中山大学计算机学院本科生实验报告

课程名称：并行程序设计与算法

实验	MPI并行应用	专业（方向）	计算机科学与技术
学号	21307035	姓名	邓栩瀛
Email	dengxy66@mail2.sysu.edu.cn	完成日期	2024.5.6

1、实验目的

使用MPI对快速傅里叶变换进行并行化。

问题描述：阅读参考文献中的串行傅里叶变换代码(fft_serial.cpp)，并使用MPI对其进行并行化。

要求：

1. 并行化：使用MPI多进程对fft_serial.cpp进行并行化。为适应MPI的消息传递机制，可能需要对fft_serial代码进行一定调整。

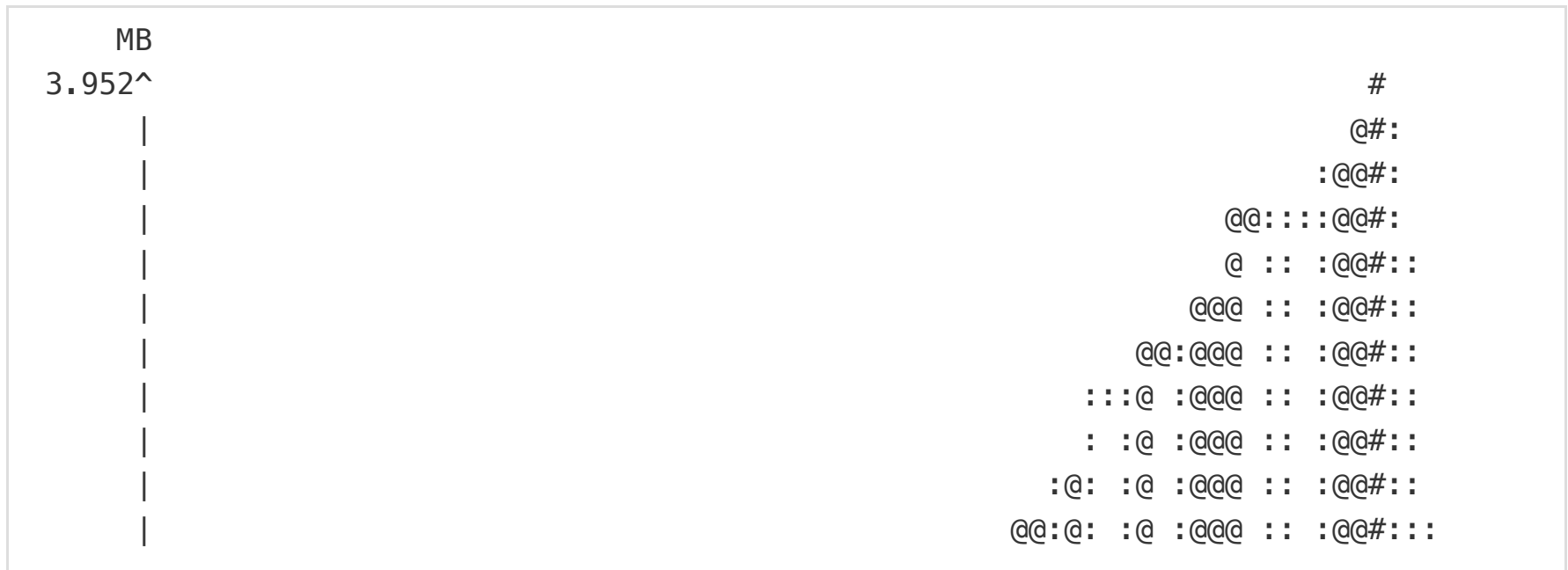
2. 优化：使用MPI_Pack/MPI-Unpack或MPI_Type_create_struct对数据重组后进行消息传递。

3. 分析：

   a) 改变并行规模（进程数）及问题规模（N），分析程序的并行性能；

   b) 通过实验对比，分析数据打包对于并行程序性能的影响；

   c) 使用Valgrind massif工具集采集并分析并行程序的内存消耗。注意Valgrind命令中增加--stacks=yes 参数采集程序运行栈内内存消耗。Valgrind 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：（注：该工具使用可参考https://valgrind.org/docs/manual/ms-manual.html）



```
    |                          :       ::         ::@@:@: :@ :@@@ :: :@@#:::
    |                        :@@:    ::::: ::::@@@:::@@:@: :@ :@@@ :: :@@#:::
    |                     ::::@@:  ::: ::::::: @  :::@@:@: :@ :@@@ :: :@@#:::
    |                    @: ::@@:  ::: ::::::: @  :::@@:@: :@ :@@@ :: :@@#:::
    |                    @: ::@@:  ::: ::::::: @  :::@@:@: :@ :@@@ :: :@@#:::
    |                    @: ::@@:::::: ::::::: @  :::@@:@: :@ :@@@ :: :@@#:::
    |                ::@@@: ::@@:: ::: ::::::: @  :::@@:@: :@ :@@@ :: :@@#:::
    |             :::::@ @: ::@@:: ::: ::::::: @  :::@@:@: :@ :@@@ :: :@@#:::
    |           @@:::::@ @: ::@@:: ::: ::::::: @  :::@@:@: :@ :@@@ :: :@@#:::
  0 +-------------------------------------------------------------------->Mi
    0                                                                   626.4

Number of snapshots: 63
 Detailed snapshots: [3, 4, 10, 11, 15, 16, 29, 33, 34, 36, 39, 41,
                      42, 43, 44, 49, 50, 51, 53, 55, 56, 57 (peak)]
```

2、实验过程和核心代码

执行命令

```
mpic++ main.cpp -o main.out
mpirun -np <num_process> ./main.out
mpic++ main2.cpp -o main2.out
mpirun -np <num_process> ./main2.out
mpirun -np <num_process> valgrind --tool=massif --stacks=yes ./main.out
```

核心代码

1.并行化：使用MPI多进程对fft_serial.cpp进行并行化

MPI相关代码

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

在并行环境下执行快速傅里叶变换（FFT）

```
for (it = 0; it < nits; it++)
{
     // 数据分发
     MPI_Bcast(x, 2 * n, MPI_DOUBLE, 0, MPI_COMM_WORLD);
     // 执行本地FFT计算
     cfft2(n, x, y, w, sgn);
```

```
        // 结果汇总
        MPI_Gather(y, 2 * n / size, MPI_DOUBLE, x, 2 * n / size, MPI_DOUBLE, 0,
MPI_COMM_WORLD);
        // 根进程的FFT计算
        if (rank == 0)
        {
            cfft2(n, x, y, w, sgn);
        }
}
```

2.优化：使用MPI_Pack/MPI-Unpack对数据重组后进行消息传递

```
for (it = 0; it < nits; it++)
{
    if (rank == 0)
    {
        // 根进程：将数据x打包到packed_buffer中
        MPI_Pack(x, 2 * n, MPI_DOUBLE, packed_buffer, 2 * n * sizeof(double), &i,
MPI_COMM_WORLD);
    }
    // 使用 MPI_Bcast 将打包后的数据 packed_buffer 广播给所有进程
    MPI_Bcast(packed_buffer, 2 * n, MPI_PACKED, 0, MPI_COMM_WORLD);
    if (rank == 0)
    {
        //根进程：解包，从packed_buffer中接收到的数据到存入y中
        MPI_Unpack(packed_buffer, 2 * n * sizeof(double), &i, y, 2 * n, MPI_DOUBLE,
MPI_COMM_WORLD);
    }
    // 执行本地的 FFT 计算
    cfft2(n, y, x, w, sgn);
    // 使用MPI_Gather将每个进程的计算结果x收集到根进程的数据y中
    MPI_Gather(x, 2 * n / size, MPI_DOUBLE, y, 2 * n / size, MPI_DOUBLE, 0,
MPI_COMM_WORLD);
    // 根进程：对收集到的数据y执行最终的FFT计算，得到最终的结果x
    if (rank == 0)
    {
        cfft2(n, y, x, w, sgn);
    }
}
```

3、实验结果

1.改变并行规模（进程数）及问题规模（N），分析程序的并行性能

进程数	问题规模				
	128	256	512	1024	2048
1	0.006659	0.015334	0.003164	0.007221	0.015131
2	0.008098	0.017871	0.003596	0.007859	0.016439
4	0.009680	0.020762	0.003866	0.008129	0.016989
8	0.035223	0.058377	0.015492	0.029054	0.056682

实验结果分析：

1. 在每个并行规模下，随着问题规模的增加，计算时间也相应增加，这是因为随着问题规模的增加，需要处理更多的数据，导致计算量增加，从而增加了计算时间。
2. 在某些情况下，随着进程数的增加，计算时间可能会减少，这是因为可以将计算任务分配给更多的处理器来并行处理，从而提高了计算效率。但是在其他情况下，随着进程数的增加，可能会出现额外的通信开销和负载不均衡等问题，导致计算时间反而增加。
3. 在并行计算中，负载均衡是非常关键的，如果任务分配不均匀，一些处理器可能会负担更重的工作量，从而导致性能下降。

2.数据打包对于并行程序性能的影响

数据打包	问题规模				
	128	256	512	1024	2048
True	0.007149	0.016647	0.003516	0.007483	0.015517
False	0.006659	0.015334	0.003164	0.007221	0.015131

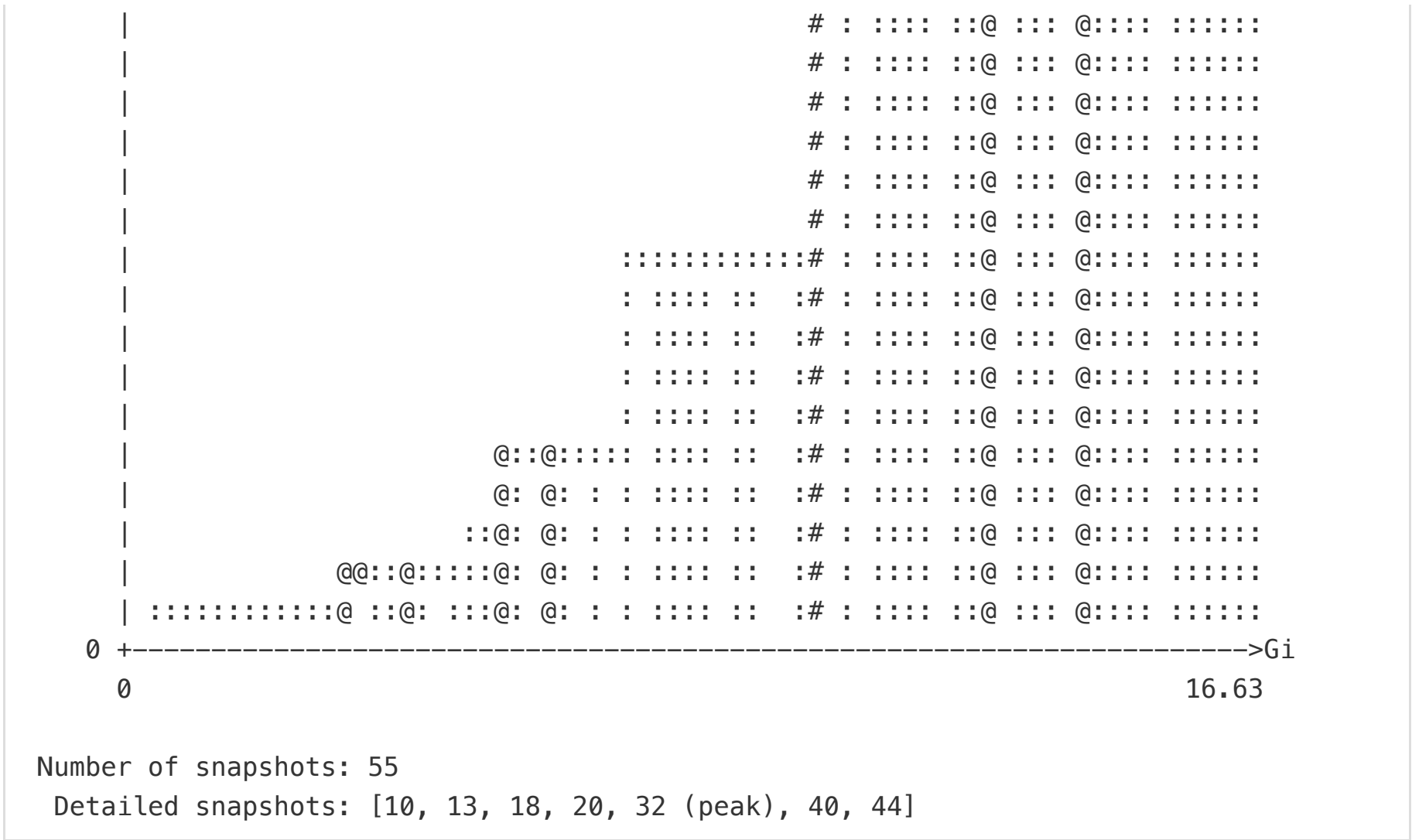
实验结果分析：

1. 数据打包会将多个数据项合并成一个较大的数据块进行传输。虽然这样可以减少通信开销，但在某些情况下，较大的数据块可能会增加传输延迟或者导致资源利用不均衡。
2. 使用数据打包可能会导致处理时间分布不均匀，即使某些处理器处理速度较快，也需要等待其他处理器处理完整个数据块才能继续进行，可能导致某些处理器空闲等待的时间增加，从而影响了整体的性能表现。

3.使用Valgrind massif工具集采集并分析并行程序的内存消耗

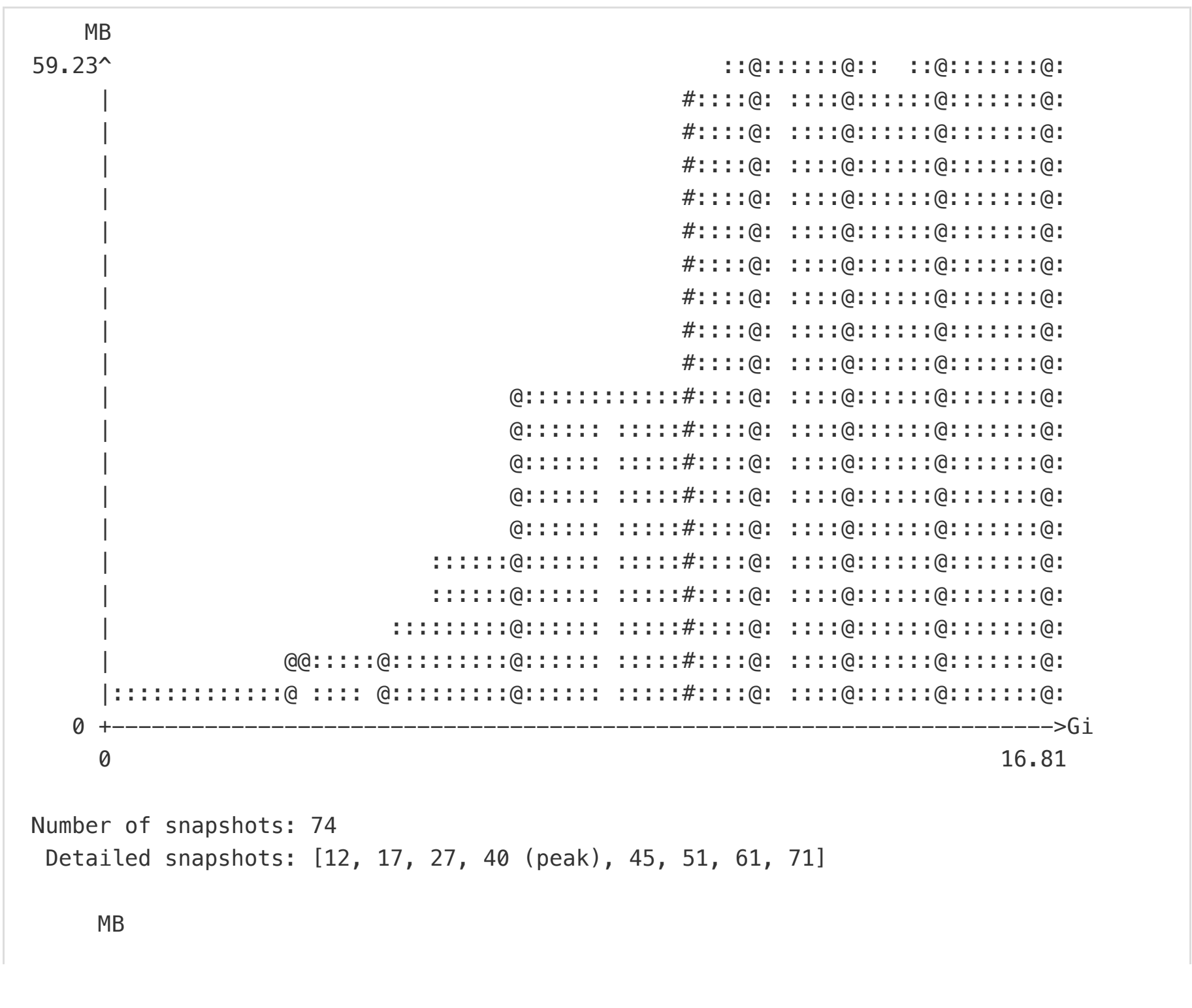
进程数为1的情况

```
    MB
59.21^                                                      :::::::@@::   ::::::::::::
     |                                                ##:::::: ::@ ::::@:::: ::::::
     |                                                # : :::: ::@ ::: @:::: ::::::
     |                                                # : :::: ::@ ::: @:::: ::::::
```




```
Number of snapshots: 55
 Detailed snapshots: [10, 13, 18, 20, 32 (peak), 40, 44]
```

进程数为2的情况




```
Number of snapshots: 74
 Detailed snapshots: [12, 17, 27, 40 (peak), 45, 51, 61, 71]
```

```
59.24^                                                    : ::@
     |                                              #::::@::::::@::::@::::::::@:::
     |                                              #::::@::::::@::::@::::::::@:::
     |                                              #::::@::::::@::::@::::::::@:::
     |                                              #::::@::::::@::::@::::::::@:::
     |                                              #::::@::::::@::::@::::::::@:::
     |                                              #::::@::::::@::::@::::::::@:::
     |                                              #::::@::::::@::::@::::::::@:::
     |                                              #::::@::::::@::::@::::::::@:::
     |                                              #::::@::::::@::::@::::::::@:::
     |                                              #::::@::::::@::::@::::::::@:::
     |                                    @::::::::::::@:#::::@::::::@::::@::::::::@:::
     |                                    @::::::::::::@:#::::@::::::@::::@::::::::@:::
     |                                    @::::::::::::@:#::::@::::::@::::@::::::::@:::
     |                                    @::::::::::::@:#::::@::::::@::::@::::::::@:::
     |                                    @::::::::::::@:#::::@::::::@::::@::::::::@:::
     |                                    @::::::::::::@:#::::@::::::@::::@::::::::@:::
     |                              @::::::@::::::::::::@:#::::@::::::@::::@::::::::@:::
     |                               @::::::@::::::::::::@:#::::@::::::@::::@::::::::@:::
     |                              ::@::::::@::::::::::::@:#::::@::::::@::::@::::::::@:::
     |                         @@::::@::@::::::@::::::::::::@:#::::@::::::@::::@::::::::@:::
     |  ::::@:::@:::@@:: :@::@::::::@::::::::::::@:#::::@::::::@::::@::::::::@:::
   0 +-------------------------------------------------------------------------------->Gi
     0                                                                                18.12

Number of snapshots: 85
 Detailed snapshots: [5, 10, 13, 14, 18, 21, 28, 42, 44 (peak), 49, 59, 69, 79]
```

实验结果分析：

1. 输出结果中的图形显示了程序在不同时间点的内存消耗情况，从输出结果中可以观察到，内存消耗在不同时间点有所波动，但总体趋势是逐渐增加的。
2. 随着进程数的增加，内存消耗量也略微增加，这可能是由于并行执行时需要额外的内存来存储通信数据等。但是，增加的量相对较小，说明程序在不同进程数下内存消耗的增长并不明显，整体上表现稳定。

4、实验感想

1. 将串行代码并行化首先需要对原有代码进行深入了解，例如，在本实验中，需要使用 MPI 对快速傅里叶变换 (FFT) 进行并行化，首先需要对算法进行深入理解，才能在多个进程之间合理地划分工作负载。
2. 在并行计算中，进程之间的通信是一个关键问题，在编写代码的时候，需要保证代码能够正确地发送和接收消息，以及正确地组织数据，从而避免死锁、竞态条件等问题。
3. 在实验中，我对程序的性能进行了分析，尝试使用了一些优化方法，例如使用 MPI_Pack/MPI-Unpack等来进行数据打包以提高消息传递的效率。
4. 通过使用 Valgrind Massif 工具集来分析程序的内存消耗，更加深入了解了内存管理在程序性能和稳定性中的

重要性，通过这个辅助工具，可以辅助识别内存泄漏和优化内存使用，从而提高程序的健壮性。

5. 通过完成本次实验，我不仅更加深入了解在MPI在实现并行计算中的应用，还学习了如何使用Valgrind等类似的工具来分析和优化程序。