**并行分布式计算**

**课程报告**

**姓名：林睿**

**学号：21307130150**

1. 任务描述
   1. 选题

使用OpenMP编程实现并行快速排序(PQS)算法

使用MPI实现Parallel sorting by regular sampling(PSRS)算法

* 1. 具体任务细节

在本实验中主要实现了两个C++程序，分别使用OpenMP框架编程实现并行快速排序算法和使用MPI实现并行正则采样排序算法。程序分别通过命令行接收两个参数<desired\_num\_threads> 和 <desire\_size\_of\_array>来对并发进程数和待排序数组大小进行控制，并在程序中包含了对错误命令行参数的处理、给定数组大小的随机数组生成、排序算法具体实现、打印结果、结果比较与验证（和串行快速排序）、性能度量（以串行的快速排序作为基准）等具体任务细节。

* 1. 实验环境

Ubuntu 2 Linux(wsl)环境

1. 算法逻辑
   1. 并行快速排序算法
      1. 快速排序算法

快速排序算法的基本流程如下：

选择基准元素：从数组中选择一个元素作为基准（通常是数组中间的元素）。

划分阶段：将数组中的元素分为两部分，小于基准的元素放在左侧，大于基准的元素放在右侧。

递归阶段：对左右两个部分递归地应用快速排序。

合并阶段：在递归的过程中，数组逐渐变得有序。

* + 1. 并行化处理

在并行快速排序中，可以通过以下方式实现并行化：

任务分配：将划分和递归的过程分解为任务，使用多个进程同时处理这些任务。

并行递归：对数组划分后，左右两个部分的排序可以并行进行，从而加速整体排序过程。

阈值处理：为了避免过多的进程创建和管理开销，可以设置一个阈值，当子数组的大小小于阈值时，使用串行排序。（阈值为所设置的进程数）

通过这些并行化措施，可以在多核处理器上更有效地利用资源，加速排序过程。

* 1. 并行正则采样排序算法
     1. 正则采样排序算法

正则采样排序算法的基本流程如下：

无序序列的划分及局部排序：根据数据块的划分方法，将无序序列划分成p(p为处理器或进程数)部分，每个处理器对其中的一部分进行串行快速排序，这样每个处理器就会拥有一个局部有序序列。

正则采样：每个处理器从局部有序序列中选取第w,2w,...,(p-1)w共p-1个代表元素。其中w = n/p^2。

确定主元：每个处理器都将自己选取好的代表元素发送给处理器p0。p0对这p段有序序列做多路归并排序，再从这排序后的序列中选取第p-1,2(p-1), ...,(p-1)(p-1)共p-1个元素作为主元。

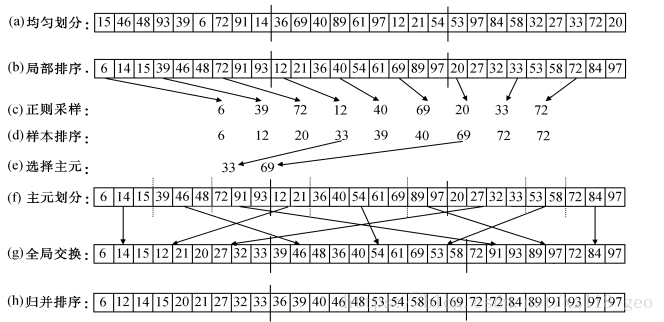
分发主元：将这p-1个主元分发给各个处理器。

局部有序序列划分：每个处理器在接收到主元后，根据主元将自己的局部有序序列划分成p段。

p段有序序列的分发：每个处理器将自己的第i段发送给第i个处理器，是处理器i都拥有所有处理器的第i段。

多路排序：每个处理器将上一步得到的p段有序序列做多路归并。

经过这7步后，一次将每个处理器的数据取出，这些数据是有序的。示例图如下：



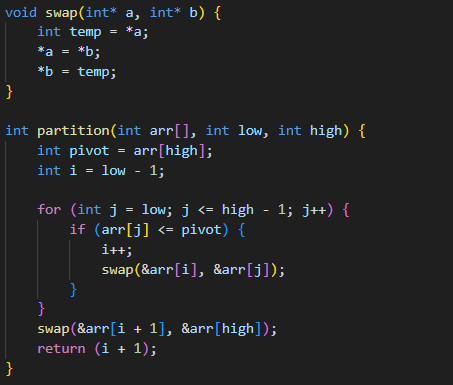
* + 1. 并行化处理

在正则采样排序中，可以通过以下方式实现并行化：

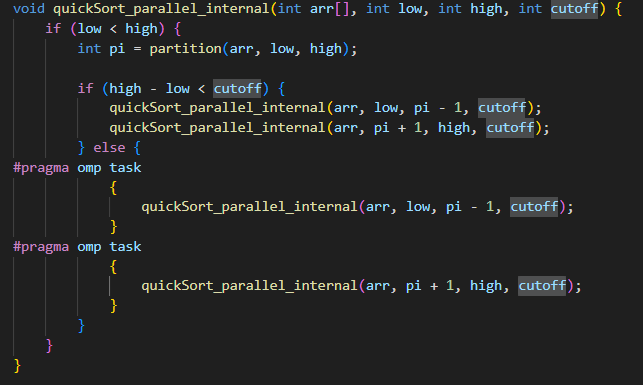
无序序列的划分及局部排序中，将无序序列划分成为p部分，并由每个处理器并行地对其负责的部分进行串行快速排序并一定间隔地选取样本。局部有序序列划分时也可以并行进行。p段有序序列的分发可以在每个处理器之间进行通讯，这个步骤也可以并行。最终的多路归并排序也可以并行。

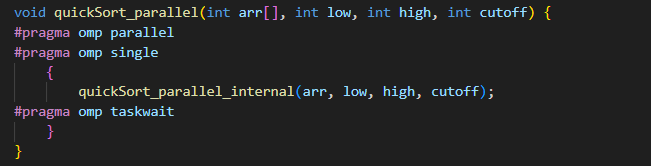
1. 编程代码实现
   1. 并行快速排序算法(OpenMP)

首先定义元素交换、划分的方法

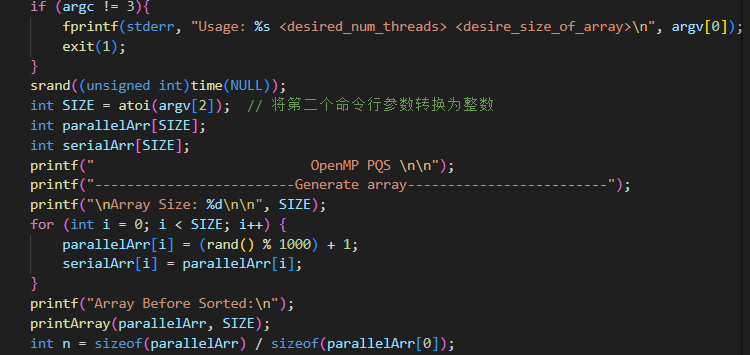


在快速排序实现的代码部分，将对两个子段的并行快速排序递归调用分配成两个任务（OpenMP调用），并且定义了一个cutoff值，如果此段数组的长度小于cutoff值（后面将其设为100），则直接使用串行化代码以提高速度并且防止无止境的递归调用。在quickSort\_parallel中创建一个进程执行内部的程序并且等待由当前进程创建的所有并行任务执行完。

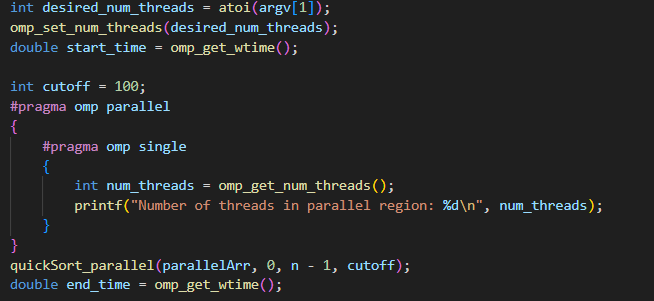




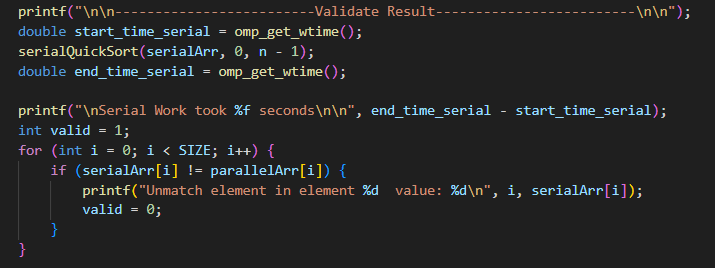
主程序中，先根据指定的数组大小，随机产生数组（元素不妨设置为1000以内，只要是int类型的，都不会改变排序算法的开销）



然后设置并行区域所使用的进程数，并且执行排序算法，以及测试时间。

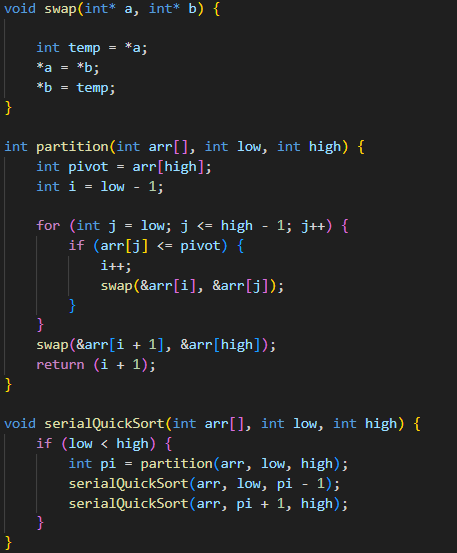
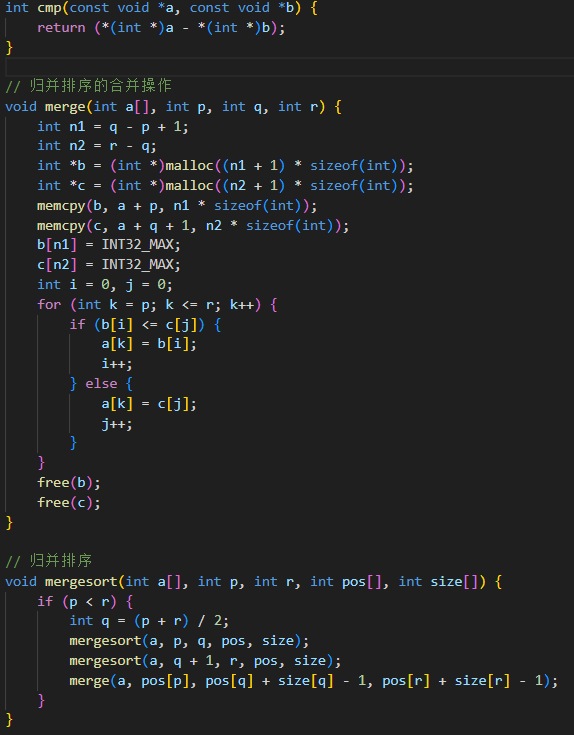
1701055763099

最后执行串行快速排序算法，比对结果与测试串行算法所需要时间以度量加速比。



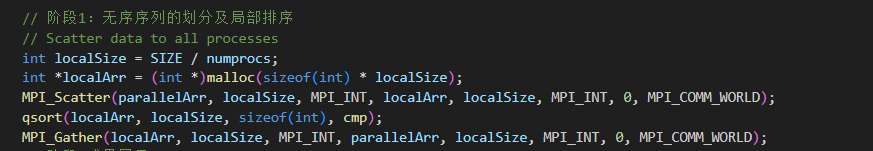
* 1. 并行正则采样排序算法(MPI)

定义归并排序和快速排序等算法方便后续使用。

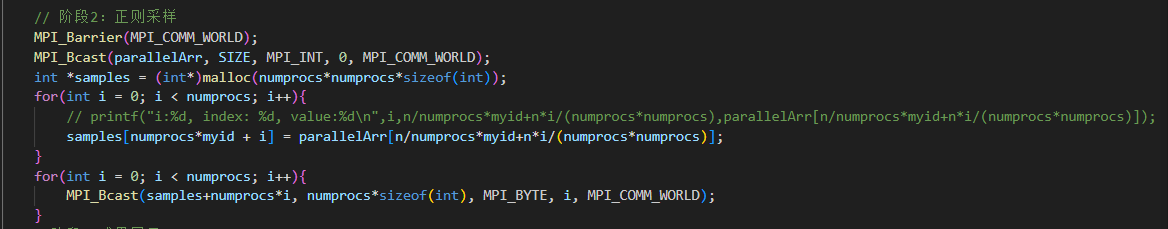


在主函数中，首先根据命令行参数指定的数组大小生成随机数组。（与3.1相同，代码不再展示）

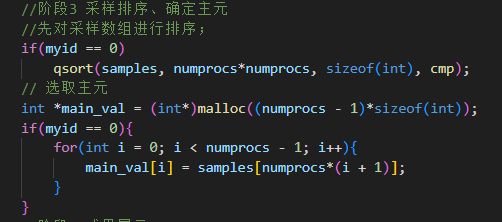
阶段1：无序序列的划分以及局部排序。使用localSize标记每个进程收到的数组数据，MPI\_Scatter方法将数据在进程中分发，再用MPI\_Gather方法将局部排好序的数组会聚到根进程中。



阶段2：正则采样，将局部排好序的数组进行正则采样，采样的数据存放在samples中，再使用MPI\_Bcast广播到所有的进程中。



阶段3：对采样数组进行排序并且间隔地选取主元。



阶段4：将这p-1个主元分发到所有的进程中。

1701070652902

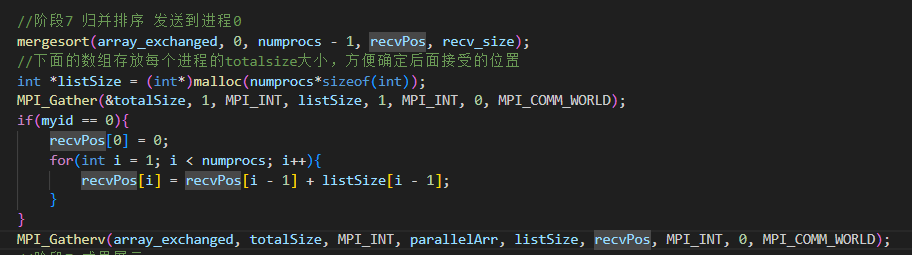
阶段5：根据这p-1个主元将有序序列划分成p段，这里定义了一个blocksize数组，每个处理器中都有这个数组，用于记录在该进程的局部数组中进行主元划分后p个小块，每个小块的元素个数，然后统一到一个blocksizes数组中，blocksizes[i]就表示第i//p号的进程需要传递到i%numproc号进程的元素个数。



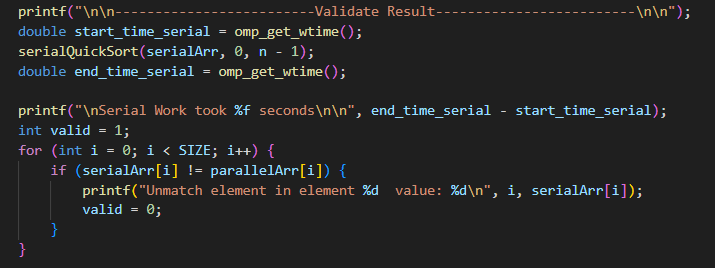
阶段6：全局交换，这也是代码中最核心的一部分，首先每个进程需要统计接收块的大小，这个通过调用MPI\_Alltoall方法传递blocksize数组中的元素来完成，然后统计每个进程最终处理的总元素数量，并把最后的数组存在array\_exchanged中，其中sendPos为每隔进程需要发送给其他进程的位置标记。然后调用MPI\_Barrier来设同步点，MPI\_Alltoallv来执行可变通信。



阶段7：归并排序并统一发送到进程0。其中listSize存放每个进程的总元素数，方便确定后面接收的位置。

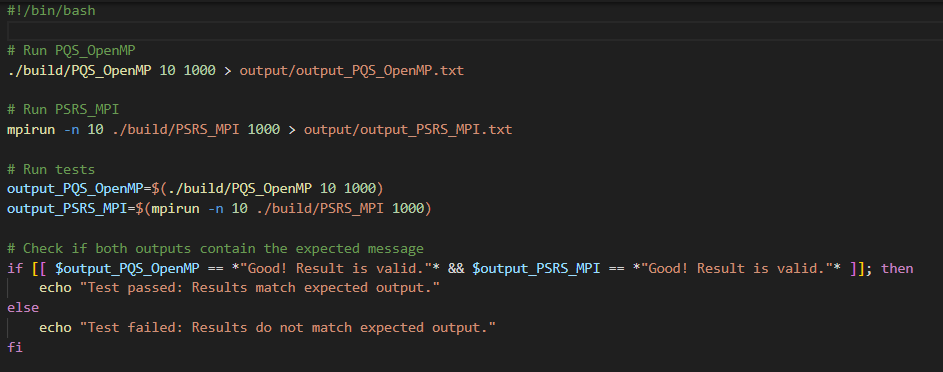


最后执行串行快速排序算法，比对结果与测试串行算法所需要时间以度量加速比。



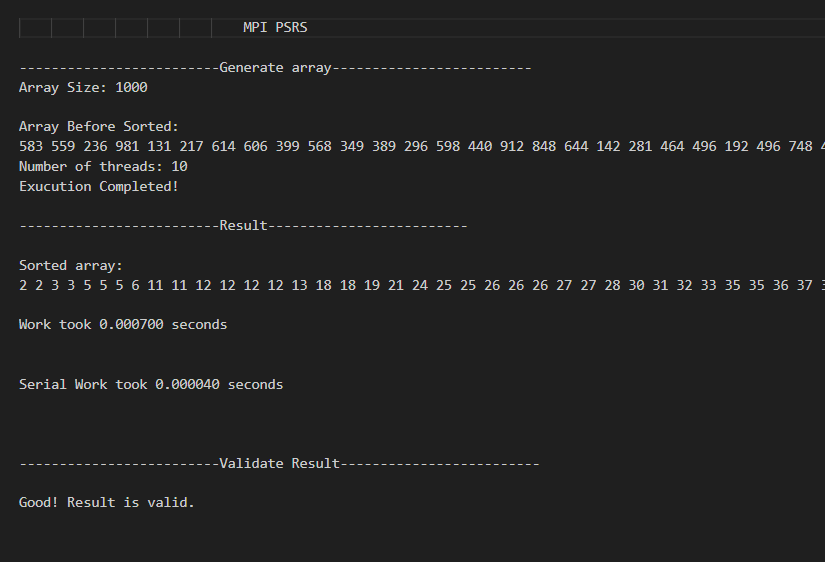
1. 实验测试
   1. 实验结果测试

运行test\_correct.sh脚本，得到输出符合预期，也可以在output目录下的txt文件中查看具体的输出情况。

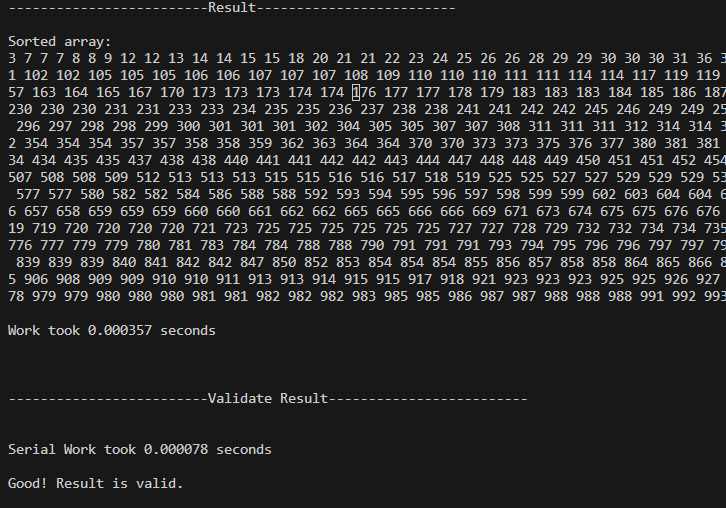


1701098123015

output\_PSRS\_MPI.txt文件如下图所示，输出结果符合预期。



也可以在终端中运行可执行文件。例如./build/PQS\_OpenMP 10 1000和mpirun -np 4 ./build/PSRS\_MPI 1000，得到的输出如下。当然也可以在examples中查看运行的样例。如./PQS\_OpenMP.sh，结果也将如下图所示。



* 1. 实验性能测试

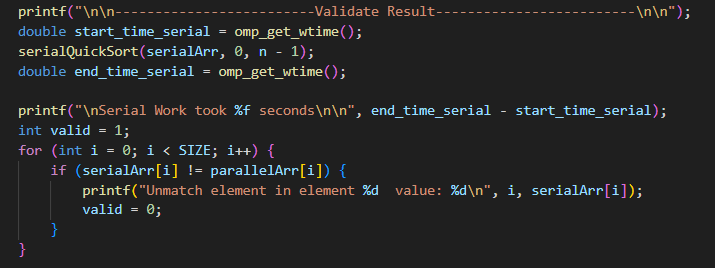
在实验的源代码PQS\_OpenMP.cpp和PSRS\_MPI.cpp中有串行快速排序的算法，我们可以用它来验证并行算法的正确性以及度量性能。

* + 1. 串行快速排序算法性能测试

在PQS\_OpenMP.cpp中指定不同的数据量大小，串行快速排序算法可以打印出对应的执行时间。如下表，对900000个元素的数组和1000000数组分别做测试的原因是在并行快速排序(OpenMP)算法中，指定数据量为1000000会出现内存不够的段错误，故只能指定数据量为900000，但并行正则采样排序(MPI)算法中可以指定数据量为1000000。

窗体底端

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | 1000 | 5000 | 10000 | 50000 | 100000 | 500000 | 900000 | 1000000 |
| time | 0.000062 | 0.00037 | 0.000846 | 0.007533 | 0.022737 | 0.417134 | 1.199372 | 1.477752 |

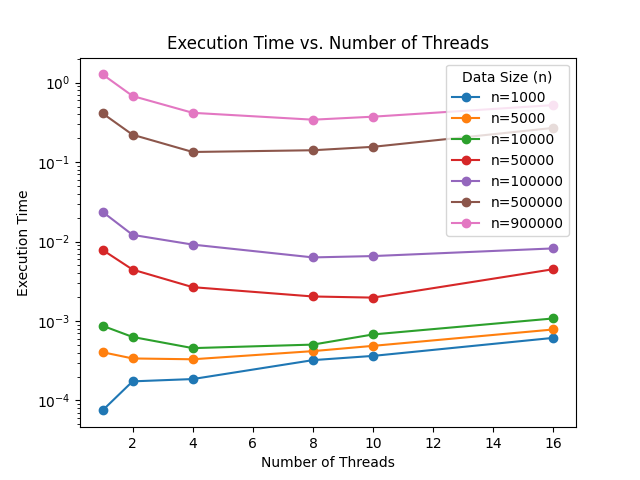


* + 1. 并行快速排序(OpenMP)的性能测试

在PQS\_OpenMP.cpp中指定不同的数据量大小和并行线程数测试时间如下表。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| n/thread | 1000 | 5000 | 10000 | 50000 | 100000 | 500000 | 900000 |
| 1 | 0.000075 | 0.000405 | 0.000872 | 0.007855 | 0.023664 | 0.41136 | 1.271363 |
| 2 | 0.000174 | 0.000338 | 0.00063 | 0.004438 | 0.012143 | 0.221102 | 0.679611 |
| 4 | 0.000186 | 0.00033 | 0.000456 | 0.002668 | 0.009162 | 0.134183 | 0.417525 |
| 8 | 0.000322 | 0.000418 | 0.000505 | 0.002037 | 0.00632 | 0.141105 | 0.342172 |
| 10 | 0.000364 | 0.000487 | 0.000678 | 0.001971 | 0.006571 | 0.155878 | 0.373318 |
| 16 | 0.000614 | 0.000783 | 0.001078 | 0.004493 | 0.008205 | 0.269768 | 0.522037 |

绘制图像如下（图像存在plot\_data目录中）：



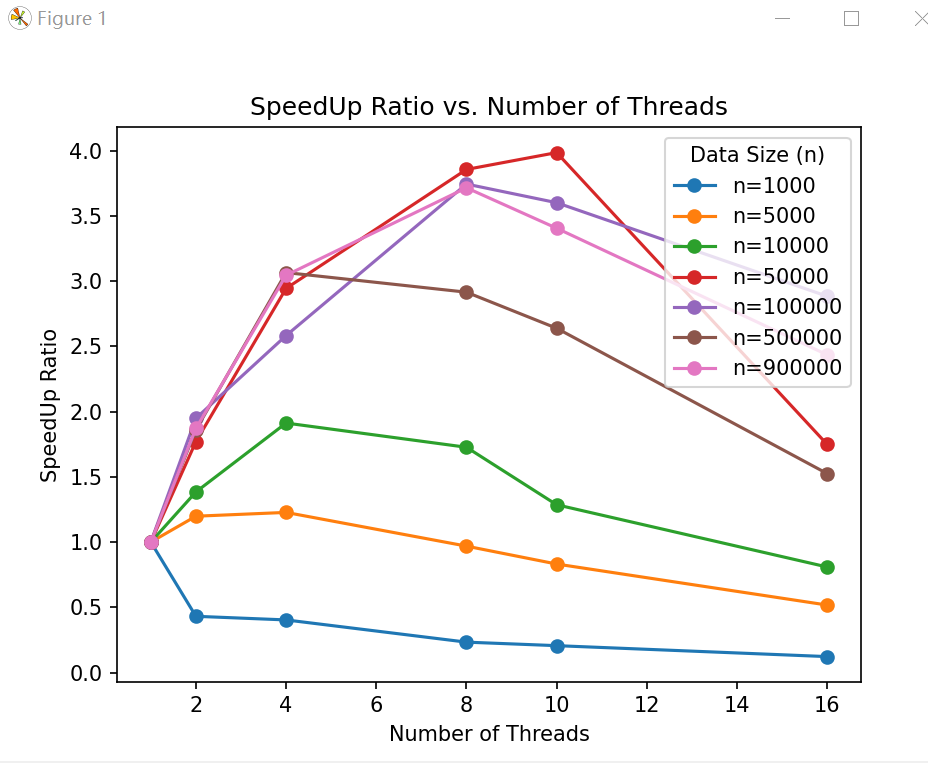
发现当线程数为1时执行时间与串行代码执行时间接近，可将其作为串行执行的基准时间，使用如下公式计算加速比：

SpeedUp =

得到加速比如下：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| n/thread | 1000 | 5000 | 10000 | 50000 | 100000 | 500000 | 900000 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 0.431 | 1.1982 | 1.3841 | 1.7699 | 1.9488 | 1.8605 | 1.8707 |
| 4 | 0.4032 | 1.2273 | 1.9123 | 2.9442 | 2.5828 | 3.0657 | 3.0450 |
| 8 | 0.2329 | 0.9689 | 1.7267 | 3.8562 | 3.7443 | 2.9153 | 3.7156 |
| 10 | 0.206 | 0.8316 | 1.2861 | 3.9853 | 3.6013 | 2.6390 | 3.4056 |
| 16 | 0.1221 | 0.5172 | 0.8089 | 1.7483 | 2.8841 | 1.5249 | 2.4354 |

绘制加速比与线程数、数据量的关系如下：



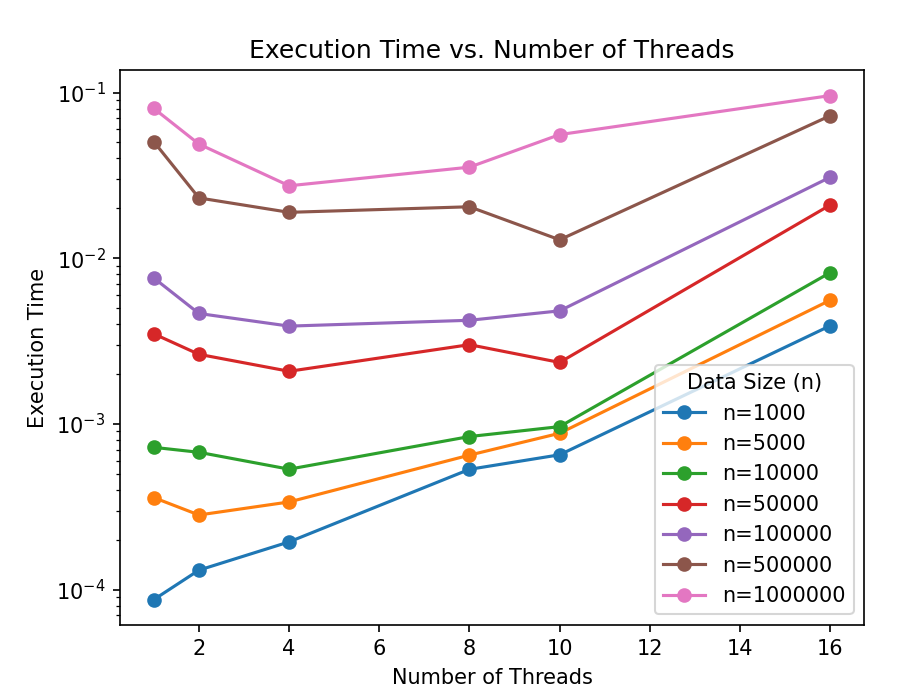
可以发现，加速比在线程数为4、8、10时都有可能达到最大值，当线程数超过10时会引起性能下降。

* + 1. 并行正则采样排序(MPI)的性能测试

在PSRS\_MPI.cpp中指定不同的数据量大小和并行线程数测试时间如下表。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| n/thread | 1000 | 5000 | 10000 | 50000 | 100000 | 500000 | 1000000 |
| 1 | 0.000087 | 0.000359 | 0.000722 | 0.003496 | 0.007597 | 0.05052 | 0.080214 |
| 2 | 0.000131 | 0.000282 | 0.000674 | 0.002631 | 0.00464 | 0.023125 | 0.048881 |
| 4 | 0.000194 | 0.000338 | 0.000533 | 0.002078 | 0.003893 | 0.018926 | 0.027372 |
| 8 | 0.000532 | 0.000648 | 0.000839 | 0.003008 | 0.004222 | 0.020449 | 0.035462 |
| 10 | 0.000651 | 0.000878 | 0.000963 | 0.00235 | 0.004813 | 0.012921 | 0.055755 |
| 16 | 0.003912 | 0.00558 | 0.008192 | 0.020949 | 0.030838 | 0.072493 | 0.095872 |

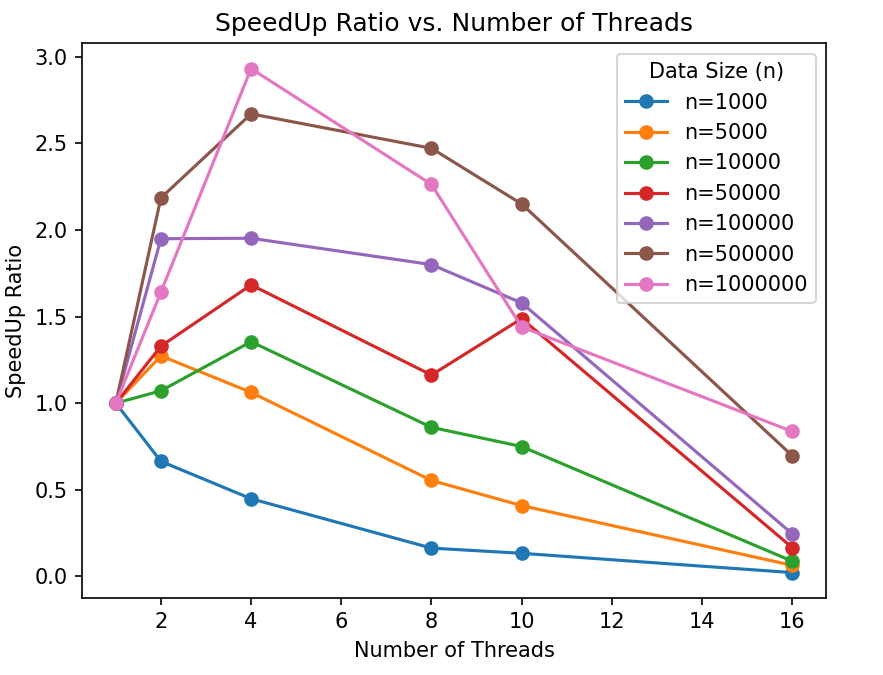
绘制图像如下（图像存在plot\_data目录中）：



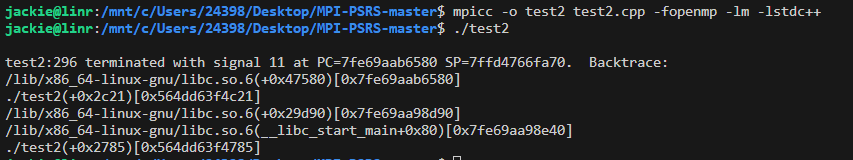
当线程数为1时正则采样排序算法退化为串行快速排序算法，但是执行时间会比普通的串行快速排序算法快一点，原因后面会进行分析，仍然将其作为串行执行的基准时间（确保使用的是同一个编译器与算法），使用同样的公式计算加速比。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| n/thread | 1000 | 5000 | 10000 | 50000 | 100000 | 500000 | 1000000 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 0.6641 | 1.273 | 1.0712 | 1.3288 | 1.9488 | 2.1846 | 1.641 |
| 4 | 0.4485 | 1.0621 | 1.3546 | 1.6824 | 1.9515 | 2.6693 | 2.9305 |
| 8 | 0.1635 | 0.554 | 0.8605 | 1.1622 | 1.7994 | 2.4705 | 2.262 |
| 10 | 0.1336 | 0.4089 | 0.7497 | 1.4877 | 1.5784 | 2.1495 | 1.4397 |
| 16 | 0.0222 | 0.0643 | 0.0881 | 0.1669 | 0.2464 | 0.6969 | 0.8367 |

绘制加速比与线程数、数据量的关系如下：



遇到的问题：

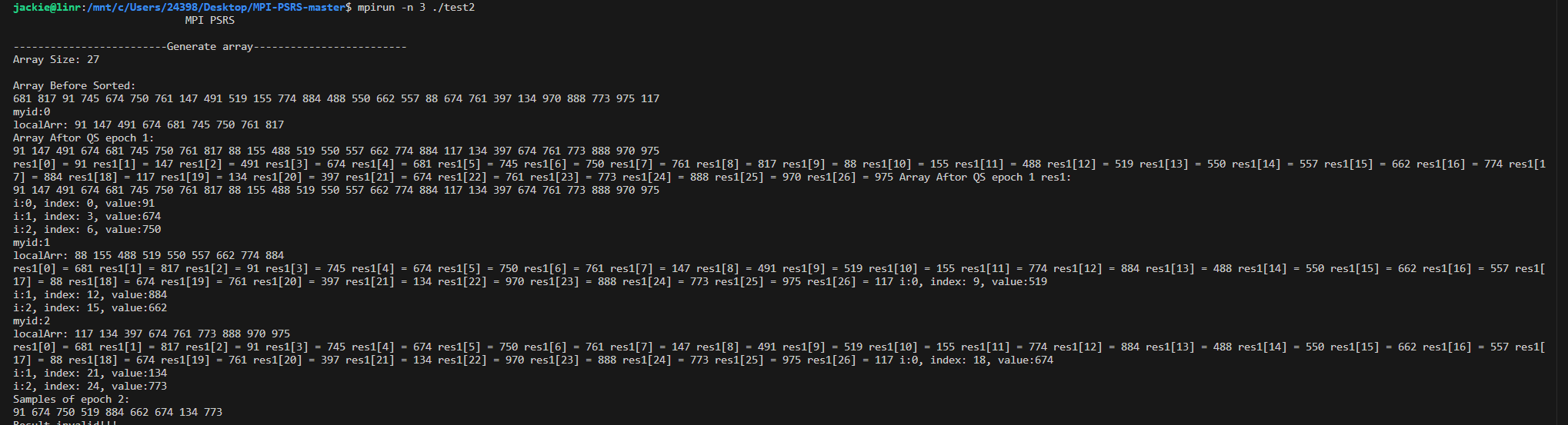


Linux(wsl):

#mpicc -o test2 test2.cpp -fopenmp -lm -lstdc++

make

mpirun -n 10 ./test2



进程间没有同步数组parallelArr的值，导致直接赋值给sample产生问题。

仍有问题，当numproc不整除SIZE，如何划分

全局交换的算法也有问题