实验目的

任务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。

任务2（二选一）：

1. 将fft_serial应用改造成基于MPI的进程并行应用（为了适合MPI的消息机制，可能需要对fft_serial的代码实现做一定调整）。Bonus:使用MPI_Pack/MPI_Unpack，或MPI_Type_create_struct实现数据重组后的消息传递。
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。

任务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。

实验过程及代码

1. ***parallel_for函数***
    i. **ThreadData 结构体:**

```
typedef struct {
    int start;
    int end;
    int increment;
    int n;
    double* x;
    double* y;
    double* w;
    double sgn;
} ThreadData;
```

这个结构体用于保存将传递给每个线程的数据。它包含了一些信息，比如线程应该执行的迭代范围（`start` 和 `end`）、步长（`increment`）、数据数组的大小（`n`），以及指向数组 `x`、`y` 和 `w` 的指针。变量 `sgn` 似乎表示FFT计算的符号。

ii. **parallel_fft 函数:**

```
void* parallel_fft(void* arg) {
    ThreadData* data = (ThreadData*)arg;
    void cfft2 ( int n, double x[], double y[], double w[], double sgn );

    for (int it = data->start; it < data->end; it++) {
        data->sgn = +1.0;
        cfft2(data->n, data->x, data->y, data->w, data->sgn);
        data->sgn = -1.0;
        cfft2(data->n, data->y, data->x, data->w, data->sgn);
    }

    pthread_exit(NULL);
}
```

这是每个pthread将执行的函数。首先将输入参数 `arg` 转换为 `ThreadData` 指针。然后，它在由 `ThreadData` 结构中的 `start` 和 `end` 指定的迭代范围内进行迭代。

·3. **修改后的for循环**

```
    for (int i = 0; i < num_threads; i++) {
     // Calculate the end index for each thread
     int end = start + iterations_per_thread + (i < remaining_iterations ? 1 : 0);

     // Assign data to the thread-specific structure
     thread_data[i].start = start;
     thread_data[i].end = end;
     thread_data[i].n = n;
     thread_data[i].x = x;
     thread_data[i].y = y;
     thread_data[i].w = w;

     // Create threads
     pthread_create(&threads[i], NULL, parallel_fft, (void*)&thread_data[i]);

     // Update the start index for the next thread
     start = end;
 }
 auto start_time = std::chrono::high_resolution_clock::now();
 // Wait for all threads to finish
 for (int i = 0; i < num_threads; i++) {
     pthread_join(threads[i], NULL);
 }
 free(threads);
 free(thread_data);
 auto end_time = std::chrono::high_resolution_clock::now();
 auto elapsed_time = std::chrono::duration_cast<std::chrono::microseconds>(end_time - start_
```

这段代码首先进行线程的创建和分配，然后等待线程完成进行时间的测量。为了记录墙上时钟，利用了C++里面的 `chrono` 库函数，能够更精确记录完成的时间。

2. ***MPI应用更改***

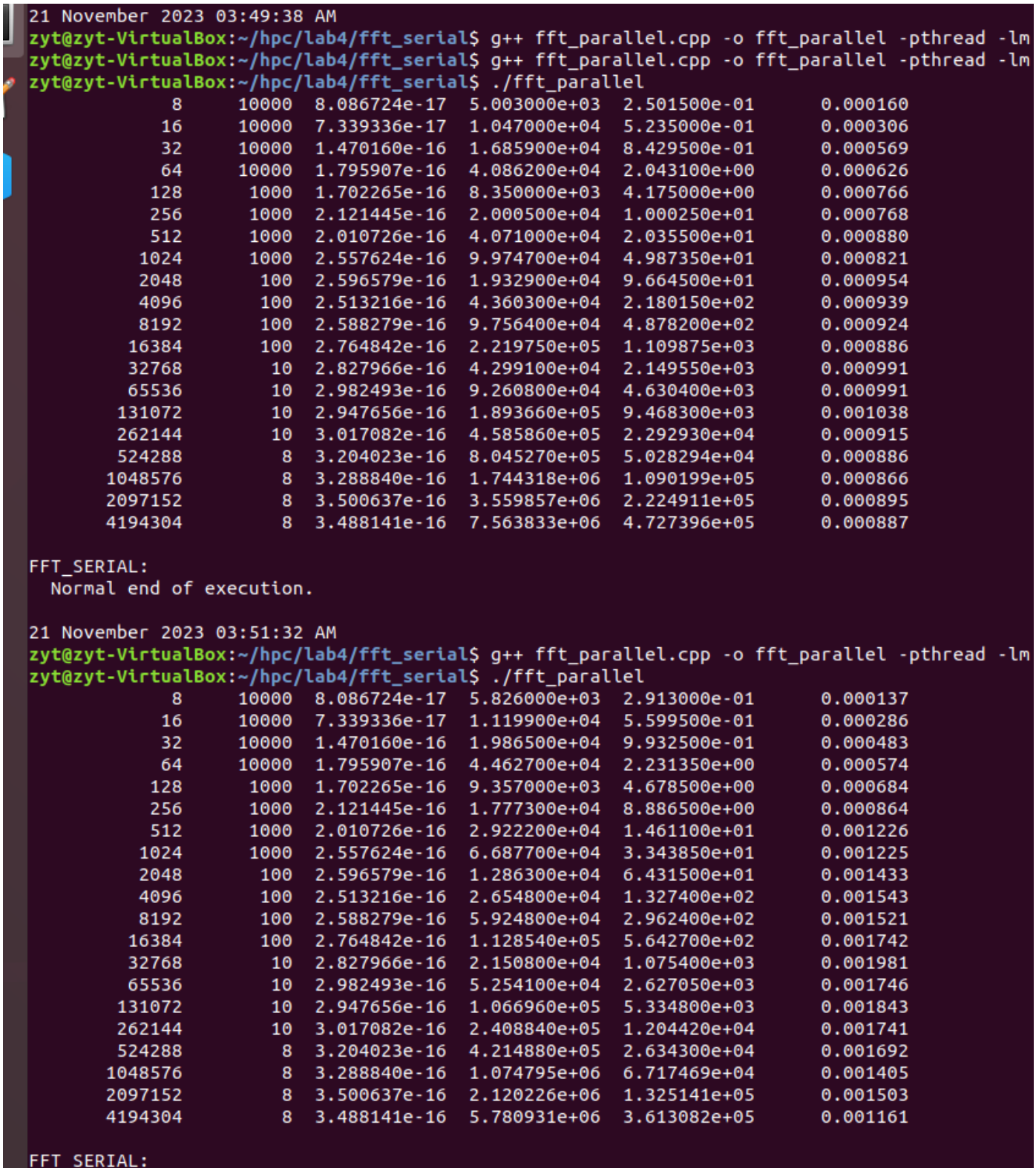
```
  while (epsilon <= diff) {
    // Scatter the data to each process
    MPI_Scatter(w, rows_per_proc * N, MPI_DOUBLE, pack_buffer, rows_per_proc * N, MPI_DOUBLE, 0,
    // Unpack the received data
    int position = 0;
  for (int i = 1; i < rows_per_proc+1; i++) {
  for (int j = 0; j < N; j++) {
    local_w[i][j] = pack_buffer[position++];
  }
}
 for (int j = 0; j < N; j++) {
  local_w[0][j]=0.0;
  local_w[ rows_per_proc+1][j]=0.0;
 }
//  for(int i=0;i<=rows_per_proc+1;i++){
//   for(int j=0;j<N;j++){
//     printf(" %f ",local_w[i][j]);
//   }

//   printf("\n");
//  }
   MPI_Sendrecv(&local_w[rows_per_proc][0],N,MPI_DOUBLE,right,1,&local_w[0][0],N,MPI_DOUBLE,left
   MPI_Sendrecv(&local_w[1][0],N,MPI_DOUBLE,left,2,&local_w[rows_per_proc+1][0],N,MPI_DOUBLE,rig



    // Each process performs its local computation
    my_diff = 0.0;
    if(rank==0){
        begin_row=2;
    }
    if(rank==size-1){
        end_row=rows_per_proc-2;
    }
      for (int i = begin_row; i < end_row; i++) {
      for (int j = 1; j < N - 1; j++) {
        local_u[i][j] = (local_w[i - 1][j] + local_w[i + 1][j] + local_w[i][j - 1] + local_w[i]
        if (my_diff < fabs(local_w[i][j] - local_u[i][j])) {
          my_diff = fabs(local_w[i][j] - local_u[i][j]);
        }
      }
    }
//      for(int i=0;i<=rows_per_proc+1;i++){
//   for(int j=0;j<N;j++){
```

```
//     printf(" %f ",local_w[i][j]);
//   }
//   printf("\n");
//  }
//  printf("**********************************************************************************\n'
    // Pack the local result for sending
    for (int i = begin_row; i < end_row; i++) {
      for (int j = 1; j < N-1; j++) {
        local_w[i][j]=local_u[i][j];
      }
    }
//      for(int i=0;i<=rows_per_proc+1;i++){
//   for(int j=0;j<N;j++){
//     printf(" %f ",local_w[i][j]);
//   }
//   printf("\n");
//  }
//  printf("**********************************************************************************\n'
     position = 0;
    for (int i = 1; i < rows_per_proc+1; i++) {
      for (int j = 0; j < N; j++) {
        pack_buffer[position++] = local_w[i][j];
      }
    }
    MPI_Barrier(MPI_COMM_WORLD);
    // Gather the local results back to the root process
    MPI_Gather(pack_buffer, rows_per_proc * N, MPI_DOUBLE, w, rows_per_proc * N, MPI_DOUBLE, 0,

    // Reduce the maximum difference across all processes
    MPI_Reduce(&my_diff, &diff, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
```

这段代码的作用就是将这个矩阵按行分块，然后每行多加上下两行，用于接收他邻近的行的数据，然后在首行和尾行进行特殊设置，他们是不需要进行迭代的。

3. ***内存和时间分析***

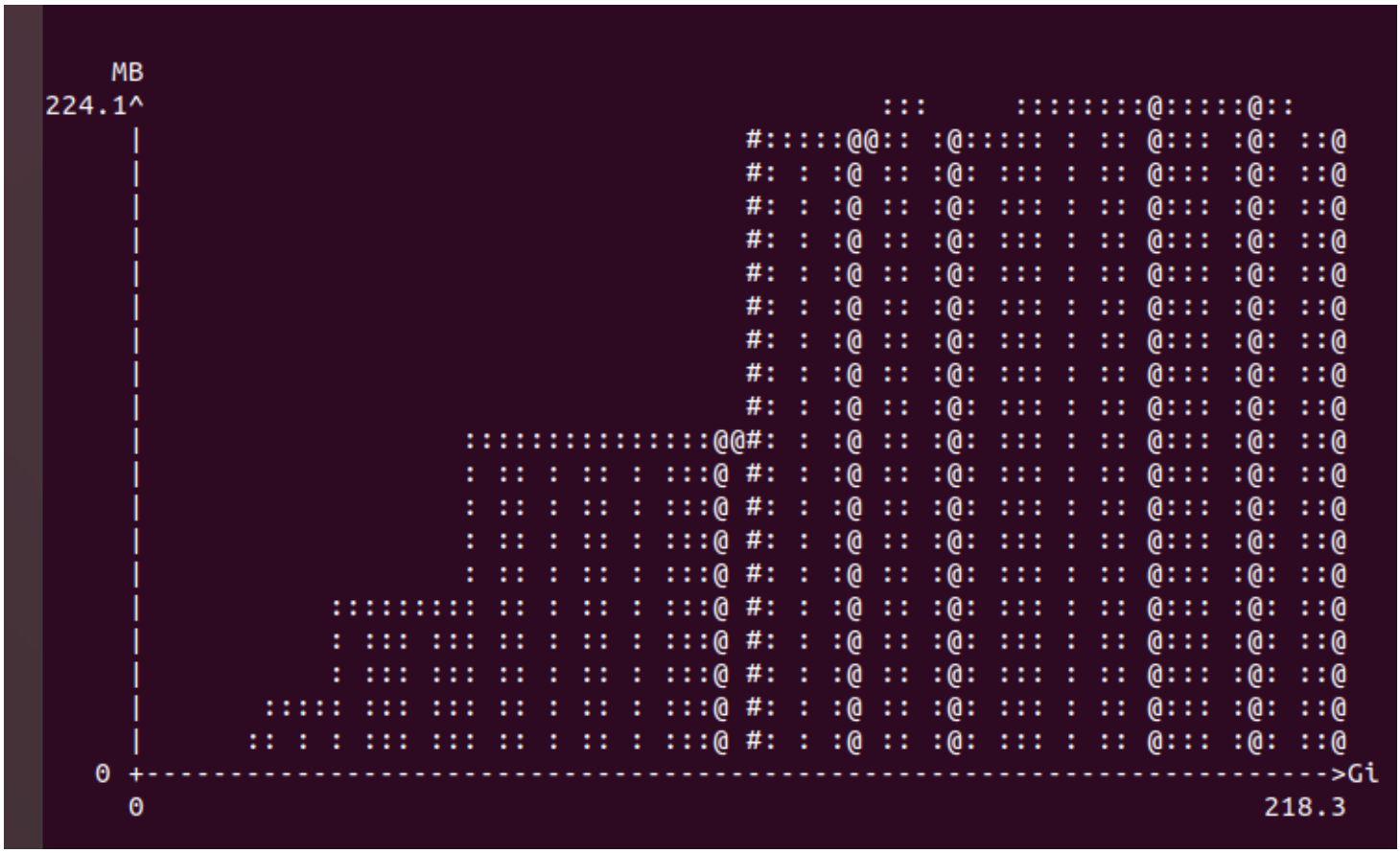
```
zyt@zyt-VirtualBox:~/hpc/lab4/fft_serial$ g++ fft_parallel.cpp -o fft_parallel -pthread
zyt@zyt-VirtualBox:~/hpc/lab4/fft_serial$ ./fft_parallel
             8      10000  8.086724e-17  5.839000e+03  2.919500e-01      0.000137
            16      10000  7.339336e-17  1.498700e+04  7.493500e-01      0.000214
            32      10000  1.470160e-16  2.236100e+04  1.118050e+00      0.000429
            64      10000  1.795907e-16  3.272500e+04  1.636250e+00      0.000782
           128       1000  1.702265e-16  8.172000e+03  4.086000e+00      0.000783
           256       1000  2.121445e-16  1.217200e+04  6.086000e+00      0.001262
           512       1000  2.010726e-16  2.477400e+04  1.238700e+01      0.001447
          1024       1000  2.557624e-16  4.094400e+04  2.047200e+01      0.002001
          2048        100  2.596579e-16  7.887000e+03  3.943500e+01      0.002337
          4096        100  2.513216e-16  2.197200e+04  1.098600e+02      0.001864
          8192        100  2.588279e-16  3.845200e+04  1.922600e+02      0.002343
         16384        100  2.764842e-16  6.635300e+04  3.317650e+02      0.002963
         32768         10  2.827966e-16  2.177400e+04  1.088700e+03      0.001956
         65536         10  2.982493e-16  3.906200e+04  1.953100e+03      0.002349
        131072         10  2.947656e-16  6.699400e+04  3.349700e+03      0.002935
        262144         10  3.017082e-16  1.875150e+05  9.375750e+03      0.002237
        524288          8  3.204023e-16  2.954520e+05  1.846575e+04      0.002413
       1048576          8  3.288840e-16  9.060630e+05  5.662894e+04      0.001666
       2097152          8  3.500637e-16  1.890512e+06  1.181570e+05      0.001686
       4194304          8  3.488141e-16  4.116246e+06  2.572654e+05      0.001630

FFT_SERIAL:
  Normal end of execution.

21 November 2023 03:53:09 AM
zyt@zyt-VirtualBox:~/hpc/lab4/fft_serial$ g++ fft_parallel.cpp -o fft_parallel -pthread
zyt@zyt-VirtualBox:~/hpc/lab4/fft_serial$ ./fft_parallel
             8      10000  8.086724e-17  1.499500e+04  7.497500e-01      0.000053
            16      10000  7.339336e-17  1.176800e+04  5.884000e-01      0.000272
            32      10000  1.470160e-16  1.991500e+04  9.957500e-01      0.000482
            64      10000  1.795907e-16  3.707300e+04  1.853650e+00      0.000691
           128       1000  1.702265e-16  9.065000e+03  4.532500e+00      0.000706
           256       1000  2.121445e-16  1.203200e+04  6.016000e+00      0.001277
           512       1000  2.010726e-16  3.016100e+04  1.508050e+01      0.001188
          1024       1000  2.557624e-16  5.214900e+04  2.607450e+01      0.001571
          2048        100  2.596579e-16  6.708000e+03  3.354000e+01      0.002748
          4096        100  2.513216e-16  2.190300e+04  1.095150e+02      0.001870
          8192        100  2.588279e-16  3.581600e+04  1.790800e+02      0.002516
         16384        100  2.764842e-16  6.818800e+04  3.409400e+02      0.002883
         32768         10  2.827966e-16  2.731000e+04  1.365500e+03      0.001560
         65536         10  2.982493e-16  4.568000e+04  2.284000e+03      0.002009
        131072         10  2.947656e-16  6.390800e+04  3.195400e+03      0.003076
        262144         10  3.017082e-16  1.932700e+05  9.663500e+03      0.002170
        524288          8  3.204023e-16  3.091760e+05  1.932350e+04      0.002306
       1048576          8  3.288840e-16  8.543520e+05  5.339700e+04      0.001767
       2097152          8  3.500637e-16  1.820259e+06  1.137662e+05      0.001751
       4194304          8  3.488141e-16  4.143692e+06  2.589808e+05      0.001620

FFT_SERIAL:
```

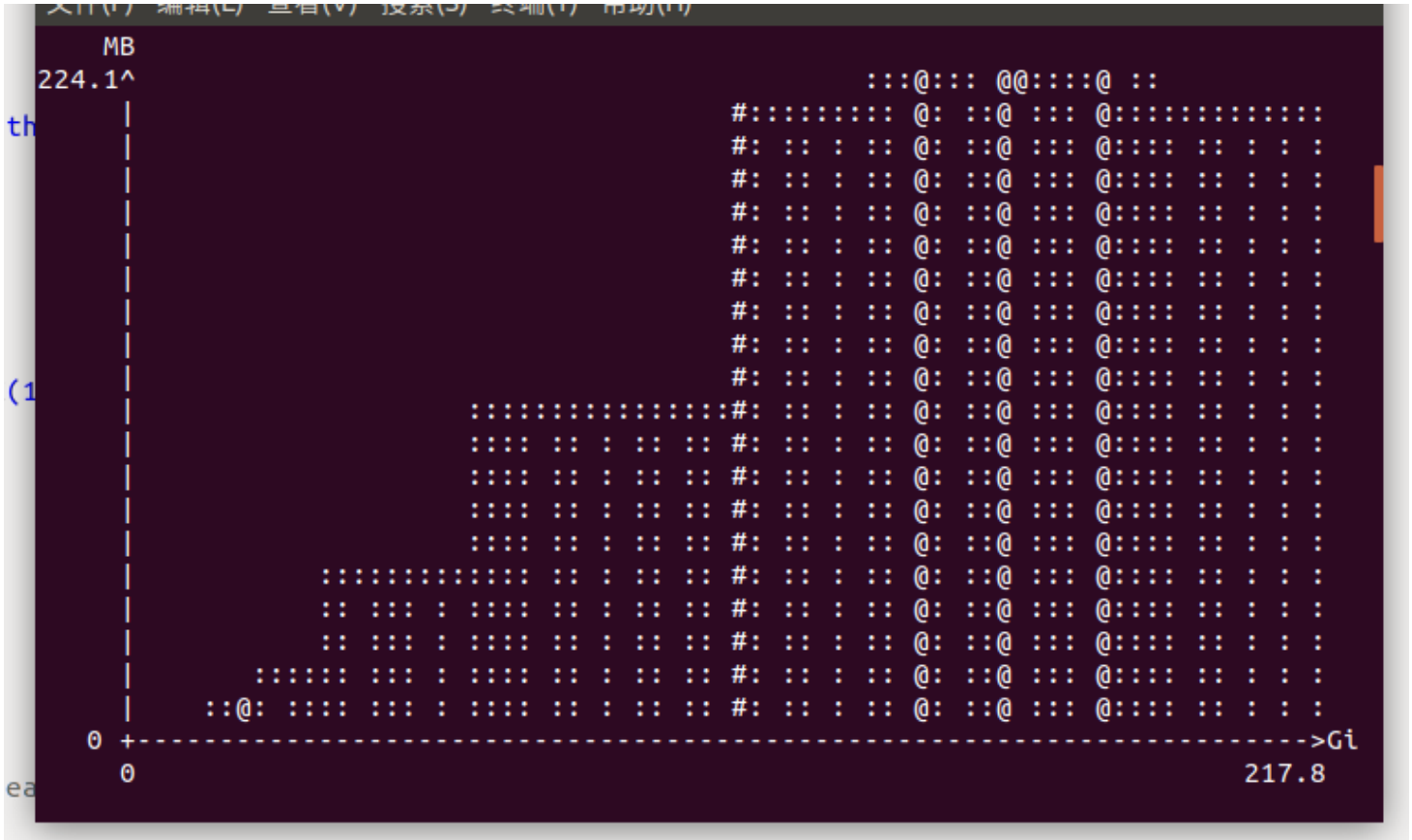


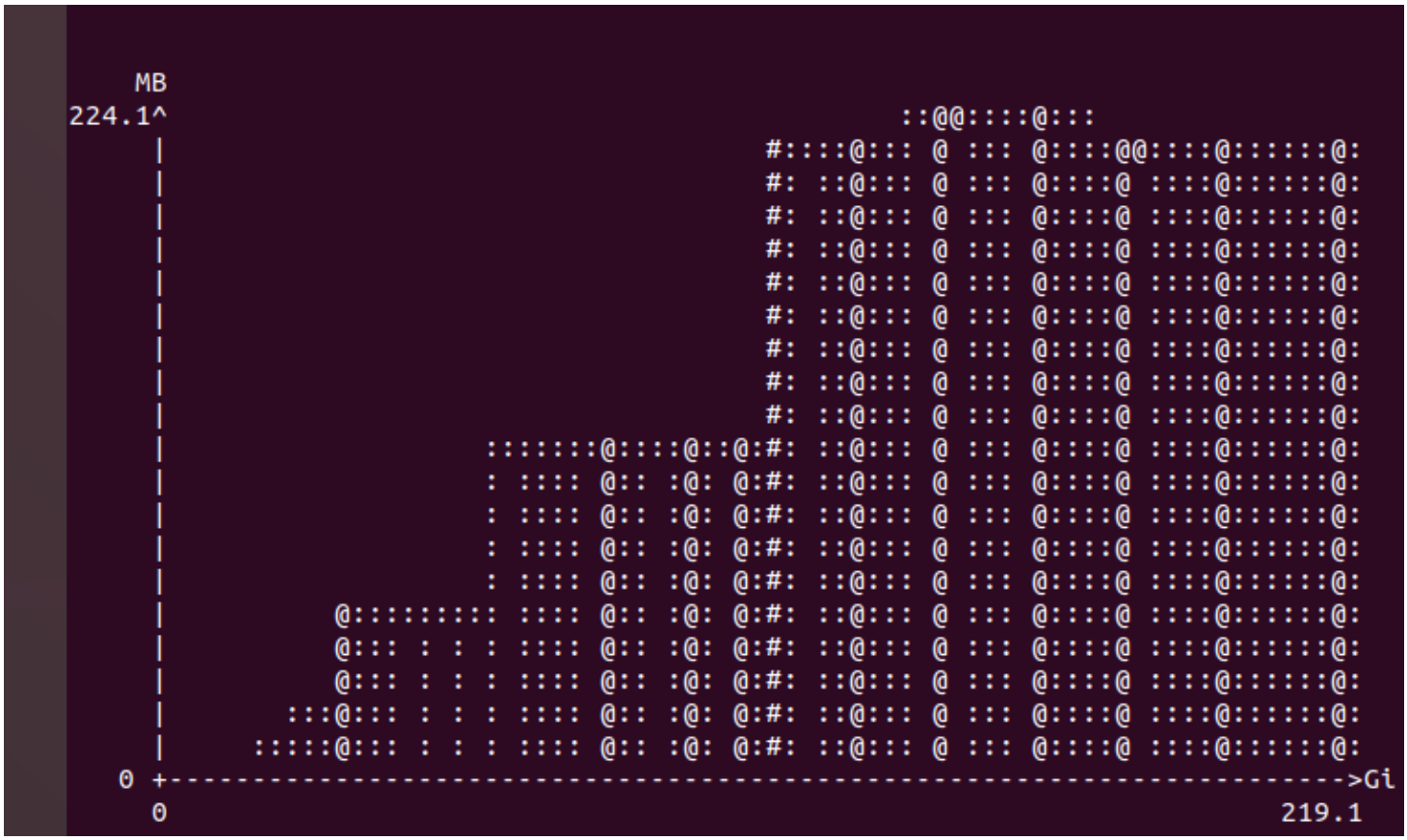
```
21 November 2023 03:49:38 AM
zyt@zyt-VirtualBox:~/hpc/lab4/fft_serial$ g++ fft_parallel.cpp -o fft_parallel -pthread -lm
zyt@zyt-VirtualBox:~/hpc/lab4/fft_serial$ g++ fft_parallel.cpp -o fft_parallel -pthread -lm
zyt@zyt-VirtualBox:~/hpc/lab4/fft_serial$ ./fft_parallel
             8      10000  8.086724e-17  5.003000e+03  2.501500e-01      0.000160
            16      10000  7.339336e-17  1.047000e+04  5.235000e-01      0.000306
            32      10000  1.470160e-16  1.685900e+04  8.429500e-01      0.000569
            64      10000  1.795907e-16  4.086200e+04  2.043100e+00      0.000626
           128       1000  1.702265e-16  8.350000e+03  4.175000e+00      0.000766
           256       1000  2.121445e-16  2.000500e+04  1.000250e+01      0.000768
           512       1000  2.010726e-16  4.071000e+04  2.035500e+01      0.000880
          1024       1000  2.557624e-16  9.974700e+04  4.987350e+01      0.000821
          2048        100  2.596579e-16  1.932900e+04  9.664500e+01      0.000954
          4096        100  2.513216e-16  4.360300e+04  2.180150e+02      0.000939
          8192        100  2.588279e-16  9.756400e+04  4.878200e+02      0.000924
         16384        100  2.764842e-16  2.219750e+05  1.109875e+03      0.000886
         32768         10  2.827966e-16  4.299100e+04  2.149550e+03      0.000991
         65536         10  2.982493e-16  9.260800e+04  4.630400e+03      0.000991
        131072         10  2.947656e-16  1.893660e+05  9.468300e+03      0.001038
        262144         10  3.017082e-16  4.585860e+05  2.292930e+04      0.000915
        524288          8  3.204023e-16  8.045270e+05  5.028294e+04      0.000886
       1048576          8  3.288840e-16  1.744318e+06  1.090199e+05      0.000866
       2097152          8  3.500637e-16  3.559857e+06  2.224911e+05      0.000895
       4194304          8  3.488141e-16  7.563833e+06  4.727396e+05      0.000887

FFT_SERIAL:
  Normal end of execution.

21 November 2023 03:51:32 AM
zyt@zyt-VirtualBox:~/hpc/lab4/fft_serial$ g++ fft_parallel.cpp -o fft_parallel -pthread -lm
zyt@zyt-VirtualBox:~/hpc/lab4/fft_serial$ ./fft_parallel
             8      10000  8.086724e-17  5.826000e+03  2.913000e-01      0.000137
            16      10000  7.339336e-17  1.119900e+04  5.599500e-01      0.000286
            32      10000  1.470160e-16  1.986500e+04  9.932500e-01      0.000483
            64      10000  1.795907e-16  4.462700e+04  2.231350e+00      0.000574
           128       1000  1.702265e-16  9.357000e+03  4.678500e+00      0.000684
           256       1000  2.121445e-16  1.777300e+04  8.886500e+00      0.000864
           512       1000  2.010726e-16  2.922200e+04  1.461100e+01      0.001226
          1024       1000  2.557624e-16  6.687700e+04  3.343850e+01      0.001225
          2048        100  2.596579e-16  1.286300e+04  6.431500e+01      0.001433
          4096        100  2.513216e-16  2.654800e+04  1.327400e+02      0.001543
          8192        100  2.588279e-16  5.924800e+04  2.962400e+02      0.001521
         16384        100  2.764842e-16  1.128540e+05  5.642700e+02      0.001742
         32768         10  2.827966e-16  2.150800e+04  1.075400e+03      0.001981
         65536         10  2.982493e-16  5.254100e+04  2.627050e+03      0.001746
        131072         10  2.947656e-16  1.066960e+05  5.334800e+03      0.001843
        262144         10  3.017082e-16  2.408840e+05  1.204420e+04      0.001741
        524288          8  3.204023e-16  4.214880e+05  2.634300e+04      0.001692
       1048576          8  3.288840e-16  1.074795e+06  6.717469e+04      0.001405
       2097152          8  3.500637e-16  2.120226e+06  1.325141e+05      0.001503
       4194304          8  3.488141e-16  5.780931e+06  3.613082e+05      0.001161

FFT_SERIAL:
```

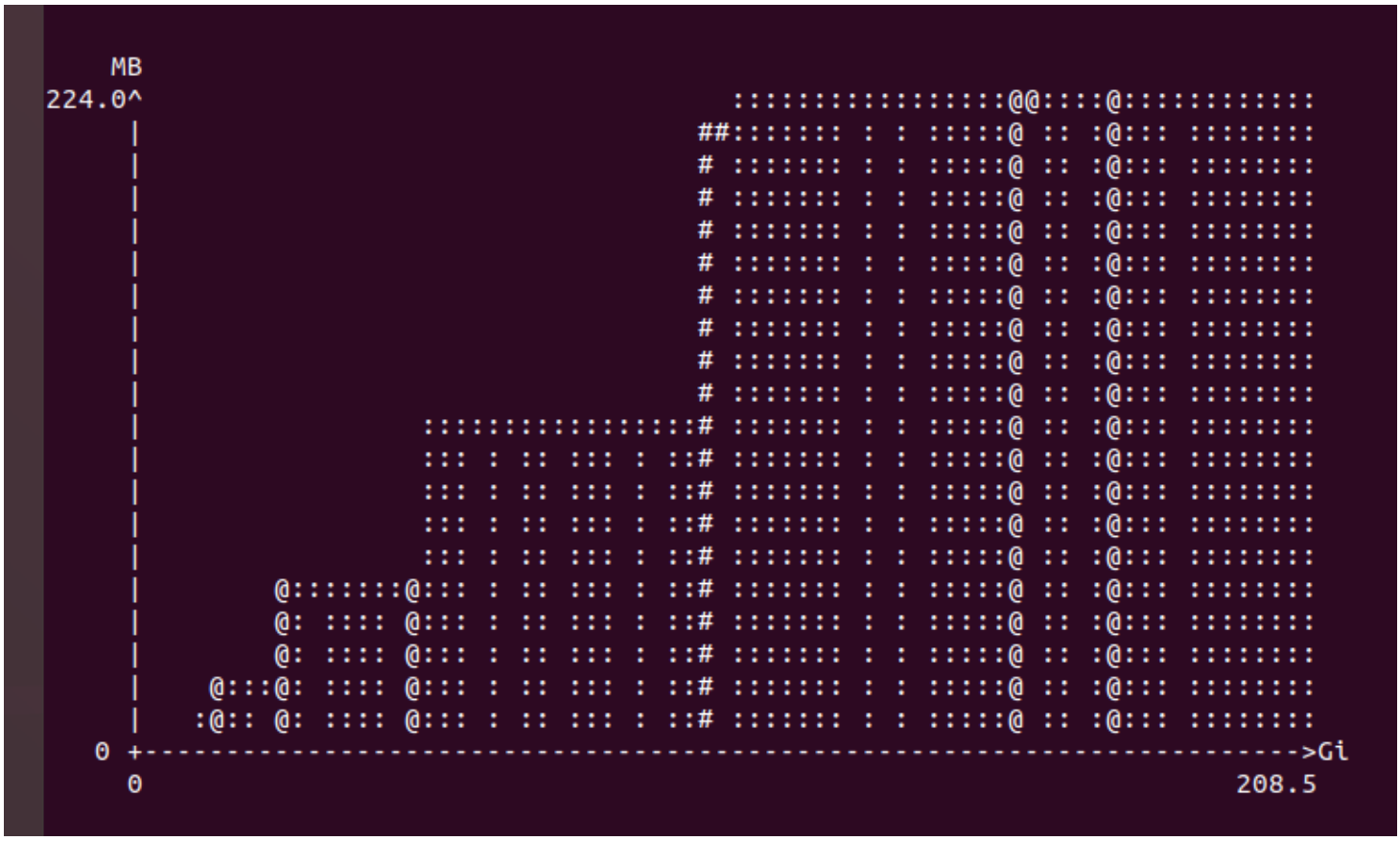
上面显示了1，2，4，8个线程进行计算的结果，因为我发现源代码在最后的迭代只有一次，这样的话就会导致吧多线程不会起作用，于是我另最后的迭代次数为8，方便观察多线程的性能。


MB
224.1^
0
0 +
219.5
>Gi
Number of snapshots: 64


MB
224.1^
0
0 +
218.3
>Gi


MB
224.1^
0
0 +
217.8
>Gi


MB
224.1^
0
0 +
219.1
>Gi



在更改问题规模的时候我发现，无论怎么更改线程数量，内存占用并没有太大变化，可能是因为多线程利用的是共享变量的原因。

实验结果

1. 展示在问题3中。
2.

```
zyt@zyt-VirtualBox:~/hpc/lab4/fft_serial$ mpirun -np 4 ./hp_mpi

  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  0.017712
      2048  0.009572
      4096  0.007758
      8192  0.004130
     16384  0.001113

     17046  0.001000

  Error tolerance achieved.
  Wallclock time = 29.405707
```

```
zyt@zyt-VirtualBox:~/hpc/lab4/fft_serial$ mpirun -np 1 ./hp_mpi

  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

     16957   0.001000

  Error tolerance achieved.
  Wallclock time = 47.358718
```

可以看到处理的时间明显减少了许多。

3.

在实验过程已经展示。

实验反思

加深了多线程编程和mpi编程的理解，同时在实验的过程中遇到不少问题，因为多线程和mpi调试比较麻烦，花费了不少时间。