.376755e-03     15.162486
          2048       100  2.447624e-16  3.986000e-03  1.993000e-05   5651.781234
          4096       100  2.479782e-16  1.018921e+01  5.094607e-02      4.823925
          8192        10  2.578088e-16  2.044000e-03  1.022000e-04   5210.176125
         16384        10  2.733986e-16  1.036080e+01  5.180399e-01      2.213884
         32768        10  2.923012e-16  8.835000e-03  4.417500e-04   5563.327674
         65536        10  2.829927e-16  7.993826e+01  3.996913e+00      1.311732
        131072         1  3.149670e-16  4.118000e-03  2.059000e-03   5410.937348
        262144         1  3.218597e-16  1.252221e+02  6.261107e+01      0.376818
        524288         1  3.281373e-16  2.252400e-02  1.126200e-02   4422.603445
```

我并不清楚原因。

再展示 parallel_for.h：

```
#ifndef PARALLEL_FOR_H
#define PARALLEL_FOR_H

#include <pthread.h>

struct for_index {
    int start;
    int end;
    int increment;
    void *(*functor)(void *, int);
    void *arg;
};

void parallel_for(int start, int end, int increment, void *(*functor)(void *, int), void *arg, int num_threads);

#endif
```

接下来着重于对 fft_serial.cpp 的改造：

首先是 ccopy 函数：

```
struct ccopy_args {
    int n;
    double *x;
    double *y;
};

void *ccopy_worker(void *args);

void ccopy(int n, double x[], double y[]) {
    struct ccopy_args ccopy_args = {n, x, y};  // 具名结构体对象
    // 使用 parallel_for 并行化循环
    parallel_for(0, n, 1, (void *(*)(void *, int))ccopy_worker, (void *)&ccopy_args, 8);
}

void *ccopy_worker(void *args) {
    struct ccopy_args *ccopy_args = (struct ccopy_args *)args;
    int n = ccopy_args->n;
    double *x = ccopy_args->x;
    double *y = ccopy_args->y;

    // 并行复制
    for (int i = 0; i < n; i++) {
        y[i * 2 + 0] = x[i * 2 + 0];
        y[i * 2 + 1] = x[i * 2 + 1];
    }

    return NULL;
}
```

先定义了一个结构体 ccopy_args，用于传递给并行函数的参数。它包含三个成员变量，分别是整数 n 和两个指向双精度浮点数数组的指针 x 和 y。

其次在 ccopy 函数中创建了一个 ccopy_args 的具名结构体对象 ccopy_args，并用传递给函数的参数进行初始化。接着，使用 parallel_for 函数并行地调用 ccopy_worker 函数，将 ccopy_args 作为参数传递，并指定线程数为 4。

最后的 ccopy_worker 工作函数与最初的代码基本相同，不过多解释。

单独改造 ccopy 的运行结果如下：

```
Accuracy check:

  FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

             N      NITS    Error         Time          Time/Call     MFLOPS

             2     10000  7.859082e-17  1.201000e-03  6.005000e-08    166.527893
             4     10000  1.209837e-16  1.670941e+00  8.354705e-05      0.478772
             8     10000  6.820795e-17  1.673000e-03  8.365000e-08   1434.548715
            16     10000  1.438671e-16  1.719526e+00  8.597630e-05      3.721956
            32      1000  1.331210e-16  5.120000e-04  2.560000e-07   3125.000000
            64      1000  1.776545e-16  1.775150e-01  8.875750e-05     21.631975
           128      1000  1.929043e-16  2.263000e-03  1.131500e-06   3959.346001
           256      1000  2.092319e-16  3.777100e-01  1.888550e-04     54.221493
           512       100  1.927488e-16  1.138000e-03  5.690000e-06   4049.209139
          1024       100  2.308607e-16  4.051100e-01  2.025550e-03     25.277085
          2048       100  2.447624e-16  5.366000e-03  2.683000e-05   4198.285501
          4096       100  2.479782e-16  8.419534e+00  4.209767e-02      5.837853
          8192        10  2.578088e-16  2.449000e-03  1.224500e-04   4348.550429
         16384        10  2.733986e-16  4.647011e+00  2.323506e-01      4.935990
         32768        10  2.923012e-16  1.269300e-02  6.346500e-04   3872.370598
         65536        10  2.829927e-16  5.673262e+01  2.836631e+00      1.848277
        131072         1  3.149670e-16  5.021000e-03  2.510500e-03   4437.809201
        262144         1  3.218597e-16  1.161721e+02  5.808605e+01      0.406173
        524288         1  3.281373e-16  2.545400e-02  1.272700e-02   3913.519290
```

然后是 cffti 函数：

```
struct CFFTI_Arg {
    int n;
    double *w;
};

void* cffti_functor(void* args, int i) {
    CFFTI_Arg* arg = static_cast<CFFTI_Arg*>(args);

    if (i < arg->n / 2) {
        double aw = 2.0 * M_PI / static_cast<double>(arg->n);
        double angle = aw * static_cast<double>(i);

        arg->w[i * 2 + 0] = cos(angle);
        arg->w[i * 2 + 1] = sin(angle);
    }

    return nullptr;
}

void cffti(int n, double w[]) {
    int n2 = n / 2;

    CFFTI_Arg arg;
    arg.n = n;
    arg.w = w;

    parallel_for(0, n2, 1, cffti_functor, &arg, 4);
}
```

首先定义结构体用于传递参数给并行函数。它包含两个成员变量：n 表示数组 w 的大小，w 是一个指向双精度浮点数数组的指针。

其次定义并行化循环中的操作函数。它接收两个参数：args 是一个 void* 指针，通过 static_cast 转换为 CFFTI_Arg* 类型，以获取传递给并行函数的结构体参数；i 是循环的当前索引。在函数中，它使用索引 i 计算角度，然后使用余弦和正弦函数计算数组 w 中的值。

最后是你主要调用的函数。它首先计算了 n 的一半作为 n2，然后创建了一个 CFFTI_Arg 结构体，设置了 n 和 w 的值。接着，它使用 parallel_for 函数调用，并传递了起始索引 0、结束索引 n2、增量 1、并行操作函数 cffti_functor、结构体参数 &arg，以及把 num_threads 设为 4。

```
Accuracy check:

  FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

          N      NITS    Error          Time           Time/Call      MFLOPS

          2     10000   7.859082e-17   7.850000e-04   3.925000e-08    254.777070
          4     10000   1.209837e-16   1.228000e-03   6.140000e-08    651.465798
          8     10000   6.820795e-17   1.653000e-03   8.265000e-08   1451.905626
         16     10000   1.438671e-16   3.047000e-03   1.523500e-07   2100.426649
         32      1000   1.331210e-16   5.130000e-04   2.565000e-07   3118.908382
         64      1000   1.776545e-16   1.087000e-03   5.435000e-07   3532.658694
        128      1000   1.929043e-16   3.286000e-03   1.643000e-06   2726.719416
        256      1000   2.092319e-16   6.212000e-03   3.106000e-06   3296.844816
        512       100   1.927488e-16   1.121000e-03   5.605000e-06   4110.615522
       1024       100   2.308607e-16   2.441000e-03   1.220500e-05   4195.002048
       2048       100   2.447624e-16   5.940000e-03   2.970000e-05   3792.592593
       4096       100   2.479782e-16   1.377400e-02   6.887000e-05   3568.462320
       8192        10   2.578088e-16   2.473000e-03   1.236500e-04   4306.348564
      16384        10   2.733986e-16   5.501000e-03   2.750500e-04   4169.714597
      32768        10   2.923012e-16   1.102000e-02   5.510000e-04   4460.254083
      65536        10   2.829927e-16   2.631900e-02   1.315950e-03   3984.102739
     131072         1   3.149670e-16   5.835000e-03   2.917500e-03   3818.721508
     262144         1   3.218597e-16   1.188500e-02   5.942500e-03   3970.207825
     524288         1   3.281373e-16   2.867300e-02   1.433650e-02   3474.164545
    1048576         1   3.285898e-16   7.405200e-02   3.702600e-02   2831.999136
```

可见 Error 值与原代码所得结果完全一致，且性能指标更优。可知之前的运行结果异常原因为 ccopy 函数改造不当。

最后是 step 函数：

```c
struct step_args {
    int n;
    int mj;
    double *a;
    double *b;
    double *c;
    double *d;
    double *w;
    double sgn;
    int lj;
};

void *step_parallel(void *args) {
    struct step_args *sargs = (struct step_args *)args;
    int n = sargs->n;
    int mj = sargs->mj;
    double *a = sargs->a;
    double *b = sargs->b;
    double *c = sargs->c;
    double *d = sargs->d;
    double *w = sargs->w;
    double sgn = sargs->sgn;

    int mj2 = 2 * mj;
    int lj = n / mj2;
    double wjw[2];

    for (int j = 0; j < lj; j++) {
        int jw = j * mj;

        wjw[0] = w[jw * 2 + 0];
        wjw[1] = (sgn < 0.0) ? -w[jw * 2 + 1] : w[jw * 2 + 1];

        for (int k = 0; k < mj; k++) {
            int ja = jw + k;
            int jb = ja;
            int jc = j * mj2 + k;
            int jd = jc;

            c[(jc)*2 + 0] = a[(ja)*2 + 0] + b[(jb)*2 + 0];
            c[(jc)*2 + 1] = a[(ja)*2 + 1] + b[(jb)*2 + 1];

            double ambr = a[(ja)*2 + 0] - b[(jb)*2 + 0];
            double ambu = a[(ja)*2 + 1] - b[(jb)*2 + 1];

            d[(jd)*2 + 0] = wjw[0] * ambr - wjw[1] * ambu;
            d[(jd)*2 + 1] = wjw[1] * ambr + wjw[0] * ambu;
        }
    }

    return NULL;
}

void step(int n, int mj, double a[], double b[], double c[], double d[], double w[], double sgn) {
    int mj2 = 2 * mj;
    int lj = n / mj2;

    struct step_args step_args = {n, mj, a, b, c, d, w, sgn, lj};

    // 使用 parallel_for 并行化循环
    parallel_for(0, lj, 1, (void *(*)(void *, int))step_parallel, (void *)&step_args, 2);
}
```

首先是 step_args 结构体，包含了执行计算步骤所需的所有参数。它存储了数组的大小 n，每个子问题的大小 mj，以及输入输出数组的指针，权重数组的指针，一个标志 sgn，以及 lj 的计算结果。

其次是 step_parallel 函数，它接受 step_args 结构体作为参数，并在指定的索引范围内执行计算。在外层循环中，它使用权重数组的实部和虚部来计算 wjw，并用它们更新输出数组。在内层循环中，它执行线性组合，将计算结果存储在输出数组中。

最后是 step 函数，用于设置并行执行的参数，并调用 parallel_for 函数来实现并行化。该函数创建了 step_args 结构体，并在并行循环中调用 step_parallel 函数。

实际上做完 step 的改造后结果如下：

```
Accuracy check:

  FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

             N      NITS    Error         Time          Time/Call     MFLOPS

             2     10000   7.859082e-17  1.216284e+00  6.081420e-05    0.164435
             4     10000   1.209837e-16  8.307015e+00  4.153507e-04    0.096304
             8     10000   6.820795e-17  3.845491e+00  1.922745e-04    0.624108
            16     10000   1.438671e-16  1.130102e+01  5.650509e-04    0.566321
            32      1000   1.331210e-16  6.335340e-01  3.167670e-04    2.525516
            64      1000   1.776545e-16  1.372985e+00  6.864925e-04    2.796826
           128      1000   1.929043e-16  9.487540e-01  4.743770e-04    9.443965
           256      1000   2.092319e-16  1.996388e+00  9.981940e-04   10.258527
           512       100   1.927488e-16  1.959470e-01  9.797350e-04   23.516563
          1024       100   2.308607e-16  1.985655e+00  9.928275e-03    5.156989
          2048       100   2.447624e-16  2.433352e+00  1.216676e-02    9.258011
          4096       100   2.479782e-16  2.129317e+01  1.064658e-01    2.308346
          8192        10   2.578088e-16  2.380884e+00  1.190442e-01    4.472960
         16384        10   2.733986e-16  2.075559e+01  1.037780e+00    1.105129
         32768        10   2.923012e-16  2.951193e+01  1.475597e+00    1.665496
         65536        10   2.829927e-16  2.373913e+02  1.186956e+01    0.441708
        131072         1   3.149670e-16  4.292861e+01  2.146431e+01    0.519053
```

但 Error 值结果正确。

故舍弃 ccopy 的改造，只选用 cffti 和 step 的改造尝试：

```
Accuracy check:

  FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

             N      NITS    Error         Time          Time/Call     MFLOPS

             2     10000   9.756093e-01  3.067068e+00  1.533534e-04    0.065209
             4     10000   9.866911e-01  6.076611e+00  3.038306e-04    0.131652
             8     10000   6.788403e-01  9.358847e+00  4.679423e-04    0.256442
            16     10000   8.134498e-01  1.241499e+01  6.207493e-04    0.515506
            32      1000   8.916772e-01  1.520252e+00  7.601260e-04    1.052457
            64      1000   9.363559e-01  1.911749e+00  9.558745e-04    2.008632
           128      1000   8.739357e-01  2.292807e+00  1.146403e-03    3.907874
           256      1000   9.572396e-01  2.793385e+00  1.396693e-03    7.331607
           512       100   8.913816e-01  4.440190e-01  2.220095e-03   10.377934
          1024       100   9.249854e-01  1.339975e+00  6.699875e-03    7.641934
          2048       100   9.218954e-01  3.933905e+00  1.966953e-02    5.726625
          4096       100   9.145336e-01  1.204018e+01  6.020089e-02    4.082332
          8192        10   9.010387e-01  3.835987e+00  1.917994e-01    2.776235
         16384        10   9.017614e-01  1.136353e+01  5.681764e-01    2.018528
         32768        10   9.149374e-01  3.724168e+01  1.862084e+00    1.319812
         65536        10   9.106033e-01  1.348798e+02  6.743988e+00    0.777415
        131072         1   9.126034e-01  5.176052e+01  2.588026e+01    0.430487
```

可见无论是 Error 值还是 MFLOPS 都仍然表现极差。

暂不知晓原因。

为方便实验，故第三小问性能分析的代码对象为仅改造 cffti 后的代码。

（2）将 heated_plate_openmp 应用改造成基于 MPI 的进程并行应用。Bonus:使用 MPI_Pack/MPI_Unpack，或 MPI_Type_create_struct 实现数据重组后的消息传递。

```
diff = 0.0;
double buf1[N], buf2[N];
MPI_Request send_request, recv_request;
MPI_Status send_status, recv_status;
int pos1 = 0, pos2 = 0;

// Pack and send data to the right neighbor
if (my_id != size - 1) {
    MPI_Pack(&w[(my_id + 1) * M / size - 1][1], N - 2, MPI_DOUBLE, buf1, 8 * (N - 2), &pos1, MPI_COMM_WORLD);
    MPI_Isend(buf1, N - 2, MPI_DOUBLE, my_id + 1, 0, MPI_COMM_WORLD, &send_request);
}

// Pack and receive data from the left neighbor
if (my_id != 0) {
    MPI_Irecv(buf2, N - 2, MPI_DOUBLE, my_id - 1, 0, MPI_COMM_WORLD, &recv_request);
}

// Wait for the receive to complete
if (my_id != 0) {
    MPI_Wait(&recv_request, &recv_status);
    MPI_Unpack(buf2, 8 * (N - 2), &pos2, &w[my_id * M / size - 1][1], N - 2, MPI_DOUBLE, MPI_COMM_WORLD);
}

pos1 = pos2 = 0;

// Pack and send data to the left neighbor
if (my_id != 0) {
    MPI_Pack(&w[my_id * M / size][1], N - 2, MPI_DOUBLE, buf1, 8 * (N - 2), &pos1, MPI_COMM_WORLD);
    MPI_Isend(buf1, N - 2, MPI_DOUBLE, my_id - 1, 0, MPI_COMM_WORLD, &send_request);
}

// Pack and receive data from the right neighbor
if (my_id != size - 1) {
    MPI_Irecv(buf2, N - 2, MPI_DOUBLE, my_id + 1, 0, MPI_COMM_WORLD, &recv_request);
}

// Wait for the receive to complete
if (my_id != size - 1) {
    MPI_Wait(&recv_request, &recv_status);
    MPI_Unpack(buf2, 8 * (N - 2), &pos2, &w[(my_id + 1) * M / size][1], N - 2, MPI_DOUBLE, MPI_COMM_WORLD);
}
```

double buf1[N], buf2[N];: 分别用于存储打包的数据的缓冲区。

MPI_Request send_request, recv_request;: 用于非阻塞发送和接收的 MPI 请求。

MPI_Status send_status, recv_status;: 用于存储发送和接收操作的状态信息。

int pos1 = 0, pos2 = 0;: 用于记录打包和解包位置的指针。

打包和发送数据给右侧邻居:

MPI_Pack: 将 w 数组中右侧邻居的一行数据打包成连续的字节流。

MPI_Isend: 使用非阻塞发送将打包的数据发送给右侧邻居。

接收左侧邻居的数据:

MPI_Irecv: 使用非阻塞接收从左侧邻居接收数据。这样可以允许计算和通信的重叠，提高性能。

等待左侧邻居的接收操作完成:

MPI_Wait: 阻塞当前进程，直到左侧邻居的接收操作完成。这是为了确保接收到正确的数据后再进行解包操作。

MPI_Unpack: 将接收到的字节流解包成数据，并存储到 w 数组中对应的位置。

重置打包和解包位置指针:

pos1 = pos2 = 0;: 重置打包和解包位置指针，准备进行下一轮的打包和解包操作。

MPI_Unpack: 将接收到的字节流解包成数据，并存储到 w 数组中对应的位置。

这个过程的核心思想是利用非阻塞通信允许计算和通信的重叠，以减小通信开销，提高程序性能。这段代码的设计假定左侧邻居发送，右侧邻居接收，然后右侧邻居发送，左侧邻居接收。

（3）性能分析任务：对任务 1 实现的并行化 fft 应用在不同规模下的性能进行分析，即分析：

1）不同规模下的并行化 fft 应用的执行时间对比；

2）不同规模下的并行化 fft 应用的内存消耗对比。

内存消耗采用“valgrind －tool=massif --time-unit=B ./your_exe”工具采集

命令 valgrind 命令中增加--stacks=yes 参数采集程序运行栈内内存消耗。

Valgrind －tool=massif 输出日志（massif.out.pid）

具体结果见后面实验结果部分

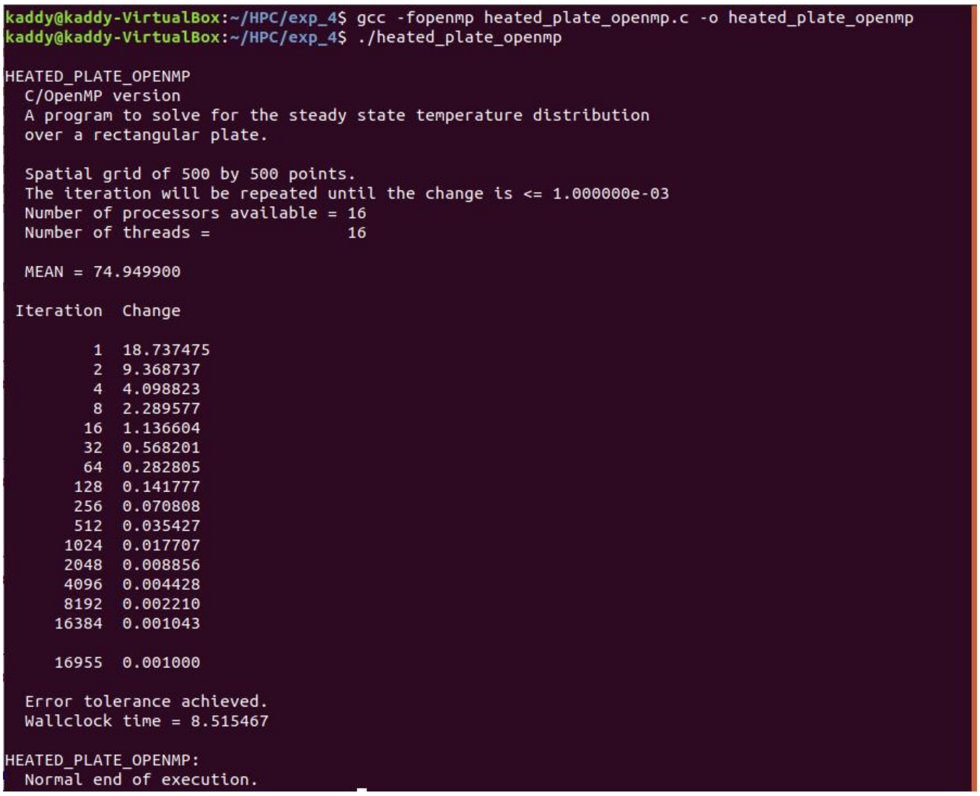
3. 实验结果

（1）通过实验 3 构造的基于 Pthreads 的 parallel_for 函数替换 fft_serial 应用中的某些计算量较大的“for 循环”,实现 for 循环分解、分配和线程并行执行。

由于本题实验内容丰富，故将实验结果融合进实验过程中讲解，详情见回实验过程及核心代码部分。此处不冗余粘贴。

（2）将 heated_plate_openmp 应用改造成基于 MPI 的进程并行应用。Bonus:使用 MPI_Pack/MPI_Unpack，或 MPI_Type_create_struct 实现数据重组后的消息传递。

运行原代码，得到结果如下：



改造为基于 MPI：

```
kaddy@kaddy-VirtualBox:~/HPC/exp_4$ mpic++ heated_plate_openmp.cpp -o heated_plate_openmp
kaddy@kaddy-VirtualBox:~/HPC/exp_4$ mpirun -n 4 ./heated_plate_openmp

HEATED_PLATE_OPENMP
  C/OpenMP version
  A program to solve for the steady state temperature distribution
  over a rectangular plate.

  Spatial grid of 500 by 500 points.
  The iteration will be repeated until the change is <= 1.000000e-03
  Number of processes =              4

  MEAN = 74.949900

 Iteration  Change

         1  18.737475
         2  9.368737
         4  4.098823
         8  2.289577
        16  1.136604
        32  0.568201
        64  0.282805
       128  0.141777
       256  0.070808
       512  0.035427
      1024  0.017707
      2048  0.008856
      4096  0.004428
      8192  0.002210
     16384  0.001043
     16955  0.001000

  Error tolerance achieved.
  Wallclock time = 12.171350
```

可见结果一致，故改造正确，但并行效率不高。

总计多进程耗时如下：

单进程	双进程	四进程	八进程
25.077148s	15.158294s	12.171350s	26.809175s

猜测是通信开销太大了，所以四进程耗时比八进程短。

（3）性能分析任务：对任务 1 实现的并行化 fft 应用在不同规模下的性能进行分析，即分析：

1）不同规模下的并行化 fft 应用的执行时间对比；

单线程：

```
Accuracy check:

  FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

          N      NITS    Error          Time           Time/Call      MFLOPS

          2     10000  7.859082e-17  5.840000e-04  2.920000e-08    342.465753
          4     10000  1.209837e-16  9.010000e-04  4.505000e-08    887.902331
          8     10000  6.820795e-17  1.262000e-03  6.310000e-08   1901.743265
         16     10000  1.438671e-16  2.148000e-03  1.074000e-07   2979.515829
         32      1000  1.331210e-16  3.840000e-04  1.920000e-07   4166.666667
         64      1000  1.776545e-16  8.290000e-04  4.145000e-07   4632.086852
        128      1000  1.929043e-16  1.669000e-03  8.345000e-07   5368.484122
        256      1000  2.092319e-16  4.041000e-03  2.020500e-06   5068.052462
        512       100  1.927488e-16  8.010000e-04  4.005000e-06   5752.808989
       1024       100  2.308607e-16  1.964000e-03  9.820000e-06   5213.849287
       2048       100  2.447624e-16  3.965000e-03  1.982500e-05   5681.715006
       4096       100  2.479782e-16  9.029000e-03  4.514500e-05   5443.792225
       8192        10  2.578088e-16  1.848000e-03  9.240000e-05   5762.770563
      16384        10  2.733986e-16  4.083000e-03  2.041500e-04   5617.830027
      32768        10  2.923012e-16  9.274000e-03  4.637000e-04   5299.978434
      65536        10  2.829927e-16  1.892900e-02  9.464500e-04   5539.521369
     131072         1  3.149670e-16  3.972000e-03  1.986000e-03   5609.828802
     262144         1  3.218597e-16  9.028000e-03  4.514000e-03   5226.619406
     524288         1  3.281373e-16  2.302700e-02  1.151350e-02   4325.996439
    1048576         1  3.285898e-16  7.212800e-02  3.606400e-02   2907.542147
    2097152         1  3.508387e-16  1.513960e-01  7.569800e-02   2908.940263
```

N	**32**	**256**	**4096**	**65536**	**2097152**
Time（s）	**0.000384**	**0.004041**	**0.009029**	**0.018929**	**0.151396**

双线程：

```
Accuracy check:

  FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

          N      NITS    Error          Time           Time/Call      MFLOPS

          2     10000  7.859082e-17  8.380000e-04  4.190000e-08    238.663484
          4     10000  1.209837e-16  9.500000e-04  4.750000e-08    842.105263
          8     10000  6.820795e-17  1.272000e-03  6.360000e-08   1886.792453
         16     10000  1.438671e-16  2.168000e-03  1.084000e-07   2952.029520
         32      1000  1.331210e-16  3.820000e-04  1.910000e-07   4188.481675
         64      1000  1.776545e-16  8.210000e-04  4.105000e-07   4677.222899
        128      1000  1.929043e-16  1.657000e-03  8.285000e-07   5407.362704
        256      1000  2.092319e-16  4.279000e-03  2.139500e-06   4786.164992
        512       100  1.927488e-16  1.032000e-03  5.160000e-06   4465.116279
       1024       100  2.308607e-16  1.922000e-03  9.610000e-06   5327.783559
       2048       100  2.447624e-16  4.181000e-03  2.090500e-05   5388.184645
       4096       100  2.479782e-16  1.032000e-02  5.160000e-05   4762.790698
       8192        10  2.578088e-16  1.795000e-03  8.975000e-05   5932.924791
      16384        10  2.733986e-16  4.149000e-03  2.074500e-04   5528.464690
      32768        10  2.923012e-16  8.442000e-03  4.221000e-04   5822.316986
      65536        10  2.829927e-16  1.986400e-02  9.932000e-04   5278.775675
     131072         1  3.149670e-16  4.023000e-03  2.011500e-03   5538.712404
     262144         1  3.218597e-16  9.234000e-03  4.617000e-03   5110.019493
     524288         1  3.281373e-16  2.298200e-02  1.149100e-02   4334.466974
    1048576         1  3.285898e-16  6.918300e-02  3.459150e-02   3031.311160
    2097152         1  3.508387e-16  1.521340e-01  7.606700e-02   2894.829032
```

N	**32**	**256**	**4096**	**65536**	**2097152**
Time（s）	**0.000382**	**0.004279**	**0.010320**	**0.019864**	**0.152134**

四线程：

```
Accuracy check:

  FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

          N      NITS    Error          Time           Time/Call      MFLOPS

          2     10000  7.859082e-17  7.730000e-04  3.865000e-08    258.732212
          4     10000  1.209837e-16  9.820000e-04  4.910000e-08    814.663951
          8     10000  6.820795e-17  1.332000e-03  6.660000e-08   1801.801802
         16     10000  1.438671e-16  2.239000e-03  1.119500e-07   2858.418937
         32      1000  1.331210e-16  3.610000e-04  1.805000e-07   4432.132964
         64      1000  1.776545e-16  8.250000e-04  4.125000e-07   4654.545455
        128      1000  1.929043e-16  1.726000e-03  8.630000e-07   5191.193511
        256      1000  2.092319e-16  4.627000e-03  2.313500e-06   4426.194078
        512       100  1.927488e-16  8.230000e-04  4.115000e-06   5599.027947
       1024       100  2.308607e-16  2.101000e-03  1.050500e-05   4873.869586
       2048       100  2.447624e-16  4.632000e-03  2.316000e-05   4863.557858
       4096       100  2.479782e-16  1.149900e-02  5.749500e-05   4274.458649
       8192        10  2.578088e-16  1.847000e-03  9.235000e-05   5765.890633
      16384        10  2.733986e-16  4.406000e-03  2.203000e-04   5205.991829
      32768        10  2.923012e-16  8.699000e-03  4.349500e-04   5650.304633
      65536        10  2.829927e-16  1.931900e-02  9.659500e-04   5427.692945
     131072         1  3.149670e-16  3.992000e-03  1.996000e-03   5581.723447
     262144         1  3.218597e-16  8.866000e-03  4.433000e-03   5322.120460
     524288         1  3.281373e-16  2.247200e-02  1.123600e-02   4432.837309
    1048576         1  3.285898e-16  7.436700e-02  3.718350e-02   2820.003496
    2097152         1  3.508387e-16  1.618640e-01  8.093200e-02   2720.814511
```

N	**32**	**256**	**4096**	**65536**	**2097152**
Time（s）	**0.000361**	**0.004627**	**0.011499**	**0.019319**	**0.161864**

八线程：

```
Accuracy check:

  FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

          N      NITS    Error          Time           Time/Call      MFLOPS

          2     10000  7.859082e-17  7.850000e-04  3.925000e-08    254.777070
          4     10000  1.209837e-16  9.740000e-04  4.870000e-08    821.355236
          8     10000  6.820795e-17  1.593000e-03  7.965000e-08   1506.591337
         16     10000  1.438671e-16  2.261000e-03  1.130500e-07   2830.605927
         32      1000  1.331210e-16  3.920000e-04  1.960000e-07   4081.632653
         64      1000  1.776545e-16  8.950000e-04  4.475000e-07   4290.502793
        128      1000  1.929043e-16  1.669000e-03  8.345000e-07   5368.484122
        256      1000  2.092319e-16  4.683000e-03  2.341500e-06   4373.265001
        512       100  1.927488e-16  8.200000e-04  4.100000e-06   5619.512195
       1024       100  2.308607e-16  2.094000e-03  1.047000e-05   4890.162369
       2048       100  2.447624e-16  4.036000e-03  2.018000e-05   5581.764123
       4096       100  2.479782e-16  1.068900e-02  5.344500e-05   4598.372158
       8192        10  2.578088e-16  2.256000e-03  1.128000e-04   4720.567376
      16384        10  2.733986e-16  5.598000e-03  2.799000e-04   4097.463380
      32768        10  2.923012e-16  1.375300e-02  6.876500e-04   3573.911147
      65536        10  2.829927e-16  2.155300e-02  1.077650e-03   4865.104626
     131072         1  3.149670e-16  4.877000e-03  2.438500e-03   4568.841501
     262144         1  3.218597e-16  8.576000e-03  4.288000e-03   5502.089552
     524288         1  3.281373e-16  2.533100e-02  1.266550e-02   3932.522206
    1048576         1  3.285898e-16  7.071800e-02  3.535900e-02   2965.513731
    2097152         1  3.508387e-16  1.603260e-01  8.016300e-02   2746.915160
```

N	**32**	**256**	**4096**	**65536**	**2097152**
Time（s）	**0.000392**	**0.004683**	**0.010689**	**0.021553**	**0.160326**

未改造的原代码：

```
 Accuracy check:

   FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

             N       NITS    Error          Time           Time/Call       MFLOPS

             2      10000   7.859082e-17   1.011000e-03   5.055000e-08     197.823937
             4      10000   1.209837e-16   1.983000e-03   9.915000e-08     403.429148
             8      10000   6.820795e-17   4.082000e-03   2.041000e-07     587.947085
            16      10000   1.438671e-16   8.923000e-03   4.461500e-07     717.247562
            32       1000   1.331210e-16   1.773000e-03   8.865000e-07     902.425268
            64       1000   1.776545e-16   4.253000e-03   2.126500e-06     902.892076
           128       1000   1.929043e-16   9.430000e-03   4.715000e-06     950.159067
           256       1000   2.092319e-16   2.164400e-02   1.082200e-05     946.220662
           512        100   1.927488e-16   4.548000e-03   2.274000e-05    1013.192612
          1024        100   2.308607e-16   1.068000e-02   5.340000e-05     958.801498
          2048        100   2.447624e-16   2.312700e-02   1.156350e-04     974.099537
          4096        100   2.479782e-16   4.965700e-02   2.482850e-04     989.830235
          8192         10   2.578088e-16   1.149700e-02   5.748500e-04     926.293816
         16384         10   2.733986e-16   2.543300e-02   1.271650e-03     901.883380
         32768         10   2.923012e-16   5.022700e-02   2.511350e-03     978.597169
         65536         10   2.829927e-16   1.106890e-01   5.534450e-03     947.317258
        131072          1   3.149670e-16   2.250600e-02   1.125300e-02     990.057762
        262144          1   3.218597e-16   4.922500e-02   2.461250e-02     958.576333
        524288          1   3.281373e-16   1.009410e-01   5.047050e-02     986.860840
       1048576          1   3.285898e-16   2.200340e-01   1.100170e-01     953.103611
       2097152          1   3.508387e-16   4.471940e-01   2.235970e-01     984.811782
```

N	**32**	**256**	**4096**	**65536**	**2097152**
Time（s）	**0.001773**	**0.021644**	**0.049657**	**0.110689**	**0.447194**

可以看到，改造后的代码虽然在改变线程数时执行时间无较大变化，但无一例外地优于原代码。所以改造是成功的。

2）不同规模下的并行化 fft 应用的内存消耗对比。

单线程：

```
 Accuracy check:
 num_thread:1

   FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

             N      NITS    Error          Time           Time/Call     MFLOPS

             2     10000  7.859082e-17  2.970000e-02  1.485000e-06      6.734007
             4     10000  1.209837e-16  3.924900e-02  1.962450e-06     20.382685
             8     10000  6.820795e-17  5.194500e-02  2.597250e-06     46.202714
            16     10000  1.438671e-16  7.506700e-02  3.753350e-06     85.257170
            32      1000  1.331210e-16  1.164800e-02  5.824000e-06    137.362637
            64      1000  1.776545e-16  2.095900e-02  1.047950e-05    183.214848
           128      1000  1.929043e-16  3.981200e-02  1.990600e-05    225.057772
           256      1000  2.092319e-16  8.248200e-02  4.124100e-05    248.296598
           512       100  1.927488e-16  1.752500e-02  8.762500e-05    262.938659
          1024       100  2.308607e-16  3.791400e-02  1.895700e-04    270.084929
          2048       100  2.447624e-16  8.074900e-02  4.037450e-04    278.987975
          4096       100  2.479782e-16  1.780580e-01  8.902900e-04    276.044884
          8192        10  2.578088e-16  3.820700e-02  1.910350e-03    278.734263
         16384        10  2.733986e-16  8.232700e-02  4.116350e-03    278.615764
         32768        10  2.923012e-16  1.711250e-01  8.556250e-03    287.228634
         65536        10  2.829927e-16  3.732200e-01  1.866100e-02    280.953861
        131072         1  3.149670e-16  7.862600e-02  3.931300e-02    283.395315
        262144         1  3.218597e-16  1.713740e-01  8.568700e-02    275.338850
        524288         1  3.281373e-16  3.566470e-01  1.783235e-01    279.309009
       1048576         1  3.285898e-16  7.408980e-01  3.704490e-01    283.055427
       2097152         1  3.508387e-16  1.550387e+00  7.751935e-01    284.059348
```

```
Command:            ./fft_serial 1
Massif arguments:   --stacks=yes
ms_print arguments: massif.out.7681
--------------------------------------------------------------------------------


    MB
112.0^                                                           ::@
     |                                       #:::::::@::::::@::::::@::::::@:::
     |                                       #:::::::@::::::@::::::@::::::@:::
     |                                       #:::::::@::::::@::::::@::::::@:::
     |                                       #:::::::@::::::@::::::@::::::@:::
     |                                       #:::::::@::::::@::::::@::::::@:::
     |                                       #:::::::@::::::@::::::@::::::@:::
     |                                       #:::::::@::::::@::::::@::::::@:::
     |                                       #:::::::@::::::@::::::@::::::@:::
     |                                       #:::::::@::::::@::::::@::::::@:::
     |                       :::::::::@::::::#:::::::@::::::@::::::@::::::@:::
     |                       :::::: ::@::::::#:::::::@::::::@::::::@::::::@:::
     |                       :::::: ::@::::::#:::::::@::::::@::::::@::::::@:::
     |                       :::::: ::@::::::#:::::::@::::::@::::::@::::::@:::
     |                       :::::: ::@::::::#:::::::@::::::@::::::@::::::@:::
     |                :@::::::::::: ::@::::::#:::::::@::::::@::::::@::::::@:::
     |                :@::::::::::: ::@::::::#:::::::@::::::@::::::@::::::@:::
     |                :@::::::::::: ::@::::::#:::::::@::::::@::::::@::::::@:::
     |            :::::@::::::::::: ::@::::::#:::::::@::::::@::::::@::::::@:::
     |         @@::::::@::::::::::: ::@::::::#:::::::@::::::@::::::@::::::@:::
   0 +----------------------------------------------------------------------->Gi
     0                                                                   9.367

Number of snapshots: 94
 Detailed snapshots: [11, 20, 36, 44 (peak), 58, 68, 78, 88]
```

双线程：

```
 Accuracy check:
 num_thread:2

   FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

             N      NITS    Error         Time          Time/Call     MFLOPS

             2     10000  7.859082e-17  3.038200e-02  1.519100e-06      6.582845
             4     10000  1.209837e-16  4.001200e-02  2.000600e-06     19.994002
             8     10000  6.820795e-17  5.170400e-02  2.585200e-06     46.418072
            16     10000  1.438671e-16  7.604900e-02  3.802450e-06     84.156268
            32      1000  1.331210e-16  1.152500e-02  5.762500e-06    138.828633
            64      1000  1.776545e-16  2.132600e-02  1.066300e-05    180.061896
           128      1000  1.929043e-16  4.010600e-02  2.005300e-05    223.407969
           256      1000  2.092319e-16  8.283600e-02  4.141800e-05    247.235501
           512       100  1.927488e-16  1.747300e-02  8.736500e-05    263.721170
          1024       100  2.308607e-16  3.847100e-02  1.923550e-04    266.174521
          2048       100  2.447624e-16  8.072900e-02  4.036450e-04    279.057092
          4096       100  2.479782e-16  1.763880e-01  8.819400e-04    278.658412
          8192        10  2.578088e-16  3.813300e-02  1.906650e-03    279.275168
         16384        10  2.733986e-16  8.152300e-02  4.076150e-03    281.363542
         32768        10  2.923012e-16  1.709100e-01  8.545500e-03    287.589960
         65536        10  2.829927e-16  3.751950e-01  1.875975e-02    279.474940
        131072         1  3.149670e-16  7.966300e-02  3.983150e-02    279.706263
        262144         1  3.218597e-16  1.732280e-01  8.661400e-02    272.391992
        524288         1  3.281373e-16  3.527710e-01  1.763855e-01    282.377860
       1048576         1  3.285898e-16  7.419520e-01  3.709760e-01    282.653325
       2097152         1  3.508387e-16  1.543649e+00  7.718245e-01    285.299262
```

```
Command:            ./fft_serial 2
Massif arguments:   --stacks=yes
ms_print arguments: massif.out.7735
-------------------------------------------------------------------------------


    MB
112.0^                                                            :
     |                                      #:::::::::::::@::::::@:::::::::@:
     |                                      #: :::::: ::::@::::::@::::::: :@:
     |                                      #: :::::: ::::@::::::@::::::: :@:
     |                                      #: :::::: ::::@::::::@::::::: :@:
     |                                      #: :::::: ::::@::::::@::::::: :@:
     |                                      #: :::::: ::::@::::::@::::::: :@:
     |                                      #: :::::: ::::@::::::@::::::: :@:
     |                                      #: :::::: ::::@::::::@::::::: :@:
     |                                      #: :::::: ::::@::::::@::::::: :@:
     |                      @:::::::::@:::::#: :::::: ::::@::::::@::::::: :@:
     |                      @:: ::::: @:::::#: :::::: ::::@::::::@::::::: :@:
     |                      @:: ::::: @:::::#: :::::: ::::@::::::@::::::: :@:
     |                      @:: ::::: @:::::#: :::::: ::::@::::::@::::::: :@:
     |                      @:: ::::: @:::::#: :::::: ::::@::::::@::::::: :@:
     |              @:::::::@:: ::::: @:::::#: :::::: ::::@::::::@::::::: :@:
     |              @:: ::: @:: ::::: @:::::#: :::::: ::::@::::::@::::::: :@:
     |              @:: ::: @:: ::::: @:::::#: :::::: ::::@::::::@::::::: :@:
     |           :::@:: ::: @:: ::::: @:::::#: :::::: ::::@::::::@::::::: :@:
     |         :::::@:: ::: @:: ::::: @:::::#: :::::: ::::@::::::@::::::: :@:
   0 +---------------------------------------------------------------------->Gi
     0                                                                  9.335

Number of snapshots: 66
 Detailed snapshots: [14, 20, 28, 34 (peak), 46, 53, 63]
```

四线程：

```
Accuracy check:
num_thread:4

  FFT ( FFT ( X(1:N) ) ) == N * X(1:N)
```

N	NITS	Error	Time	Time/Call	MFLOPS
2	10000	7.859082e-17	3.047500e-02	1.523750e-06	6.562756
4	10000	1.209837e-16	4.020000e-02	2.010000e-06	19.900498
8	10000	6.820795e-17	5.235100e-02	2.617550e-06	45.844396
16	10000	1.438671e-16	7.579900e-02	3.789950e-06	84.433832
32	1000	1.331210e-16	1.186500e-02	5.932500e-06	134.850400
64	1000	1.776545e-16	2.200200e-02	1.100100e-05	174.529588
128	1000	1.929043e-16	4.080400e-02	2.040200e-05	219.586315
256	1000	2.092319e-16	8.382500e-02	4.191250e-05	244.318521
512	100	1.927488e-16	1.731400e-02	8.657000e-05	266.143006
1024	100	2.308607e-16	3.904100e-02	1.952050e-04	262.288364
2048	100	2.447624e-16	8.112000e-02	4.056000e-04	277.712032
4096	100	2.479782e-16	1.766000e-01	8.830000e-04	278.323896
8192	10	2.578088e-16	3.852100e-02	1.926050e-03	276.462189
16384	10	2.733986e-16	8.238600e-02	4.119300e-03	278.416236
32768	10	2.923012e-16	1.736340e-01	8.681700e-03	283.078199
65536	10	2.829927e-16	3.730520e-01	1.865260e-02	281.080386
131072	1	3.149670e-16	7.844400e-02	3.922200e-02	284.052827
262144	1	3.218597e-16	1.688050e-01	8.440250e-02	279.529161
524288	1	3.281373e-16	3.508840e-01	1.754420e-01	283.896444
1048576	1	3.285898e-16	7.453330e-01	3.726665e-01	281.371146
2097152	1	3.508387e-16	1.526207e+00	7.631035e-01	288.559756

```
Command:            ./fft_serial 4
Massif arguments:   --stacks=yes
ms_print arguments: massif.out.7826
--------------------------------------------------------------------------------


    MB
112.1^                                                              ::
     |                                       ##:::::::::@::::::::::: :::::::::
     |                                       # : ::: :: @: : :: : :: :: :  :::
     |                                       # : ::: :: @: : :: : :: :: :  :::
     |                                       # : ::: :: @: : :: : :: :: :  :::
     |                                       # : ::: :: @: : :: : :: :: :  :::
     |                                       # : ::: :: @: : :: : :: :: :  :::
     |                                       # : ::: :: @: : :: : :: :: :  :::
     |                                       # : ::: :: @: : :: : :: :: :  :::
     |                                       # : ::: :: @: : :: : :: :: :  :::
     |                        :::::::::::::::# : ::: :: @: : :: : :: :: :  :::
     |                        :: :: :::: : : # : ::: :: @: : :: : :: :: :  :::
     |                        :: :: :::: : : # : ::: :: @: : :: : :: :: :  :::
     |                        :: :: :::: : : # : ::: :: @: : :: : :: :: :  :::
     |                        :: :: :::: : : # : ::: :: @: : :: : :: :: :  :::
     |                :::::::::: :: :::: : : # : ::: :: @: : :: : :: :: :  :::
     |                : ::: : :: :: :::: : : # : ::: :: @: : :: : :: :: :  :::
     |                : ::: : :: :: :::: : : # : ::: :: @: : :: : :: :: :  :::
     |           :::::: ::: : :: :: :::: : : # : ::: :: @: : :: : :: :: :  :::
     |          ::: ::: ::: : :: :: :::: : : # : ::: :: @: : :: : :: :: :  :::
   0 +------------------------------------------------------------------------>Gi
     0                                                                   9.400

Number of snapshots: 50
 Detailed snapshots: [3, 28 (peak), 35]
```

八线程：

```
Accuracy check:
num_thread:8

  FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

            N      NITS    Error         Time          Time/Call     MFLOPS

            2     10000  7.859082e-17  3.061100e-02  1.530550e-06      6.533599
            4     10000  1.209837e-16  3.971500e-02  1.985750e-06     20.143523
            8     10000  6.820795e-17  5.221300e-02  2.610650e-06     45.965564
           16     10000  1.438671e-16  7.690400e-02  3.845200e-06     83.220639
           32      1000  1.331210e-16  1.186900e-02  5.934500e-06    134.804954
           64      1000  1.776545e-16  2.128900e-02  1.064450e-05    180.374841
          128      1000  1.929043e-16  4.014700e-02  2.007350e-05    223.179814
          256      1000  2.092319e-16  8.367100e-02  4.183550e-05    244.768199
          512       100  1.927488e-16  1.749900e-02  8.749500e-05    263.329333
         1024       100  2.308607e-16  3.889200e-02  1.944600e-04    263.293222
         2048       100  2.447624e-16  8.233900e-02  4.116950e-04    273.600602
         4096       100  2.479782e-16  1.751860e-01  8.759300e-04    280.570365
         8192        10  2.578088e-16  3.842200e-02  1.921100e-03    277.174535
        16384        10  2.733986e-16  8.182500e-02  4.091250e-03    280.325084
        32768        10  2.923012e-16  1.743740e-01  8.718700e-03    281.876885
        65536        10  2.829927e-16  3.771300e-01  1.885650e-02    278.040994
       131072         1  3.149670e-16  7.837100e-02  3.918550e-02    284.317413
       262144         1  3.218597e-16  1.664780e-01  8.323900e-02    283.436370
       524288         1  3.281373e-16  3.442430e-01  1.721215e-01    289.373262
      1048576         1  3.285898e-16  7.320240e-01  3.660120e-01    286.486782
      2097152         1  3.508387e-16  1.524799e+00  7.623995e-01    288.826213
```

```
Command:            ./fft_serial 8
Massif arguments:   --stacks=yes
ms_print arguments: massif.out.8006
--------------------------------------------------------------------------------


    MB
112.1^                                                            :
     |                                       #:::::::@::::@:::::::@::::::@::::
     |                                       #:::::::@::::@:::::::@::::::@::::
     |                                       #:::::::@::::@:::::::@::::::@::::
     |                                       #:::::::@::::@:::::::@::::::@::::
     |                                       #:::::::@::::@:::::::@::::::@::::
     |                                       #:::::::@::::@:::::::@::::::@::::
     |                                       #:::::::@::::@:::::::@::::::@::::
     |                                       #:::::::@::::@:::::::@::::::@::::
     |                                       #:::::::@::::@:::::::@::::::@::::
     |                       ::@:::::::::@:::#:::::::@::::@:::::::@::::::@::::
     |                       ::@: :::::::@:::#:::::::@::::@:::::::@::::::@::::
     |                       ::@: :::::::@:::#:::::::@::::@:::::::@::::::@::::
     |                       ::@: :::::::@:::#:::::::@::::@:::::::@::::::@::::
     |                       ::@: :::::::@:::#:::::::@::::@:::::::@::::::@::::
     |               @:::::::::@: :::::::@:::#:::::::@::::@:::::::@::::::@::::
     |               @:::::::::@: :::::::@:::#:::::::@::::@:::::::@::::::@::::
     |               @:::::::::@: :::::::@:::#:::::::@::::@:::::::@::::::@::::
     |            :::@:::::::::@: :::::::@:::#:::::::@::::@:::::::@::::::@::::
     |         ::::::@:::::::::@: :::::::@:::#:::::::@::::@:::::::@::::::@::::
   0 +----------------------------------------------------------------------->Gi
     0                                                                   9.336

Number of snapshots: 79
 Detailed snapshots: [14, 24, 33, 37 (peak), 46, 52, 62, 72]
```

未改造原代码：

```
  Accuracy check:

    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

             N      NITS    Error          Time           Time/Call      MFLOPS

             2     10000   7.859082e-17   2.409100e-02   1.204550e-06       8.301855
             4     10000   1.209837e-16   3.382200e-02   1.691100e-06      23.653243
             8     10000   6.820795e-17   4.681300e-02   2.340650e-06      51.267810
            16     10000   1.438671e-16   8.086600e-02   4.043300e-06      79.143274
            32      1000   1.331210e-16   1.443600e-02   7.218000e-06     110.834026
            64      1000   1.776545e-16   3.061900e-02   1.530950e-05     125.412326
           128      1000   1.929043e-16   6.457200e-02   3.228600e-05     138.759834
           256      1000   2.092319e-16   1.427860e-01   7.139300e-05     143.431429
           512       100   1.927488e-16   3.119900e-02   1.559950e-04     147.697042
          1024       100   2.308607e-16   6.876000e-02   3.438000e-04     148.923793
          2048       100   2.447624e-16   1.480000e-01   7.400000e-04     152.216216
          4096       100   2.479782e-16   3.233700e-01   1.616850e-03     151.999258
          8192        10   2.578088e-16   6.960900e-02   3.480450e-03     152.991711
         16384        10   2.733986e-16   1.514930e-01   7.574650e-03     151.410296
         32768        10   2.923012e-16   3.226150e-01   1.613075e-02     152.354974
         65536        10   2.829927e-16   6.932290e-01   3.466145e-02     151.259685
        131072         1   3.149670e-16   1.462120e-01   7.310600e-02     152.396794
        262144         1   3.218597e-16   3.153090e-01   1.576545e-01     149.649772
        524288         1   3.281373e-16   6.428640e-01   3.214320e-01     154.954578
       1048576         1   3.285898e-16   1.365537e+00   6.827685e-01     153.577091
       2097152         1   3.508387e-16   2.839851e+00   1.419925e+00     155.079235
```

```
Command:            ./fft_serial
Massif arguments:   --stacks=yes
ms_print arguments: massif.out.8411
--------------------------------------------------------------------------------


    MB
112.0^                                          :             @:
     |                                        #::@::::::::::::@:::::::@::::::
     |                                        #::@:: :::::::::@:::::::@::::::
     |                                        #::@:: :::::::::@:::::::@::::::
     |                                        #::@:: :::::::::@:::::::@::::::
     |                                        #::@:: :::::::::@:::::::@::::::
     |                                        #::@:: :::::::::@:::::::@::::::
     |                                        #::@:: :::::::::@:::::::@::::::
     |                                        #::@:: :::::::::@:::::::@::::::
     |                                        #::@:: :::::::::@:::::::@::::::
     |                        @:@::@::::::::::#::@:: :::::::::@:::::::@::::::
     |                        @:@: @::::::::: #::@:: :::::::::@:::::::@::::::
     |                        @:@: @::::::::: #::@:: :::::::::@:::::::@::::::
     |                        @:@: @::::::::: #::@:: :::::::::@:::::::@::::::
     |                        @:@: @::::::::: #::@:: :::::::::@:::::::@::::::
     |                 :::::::@:@: @::::::::: #::@:: :::::::::@:::::::@::::::
     |                 :::::::@:@: @::::::::: #::@:: :::::::::@:::::::@::::::
     |                 :::::::@:@: @::::::::: #::@:: :::::::::@:::::::@::::::
     |               ::@::::::@:@: @::::::::: #::@:: :::::::::@:::::::@::::::
     |             ::::@::::::@:@: @::::::::: #::@:: :::::::::@:::::::@::::::
   0 +------------------------------------------------------------------------>Gi
     0                                                                    28.89

Number of snapshots: 75
 Detailed snapshots: [2, 8, 16, 24, 26, 29, 39 (peak), 43, 57, 67]
```

4. 实验感想

//可以写写过程中遇到的问题，你是怎么解决的。以及可以写你对此次实验的一些理解……

这次实验好难啊，花了好长时间，和朋友也讨论了好久。

我至今仍然不知道第一题中我对 ccopy 的改造为什么会造成 MFLOPS 交替异常，也对 step 函数的改造性能差感到遗憾