中山大学计算机学院本科生实验报告

（2021学年秋季学期）

课程名称：高性能计算程序设计基础　　　　　　批改人：

实验	LAB5	专业	计算机科学与技术（超算）
学号	19335162	姓名	潘思晗
Email	pansh25@mail2.sysu.edu.cn	完成日期	2021.11.30

一、实验目的

> *1.*通过实验 *4*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.*任务二选一
>
> - 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。
>
> - 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*1*和并行化 *fft*应用，包括：
>
> - 不同问题规模的并行化 *fft*应用并行执行时间对比，其中问题规模定义为*N*变化范围*2*，*4*，*6*，*8*，*16*，*32*，*64*，*128*，*……*，*2097152*；并行规模为*1*，*2*，*4*，*8* 进/线程。
> - 内存消耗对比

二、实验过程和核心代码

1.改编fft_serial

函数文件

头文件

函数头文件中，不包含`parallel_for()`函数的实现，仅包含其定义和库文件

```
#ifndef PARALLEL_FOR_H
#define PARALLEL_FOR_H
void parallel_for(int start, int end, int increment, void *
(*functor)(void*), void *arg, int num_thread);
#endif
```

该文件命名为`parallel_for.h`。

实现文件

函数的实现文件与头文件同名，命名为`parallel_for.cpp`，为了能在过程中传递参数，我新建了`args`类，并且在`for_index`类中加入了指向args类的指针。

算法的实现如下所示：

```
#include<stdlib.h>
#include <pthread.h>
#include "parallel_for.h"

struct for_index {
    int start;
    int end;
    int increment;
    void *args;
};

struct args{
  double *A,*B,*C,*D,*W, sgn;
  int n, mj;
};
```

```c

void parallel_for(int start, int end, int increment, void *(*functor)(void *), void *arg, int num_thread)
{
    long thread;
    pthread_t *thread_handles;
    thread_handles = malloc(num_thread * sizeof(pthread_t)); // 为每个线程的pthread_t对象分配内存

    for (thread = 0; thread < num_thread; thread++)
    {
        // 确定每个线程的开始和结束
        int my_rank = thread;
        int my_first, my_last;
        int quotient = (end - start) / num_thread;
        int remainder = (end - start) % num_thread;
        int my_count;
        if (my_rank < remainder)
        {
            my_count = quotient + 1;
            my_first = my_rank * my_count;
        }
        else
        {
            my_count = quotient;
            my_first = my_rank * my_count + remainder;
        }
        my_last = my_first + my_count;
        struct for_index *index;
        index = malloc(sizeof(struct for_index));
        index->start = start + my_first;
        index->end = start + my_last;
        index->increment = increment;
        index->args = arg;
        pthread_create(&thread_handles[thread], NULL, functor, index);
    }

    for (thread = 0; thread < num_thread; thread++)
    {
        pthread_join(thread_handles[thread], NULL);
    }
    free(thread_handles);
```

```
}
```

函数改编

首先在main函数中添加接受线程数目的语句并打印：

```
thread_num = atoi(argv[1]);
printf("Number of Threads: %d\n", thread_num);
```

重点改编`step`函数，先使用`args`类将step函数接收到的参数初始化，如下所示：

```
int mj2 = 2 * mj;
int lj  = n / mj2;

struct args Arg;
Arg.A = a;
Arg.B = b;
Arg.C = c;
Arg.D = d;
Arg.W = w;
Arg.sgn = sgn;
Arg.n = n;
Arg.mj  = mj;
```

然后将`step`函数中的循环改编为函数的形式，创建函数`func_1()`，接受参数index的值以及index中args类的值，改造原本的for循环，具体代码如下所示：

```
void *func_1(void *args)
{
    struct for_index *index = (struct for_index *)args;
    struct args *true_arg = (struct args *)(index->args);
    double ambr;
    double ambu;
    int j, ja, jb, jc, jd, jw, k, lj, mj2;
    double wjw[2];

    mj2 = 2 * (true_arg->mj);
    lj  = (true_arg->n) / mj2;

```

```
    for ( int j = index->start; j < index->end; j = j + index->increment )
    {
      jw = j * (true_arg->mj);
      ja  = jw;
      jb  = ja;
      jc  = j * mj2;
      jd  = jc;

      wjw[0] = true_arg->W[jw*2+0];
      wjw[1] = true_arg->W[jw*2+1];

      if ( (true_arg->sgn) < 0.0 )
      {
        wjw[1] = - wjw[1];
      }

      for ( k = 0; k < (true_arg->mj); k++ )
      {
        true_arg->C[(jc+k)*2+0] = true_arg->A[(ja+k)*2+0] + true_arg->B[(jb+k)*2+0];
        true_arg->C[(jc+k)*2+1] = true_arg->A[(ja+k)*2+1] + true_arg->B[(jb+k)*2+1];

        ambr = true_arg->A[(ja+k)*2+0] - true_arg->B[(jb+k)*2+0];
        ambu = true_arg->A[(ja+k)*2+1] - true_arg->B[(jb+k)*2+1];

        true_arg->D[(jd+k)*2+0] = wjw[0] * ambr - wjw[1] * ambu;
        true_arg->D[(jd+k)*2+1] = wjw[1] * ambr + wjw[0] * ambu;
      }
    }
}
```

并在原来的循环位置调用函数`parallel for()`：

```
parallel_for(0, lj, 1, func_1, (void*)&Arg, thread_num);
```

改变完成后，编译运行的指令如下所示：

```
gcc -std=c99 -g -o fft_parallel fft_parallel.c -lpthread
./fft_parallel
```

2.改编成 MPI 版本

> 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。

首先将MPI的参数初始化：

```
MPI_Status status;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

再将对矩阵初始化的几个循环改编为 MPI 版本，随后改编循环计算直到误差小于0.001，多个线程同时进入循环并判断跳出的条件，`while(diff >= epsilon)`，在while循环内，使用 MPI_Sendrecv 进行数据的发送和接受，从而确定各个进程内的数据：

```
for (i = 0; i < local_m; i++)
    sendbuf[i] = w[i][1];
MPI_Sendrecv(sendbuf, local_m, MPI_DOUBLE, L, tag,
             recvbuf, local_m, MPI_DOUBLE, R, tag,
             MPI_COMM_WORLD, &status);
for (i = 0; i < local_m; i++)
    w[i][local_n + 1] = recvbuf[i];
for (i = 0; i < local_m; i++)
    sendbuf[i] = w[i][local_n];
MPI_Sendrecv(sendbuf, local_m, MPI_DOUBLE, R, tag,
             recvbuf, local_m, MPI_DOUBLE, L, tag,
             MPI_COMM_WORLD, &status);
for (i = 0; i < local_m; i++)
    w[i][0] = recvbuf[i];
```

之后改编保存结果和计算新结果和diff值的各个循环：

```
for (i = 0; i < local_m; i++)
{
    for (j = 0; j < local_n + 2; j++)
```

```
    {
        u[i][j] = w[i][j];
    }
}

for (i = 1; i < local_m - 1; i++)
{
    if (rank == 0)
    {
        start_col = 2;
        end_col = local_n;
    }
    else if (rank == size - 1)
    {
        start_col = 1;
        end_col = local_n - 1;
    }
    else
    {
        start_col = 1;
        end_col = local_n;
    }
    for (j = start_col; j < end_col + 1; j++)
    {
        w[i][j] = (u[i - 1][j] + u[i + 1][j] + u[i][j - 1] + u[i][j + 1]) / 4.0;
    }
}

diff = 0.0;
for (i = 1; i < local_m - 1; i++)
{
    if (rank == 0)
    {
        start_col = 2;
        end_col = local_n;
    }
    else if (rank == size - 1)
    {
        start_col = 1;
        end_col = local_n - 1;
    }
    else
```

```
    {
        start_col = 1;
        end_col = local_n;
    }
    for (j = start_col; j < end_col + 1; j++)
    {
        if (diff < fabs(w[i][j] - u[i][j]))
        {
            diff = fabs(w[i][j] - u[i][j]);
        }
    }
}
```

各个子进程将计算好的diff值发送给主进程，由主进程接收后进行比较，取最大值，最后使用 MPI_Bcast 函数对得到的 diff 值进行广播。

```
if (rank == 0)
{
    int i;
    double temp_diff = 999;
    for (i = 1; i < size; i++)
    {
        MPI_Recv(&temp_diff, 1, MPI_DOUBLE, i, 20, MPI_COMM_WORLD, &status);
        if (temp_diff > diff)
            diff = temp_diff;
    }
}
if (rank != 0)
{
    MPI_Send(&diff, 1, MPI_DOUBLE, 0, 20, MPI_COMM_WORLD);
}

MPI_Bcast(&diff, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
iterations++;
if (rank == 0)
{
    if (iterations == iterations_print)
    {
        printf("  %8d  %f\n", iterations, diff);
        iterations_print = 2 * iterations_print;
    }
```

```
}
}
```

3.性能分析

（1）时间对比

对不同问题规模的并行化fft应用并行执行时间对比，其中问题规模定义为N变化范围2，4，6，8，16，32，64，128，……， 2097152；并行规模为1，2，4，8 进/线程。

（2）内存消耗对比

内存消耗采用 “valgrind massif”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。

首先使用如下命令安装valgrind 工具：

```
sudo apt-get install valgrind
```

安装完成后，运行的示例命令如下：

```
valgrind --tool=massif --stacks=yes ./fft_parallel 1
ms_print massif.out.5247
```

```
valgrind --tool=massif --stacks=yes ./fft_parallel 2
ms_print massif.out.11637
```

具体结果见第三部分。

三、实验结果

打印CPU相关信息如下

```
emilylyly@emilylyly-VirtualBox:~$ lscpu
架构：            x86_64
CPU 运行模式：    32-bit, 64-bit
字节序：          Little Endian
CPU:              1
在线 CPU 列表：   0
每个核的线程数：  1
每个座的核数：    1
座：              1
NUMA 节点：       1
厂商 ID：         GenuineIntel
CPU 系列：        6
型号：            142
型号名称：        Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz
步进：            11
CPU MHz：         1992.002
BogoMIPS：        3984.00
超管理器厂商：    KVM
虚拟化类型：      完全
L1d 缓存：        32K
L1i 缓存：        32K
L2 缓存：         256K
L3 缓存：         8192K
NUMA 节点0 CPU：  0
标记：            fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge
freq pni pclmulqdq monitor ssse3 cx16 pcid sse4_1 sse4_2 x2apic movbe
_l1d arch_capabilities
```

1.改编fft_serial

编译和运行初始程序的命令如下：

```
gcc -std=c99 -g -o fft_serial fft_serial.c -lm
./fft_serial
```

编译和运行改造后程序的命令如下：

```
gcc -std=c99 -g -o fft_parallel fft_parallel.c -lpthread
./fft_parallel
```

未改造前的结果如下：

```
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab5/code1$ ./fft_serial
30 November 2021 03:57:33 PM

FFT_SERIAL
  C version

  Demonstrate an implementation of the Fast Fourier Transform
  of a complex data vector.

  Accuracy check:

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

             N      NITS    Error         Time          Time/Call     MFLOPS

             2     10000  7.859082e-17  2.706000e-03  1.353000e-07     73.909830
             4     10000  1.209837e-16  5.202000e-03  2.601000e-07    153.787005
             8     10000  6.820795e-17  9.542000e-03  4.771000e-07    251.519598
            16     10000  1.438671e-16  2.271300e-02  1.135650e-06    281.776956
            32      1000  1.331210e-16  4.008000e-03  2.004000e-06    399.201597
            64      1000  1.776545e-16  9.198000e-03  4.599000e-06    417.482061
           128      1000  1.929043e-16  2.247400e-02  1.123700e-05    398.682922
           256      1000  2.092319e-16  5.571600e-02  2.785800e-05    367.578433
           512       100  1.927488e-16  9.177000e-03  4.588500e-05    502.124877
          1024       100  2.308607e-16  2.237800e-02  1.118900e-04    457.592278
          2048       100  2.447624e-16  5.236300e-02  2.618150e-04    430.227451
          4096       100  2.479782e-16  9.266600e-02  4.633300e-04    530.421082
          8192        10  2.578088e-16  2.511600e-02  1.255800e-03    424.016563
         16384        10  2.733986e-16  7.110100e-02  3.555050e-03    322.605871
         32768        10  2.923012e-16  7.714400e-02  3.857200e-03    637.146116
         65536        10  2.829927e-16  2.630390e-01  1.315195e-02    398.638985
        131072         1  3.149670e-16  3.867800e-02  1.933900e-02    576.095972
        262144         1  3.218597e-16  1.506130e-01  7.530650e-02    313.292478
        524288         1  3.281373e-16  2.130850e-01  1.065425e-01    467.488185
       1048576         1  3.285898e-16  3.484670e-01  1.742335e-01    601.822267

FFT_SERIAL:
  Normal end of execution.
```

改造后的结果如下：

```
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab5/code1$ ./fft_parallel 1
Number of Threads: 1
30 November 2021 10:08:41 PM

FFT_SERIAL
  C version

  Demonstrate an implementation of the Fast Fourier Transform
  of a complex data vector.

  Accuracy check:

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

             N      NITS    Error         Time          Time/Call     MFLOPS

             2     10000  7.859082e-17  1.885440e-01  9.427200e-06      1.060760
             4     10000  1.209837e-16  3.681600e-01  1.840800e-05      2.172968
             8     10000  6.820795e-17  5.553690e-01  2.776845e-05      4.321451
            16     10000  1.438671e-16  7.748510e-01  3.874255e-05      8.259653
            32      1000  1.331210e-16  9.752500e-02  4.876250e-05     16.406050
            64      1000  1.776545e-16  1.245260e-01  6.226300e-05     30.836934
           128      1000  1.929043e-16  1.472430e-01  7.362150e-05     60.851789
           256      1000  2.092319e-16  1.998390e-01  9.991950e-05    102.482498
           512       100  1.927488e-16  3.449700e-02  1.724850e-04    133.576833
          1024       100  2.308607e-16  3.440800e-02  1.720400e-04    297.605208
          2048       100  2.447624e-16  5.141000e-02  2.570500e-04    438.202684
          4096       100  2.479782e-16  8.983600e-02  4.491800e-04    547.130326
          8192        10  2.578088e-16  1.499300e-02  7.496500e-04    710.304809
         16384        10  2.733986e-16  3.128200e-02  1.564100e-03    733.252350
         32768        10  2.923012e-16  6.824300e-02  3.412150e-03    720.249696
         65536        10  2.829927e-16  1.416500e-01  7.082500e-03    740.258383
        131072         1  3.149670e-16  2.987100e-02  1.493550e-02    745.948914
        262144         1  3.218597e-16  6.147500e-02  3.073750e-02    767.562749
        524288         1  3.281373e-16  1.249660e-01  6.248300e-02    797.134581
       1048576         1  3.285898e-16  2.764030e-01  1.382015e-01    758.729826

FFT_SERIAL:
  Normal end of execution.
```

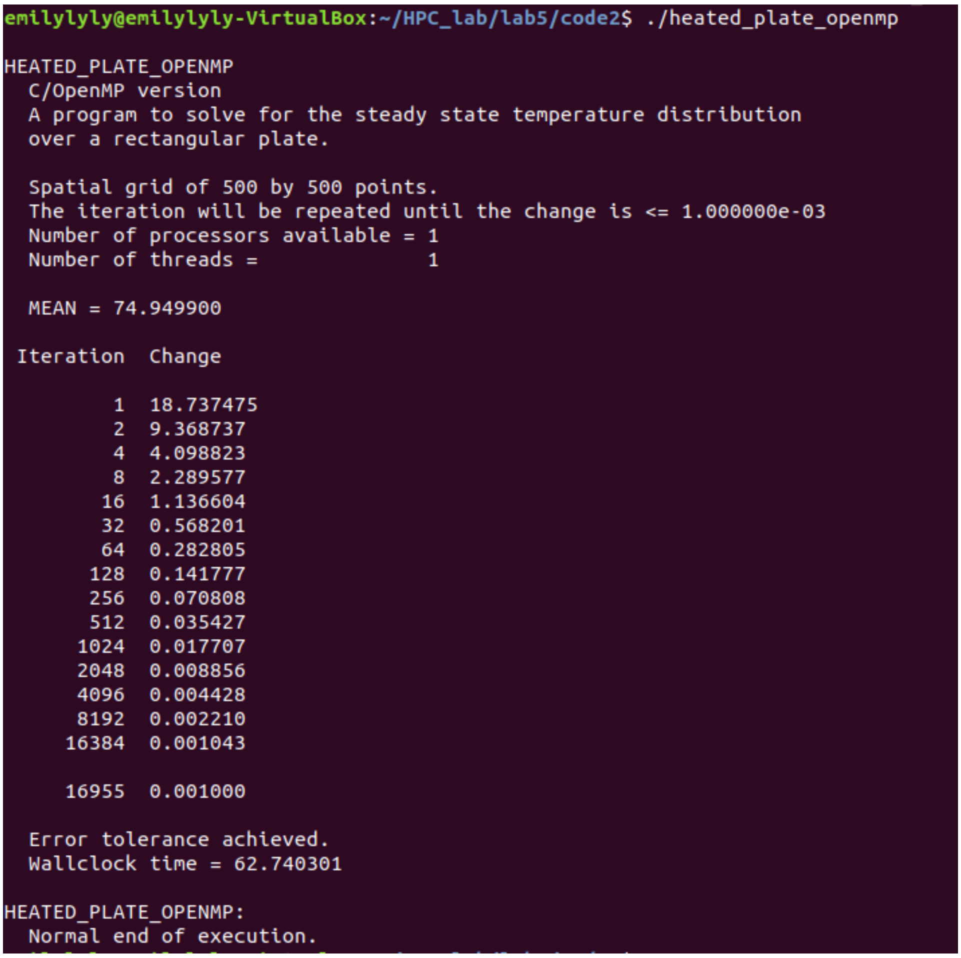
具体对比分析见第三部分：性能分析。

2.改编成MPI版本

编译和运行程序的命令如下：

```
mpicc -o MPI_parallel MPI_parallel.c
mpirun -np <number of threads> MPI_parallel
```

未修改前的结果如下：


```
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab5/code2$ ./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 processors available = 1
  Number of threads =              1

  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 = 62.740301

HEATED_PLATE_OPENMP:
  Normal end of execution.
```

改编为MPI版本后的结果：

进程数为1时：

```
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab5/code2$ mpirun -np 1 MPI_parallel

HEATED_PLATE_MPI
  C/MPI 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 threads =                1

  MEAN = 74.849699

 Iteration  Change

         1  31.287575
         2  10.928106
         4  4.093343
         8  2.286516
        16  1.135085
        32  0.567442
        64  0.282426
       128  0.141587
       256  0.070713
       512  0.035380
      1024  0.017684
      2048  0.008844
      4096  0.004422
      8192  0.002207
     16384  0.001042

     16946  0.001000

  Error tolerance achieved.
  Wallclock time = 44.697097
```

进程数为2：

```
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab5/code2$ mpirun -np 2 MPI_parallel

HEATED_PLATE_MPI
  C/MPI 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 threads =                2

  MEAN = 74.849699

 Iteration  Change

         1  18.712425
         2  9.356212
         4  4.093343
         8  2.286516
        16  1.135085
        32  0.567442
        64  0.282426
       128  0.141587
       256  0.070713
       512  0.035380
      1024  0.017684
      2048  0.008844
      4096  0.004422
      8192  0.002207
     16384  0.001041

     16933  0.001000

  Error tolerance achieved.
  Wallclock time = 46.485245
```

进程数为4：

```
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab5/code2$ mpirun -np 4 MPI_parallel

HEATED_PLATE_MPI
  C/MPI 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 threads =               4

  MEAN = 74.849699

 Iteration  Change

         1  18.712425
         2  9.356212
         4  4.093343
         8  2.286516
        16  1.135085
        32  0.567442
        64  0.282426
       128  0.141587
       256  0.070713
       512  0.035380
      1024  0.017684
      2048  0.008844
      4096  0.004422
      8192  0.002207
     16384  0.001041

     16933  0.001000

  Error tolerance achieved.
  Wallclock time = 60.364633
```

进程数为8时：

```
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab5/code2$ mpirun -np 8 MPI_parallel

HEATED_PLATE_MPI
  C/MPI 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 threads =               8

  MEAN = 74.849699

 Iteration  Change

         1  18.712425
         2  9.356212
         4  4.093343
         8  2.286516
        16  1.135085
        32  0.567442
        64  0.282426
       128  0.141587
       256  0.070713
       512  0.035380
      1024  0.017684
      2048  0.008844
      4096  0.004422
      8192  0.002207
     16384  0.001041

     16933  0.001000

  Error tolerance achieved.
  Wallclock time = 85.504255
```

从测试情况来看，结果是一致的，说明改编正确，但是使用MPI并行后，1~4个进程时运行时间有缩短，增加到8个进程后反而不明显，可能是通信开销增加了。

3.性能分析

（1）时间对比

初始串行版本：

```
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab5/code1$ ./fft_serial
30 November 2021 03:57:33 PM

FFT_SERIAL
  C version

  Demonstrate an implementation of the Fast Fourier Transform
  of a complex data vector.

  Accuracy check:

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

             N      NITS    Error         Time          Time/Call     MFLOPS

             2     10000  7.859082e-17  2.706000e-03  1.353000e-07     73.909830
             4     10000  1.209837e-16  5.202000e-03  2.601000e-07    153.787005
             8     10000  6.820795e-17  9.542000e-03  4.771000e-07    251.519598
            16     10000  1.438671e-16  2.271300e-02  1.135650e-06    281.776956
            32      1000  1.331210e-16  4.008000e-03  2.004000e-06    399.201597
            64      1000  1.776545e-16  9.198000e-03  4.599000e-06    417.482061
           128      1000  1.929043e-16  2.247400e-02  1.123700e-05    398.682922
           256      1000  2.092319e-16  5.571600e-02  2.785800e-05    367.578433
           512       100  1.927488e-16  9.177000e-03  4.588500e-05    502.124877
          1024       100  2.308607e-16  2.237800e-02  1.118900e-04    457.592278
          2048       100  2.447624e-16  5.236300e-02  2.618150e-04    430.227451
          4096       100  2.479782e-16  9.266600e-02  4.633300e-04    530.421082
          8192        10  2.578088e-16  2.511600e-02  1.255800e-03    424.016563
         16384        10  2.733986e-16  7.110100e-02  3.555050e-03    322.605871
         32768        10  2.923012e-16  7.714400e-02  3.857200e-03    637.146116
         65536        10  2.829927e-16  2.630390e-01  1.315195e-02    398.638985
        131072         1  3.149670e-16  3.867800e-02  1.933900e-02    576.095972
        262144         1  3.218597e-16  1.506130e-01  7.530650e-02    313.292478
        524288         1  3.281373e-16  2.130850e-01  1.065425e-01    467.488185
       1048576         1  3.285898e-16  3.484670e-01  1.742335e-01    601.822267

FFT_SERIAL:
  Normal end of execution.
```

并行规模为1时结果如下：

```
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab5/code1$ ./fft_parallel 1
Number of Threads: 1
30 November 2021 10:08:41 PM

FFT_SERIAL
  C version

  Demonstrate an implementation of the Fast Fourier Transform
  of a complex data vector.

  Accuracy check:

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

             N      NITS    Error          Time          Time/Call     MFLOPS

             2     10000  7.859082e-17  1.885440e-01  9.427200e-06      1.060760
             4     10000  1.209837e-16  3.681600e-01  1.840800e-05      2.172968
             8     10000  6.820795e-17  5.553690e-01  2.776845e-05      4.321451
            16     10000  1.438671e-16  7.748510e-01  3.874255e-05      8.259653
            32      1000  1.331210e-16  9.752500e-02  4.876250e-05     16.406050
            64      1000  1.776545e-16  1.245260e-01  6.226300e-05     30.836934
           128      1000  1.929043e-16  1.472430e-01  7.362150e-05     60.851789
           256      1000  2.092319e-16  1.998390e-01  9.991950e-05    102.482498
           512       100  1.927488e-16  3.449700e-02  1.724850e-04    133.576833
          1024       100  2.308607e-16  3.440800e-02  1.720400e-04    297.605208
          2048       100  2.447624e-16  5.141000e-02  2.570500e-04    438.202684
          4096       100  2.479782e-16  8.983600e-02  4.491800e-04    547.130326
          8192        10  2.578088e-16  1.499300e-02  7.496500e-04    710.304809
         16384        10  2.733986e-16  3.128200e-02  1.564100e-03    733.252350
         32768        10  2.923012e-16  6.824300e-02  3.412150e-03    720.249696
         65536        10  2.829927e-16  1.416500e-01  7.082500e-03    740.258383
        131072         1  3.149670e-16  2.987100e-02  1.493550e-02    745.948914
        262144         1  3.218597e-16  6.147500e-02  3.073750e-02    767.562749
        524288         1  3.281373e-16  1.249660e-01  6.248300e-02    797.134581
       1048576         1  3.285898e-16  2.764030e-01  1.382015e-01    758.729826

FFT_SERIAL:
  Normal end of execution.
```

并行规模为2时结果如下：

```
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab5/code1$ ./fft_parallel 2
Number of Threads: 2
30 November 2021 10:18:12 PM

FFT_SERIAL
  C version

  Demonstrate an implementation of the Fast Fourier Transform
  of a complex data vector.

  Accuracy check:

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

             N      NITS    Error          Time          Time/Call     MFLOPS

             2     10000  7.859082e-17  4.829290e-01  2.414645e-05      0.414140
             4     10000  1.209837e-16  1.035260e+00  5.176300e-05      0.772753
             8     10000  6.820795e-17  1.642647e+00  8.213235e-05      1.461056
            16     10000  1.438671e-16  2.226772e+00  1.113386e-04      2.874116
            32      1000  1.331210e-16  2.135930e-01  1.067965e-04      7.490882
            64      1000  1.776545e-16  3.358410e-01  1.679205e-04     11.433982
           128      1000  1.929043e-16  3.797830e-01  1.898915e-04     23.592420
           256      1000  2.092319e-16  4.329330e-01  2.164665e-04     47.305241
           512       100  1.927488e-16  5.137100e-02  2.568550e-04     89.700415
          1024       100  2.308607e-16  6.987300e-02  3.493650e-04    146.551601
          2048       100  2.447624e-16  9.233900e-02  4.616950e-04    243.970587
          4096       100  2.479782e-16  1.447560e-01  7.237800e-04    339.550692
          8192        10  2.578088e-16  2.326600e-02  1.163300e-03    457.732313
         16384        10  2.733986e-16  4.637900e-02  2.318950e-03    494.568663
         32768        10  2.923012e-16  8.476400e-02  4.238200e-03    579.868812
         65536        10  2.829927e-16  1.878130e-01  9.390650e-03    558.308530
        131072         1  3.149670e-16  3.729900e-02  1.864950e-02    597.395104
        262144         1  3.218597e-16  7.839500e-02  3.919750e-02    601.899611
        524288         1  3.281373e-16  1.740890e-01  8.704450e-02    572.205711
       1048576         1  3.285898e-16  3.671150e-01  1.835575e-01    571.252060

FFT_SERIAL:
  Normal end of execution.
```

并行规模为4时结果如下：

```
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab5/code1$ ./fft_parallel 4
Number of Threads: 4
30 November 2021 09:21:49 PM

FFT_SERIAL
  C version

  Demonstrate an implementation of the Fast Fourier Transform
  of a complex data vector.

  Accuracy check:

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

             N      NITS    Error         Time          Time/Call     MFLOPS

             2     10000  7.859082e-17  6.702030e-01  3.351015e-05      0.298417
             4     10000  1.209837e-16  1.365242e+00  6.826210e-05      0.585977
             8     10000  6.820795e-17  2.111126e+00  1.055563e-04      1.136834
            16     10000  1.438671e-16  2.766156e+00  1.383078e-04      2.313680
            32      1000  1.331210e-16  3.366930e-01  1.683465e-04      4.752104
            64      1000  1.776545e-16  4.060680e-01  2.030340e-04      9.456544
           128      1000  1.929043e-16  5.003470e-01  2.501735e-04     17.907572
           256      1000  2.092319e-16  5.787780e-01  2.893890e-04     35.384897
           512       100  1.927488e-16  6.890000e-02  3.445000e-04     66.879536
          1024       100  2.308607e-16  8.599900e-02  4.299950e-04    119.071152
          2048       100  2.447624e-16  1.068490e-01  5.342450e-04    210.839596
          4096       100  2.479782e-16  1.495600e-01  7.478000e-04    328.644022
          8192        10  2.578088e-16  2.299500e-02  1.149750e-03    463.126767
         16384        10  2.733986e-16  4.117900e-02  2.058950e-03    557.021783
         32768        10  2.923012e-16  7.649400e-02  3.824700e-03    642.560201
         65536        10  2.829927e-16  1.619520e-01  8.097600e-03    647.460976
        131072         1  3.149670e-16  2.922400e-02  1.461200e-02    762.463728
        262144         1  3.218597e-16  6.321000e-02  3.160500e-02    746.494542
        524288         1  3.281373e-16  1.342480e-01  6.712400e-02    742.020142
       1048576         1  3.285898e-16  2.821750e-01  1.410875e-01    743.209710

FFT_SERIAL:
  Normal end of execution.
```

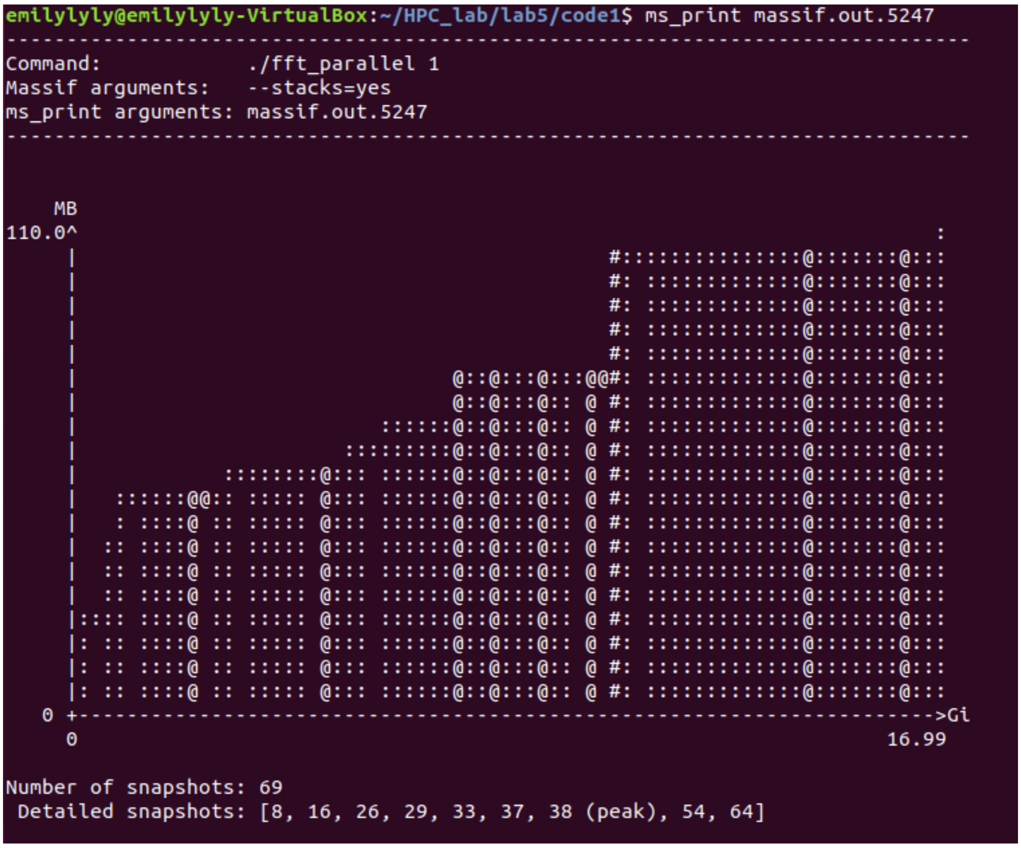
并行规模为8时结果如下：

```
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab5/code1$ ./fft_parallel 8
Number of Threads: 8
30 November 2021 09:54:58 PM

FFT_SERIAL
  C version

  Demonstrate an implementation of the Fast Fourier Transform
  of a complex data vector.

  Accuracy check:

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

             N      NITS    Error         Time          Time/Call     MFLOPS

             2     10000  7.859082e-17  1.965425e+00  9.827125e-05      0.101759
             4     10000  1.209837e-16  3.965977e+00  1.982989e-04      0.201716
             8     10000  6.820795e-17  6.261539e+00  3.130770e-04      0.383292
            16     10000  1.438671e-16  8.669114e+00  4.334557e-04      0.738253
            32      1000  1.331210e-16  1.048807e+00  5.244035e-04      1.525543
            64      1000  1.776545e-16  1.260786e+00  6.303930e-04      3.045719
           128      1000  1.929043e-16  1.451771e+00  7.258855e-04      6.171772
           256      1000  2.092319e-16  1.697341e+00  8.486705e-04     12.065931
           512       100  1.927488e-16  1.953510e-01  9.767550e-04     23.588310
          1024       100  2.308607e-16  2.177490e-01  1.088745e-03     47.026622
          2048       100  2.447624e-16  2.543550e-01  1.271775e-03     88.569126
          4096       100  2.479782e-16  3.223870e-01  1.611935e-03    152.462723
          8192        10  2.578088e-16  3.967500e-02  1.983750e-03    268.420920
         16384        10  2.733986e-16  5.923500e-02  2.961750e-03    387.230522
         32768        10  2.923012e-16  9.546300e-02  4.773150e-03    514.880111
         65536        10  2.829927e-16  1.749530e-01  8.747650e-03    599.347253
        131072         1  3.149670e-16  3.318200e-02  1.659100e-02    671.515882
        262144         1  3.218597e-16  6.886200e-02  3.443100e-02    685.224362
        524288         1  3.281373e-16  1.370220e-01  6.851100e-02    726.998000
       1048576         1  3.285898e-16  3.041500e-01  1.520750e-01    689.512412

FFT_SERIAL:
  Normal end of execution.
```

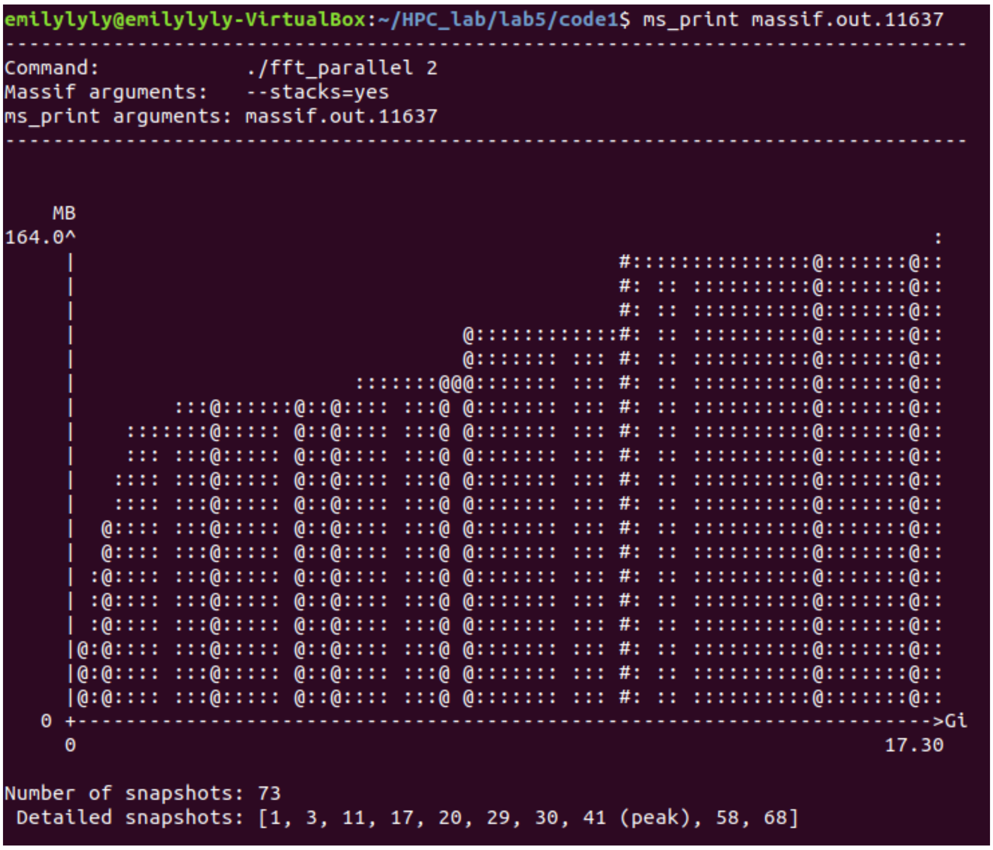
对比分析可以看到，在N较小时，例如N=2048及以下，此时串行fft的执行时间最短，并行fft的并行规模越大，时间越长，性能越差，而随着N的增大，例如N增长到4096及以上，此时并行的优势显现了出来，随着并行规模的增长，所花时间明显减小，且同样N的情况下并行规模越大，时间越短，优势更明显。

（2）内存消耗对比

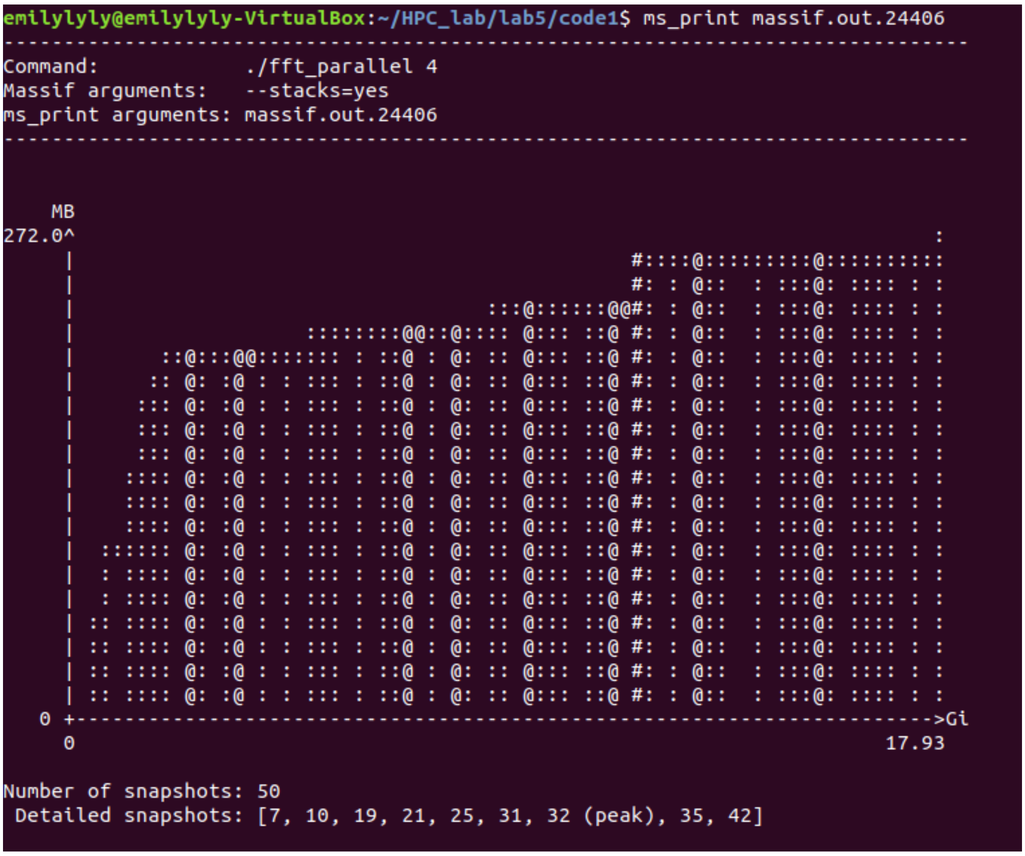
并行规模为1的内存消耗图像：



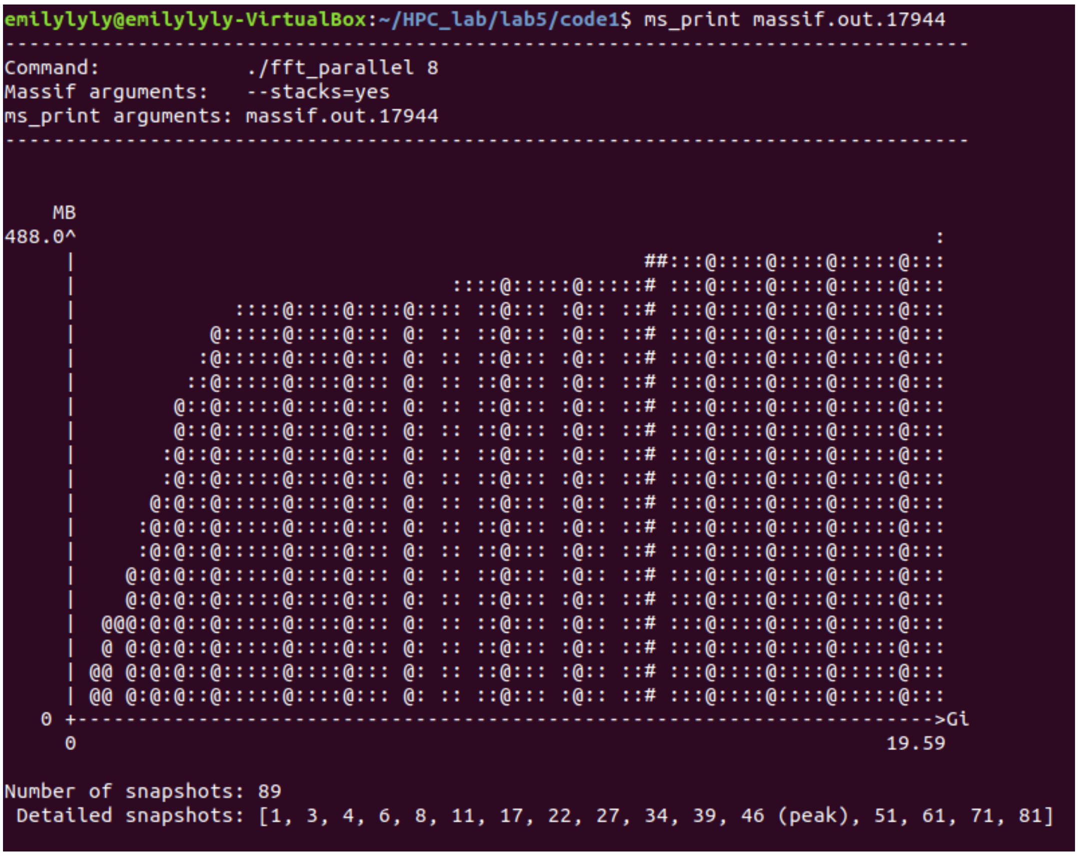
并行规模为2的内存消耗图像：



并行规模为4的内存消耗图像：



并行规模为8的内存消耗图像：



通过以上图像的对比可以看出，随着运行时间的增加，内存消耗量都是在不断增大的，并且并行规模越大，最终的内存消耗量越大。

四、实验感想

这次实验使用之前编写的并行函数对fft程序做了改编，过程中遇到对的最大难题就是关于如何传递函数参数的问题，通过网上资料查询以及与同学交流，总结出了使用结构体传递参数的方法，成功解决这一问题。

此外，在编译fft文件时一开始持续报错

```
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab5/code1$ gcc -std=c99 -g -o fft_serial fft_serial.c
/tmp/ccCWJ8o2.o：在函数‘main’中：
/home/emilylyly/HPC_lab/lab5/code1/fft_serial.c:167：对‘pow’未定义的引用
/home/emilylyly/HPC_lab/lab5/code1/fft_serial.c:168：对‘pow’未定义的引用
/home/emilylyly/HPC_lab/lab5/code1/fft_serial.c:170：对‘sqrt’未定义的引用
/tmp/ccCWJ8o2.o：在函数‘cfft2’中：
/home/emilylyly/HPC_lab/lab5/code1/fft_serial.c:315：对‘log’未定义的引用
/tmp/ccCWJ8o2.o：在函数‘cffti’中：
/home/emilylyly/HPC_lab/lab5/code1/fft_serial.c:401：对‘cos’未定义的引用
/home/emilylyly/HPC_lab/lab5/code1/fft_serial.c:402：对‘sin’未定义的引用
/tmp/ccCWJ8o2.o：在函数‘ggl’中：
/home/emilylyly/HPC_lab/lab5/code1/fft_serial.c:474：对‘fmod’未定义的引用
collect2: error: ld returned 1 exit status
```

后来发现是因为fft中调用了<math.h>的库所致，要在编译命令中加入`-lm`参数，添加参数之后没有报错。

总体而言，通过这次实验，我对于并行程序的编写有了新的理解和体会，受益匪浅。

五、参考资料

1.https://blog.csdn.net/u013176681/article/details/18272879

2.https://blog.csdn.net/xianquji1676/article/details/106168317

3.https://blog.csdn.net/u012206617/article/details/94383568