Introduction to HPC
HW5 Report

姓名: 任一

学号:2018011423

ry18@mails.tsinghua.edu.cn

2020 年 4 月 24 日

实验环境	
操作系统:	Windows10 家庭版 18362.72 Windows Subsystem for Linux
gcc 版本:	gcc version 7.5.0
集群处理器：	Intel(R) Xeon(R) E5-2680 v3
集群核心数：	24

1 Exercise5.4

各规约操作符与其初始化的变量值如下表：

表 1: 规约运算符与其对应的变量初始值表

Operator	Initial Value
&&	1
||	0
&	$111...111_2$
|	0
^	0

2 Exercise5.5

2.1 1

串行相加时，当完成最后一次相加后，寄存器中的数值是 $1.008e+03$, 当这个数字被存储到内存中时，该数值被四舍五入为 3 位十进制有效数字，即 $sum = 1.01e+03$. 因此输出的值为 1010.0.

2.2 2

使用 2 线程并行相加时, 0 号线程负责前两个数的相加，局部和为 $4.00e+00$, 1 号线程负责后两个数的相加，局部和为 $1.00e+03$(此处发生了四舍五入). 这两个线程局部和相加，得到的结果为 $1.00+03$(此处发生了四舍五入). 因此输出的值为 1000.0.

3 PA1

本题中我主要使用了 parallel for 语句，使得每一个 Process 中都采用了 OMP 的并行方式。

本题在集群上的目录为/home/2018011423/HW5/PA1，在该文件夹下输入 make 即可编译，make run 即可运行。

3.1 并行串行结果之差二范数

在所有测试中，我的并行结果与串行结果之差的二范数都是 0，这体现出了我的并行程序的正确性。

3.2 矩阵乘法与二范数计算时间分析

3.2.1 矩阵乘法计算时间

表 2: 并行计算时间表

parallel calc time				
		MATRIX ORDER		
	Processors	2160	4320	7200
10 THREADS	4	0.012486	0.017767	0.028428
	9	0.027715	0.027137	0.038875
	16	0.045799	0.04003	0.053367
20 THREADS	4	0.021975	0.02776	0.038028
	9	0.050516	0.050927	0.062229
	16	0.086376	0.082736	0.092697

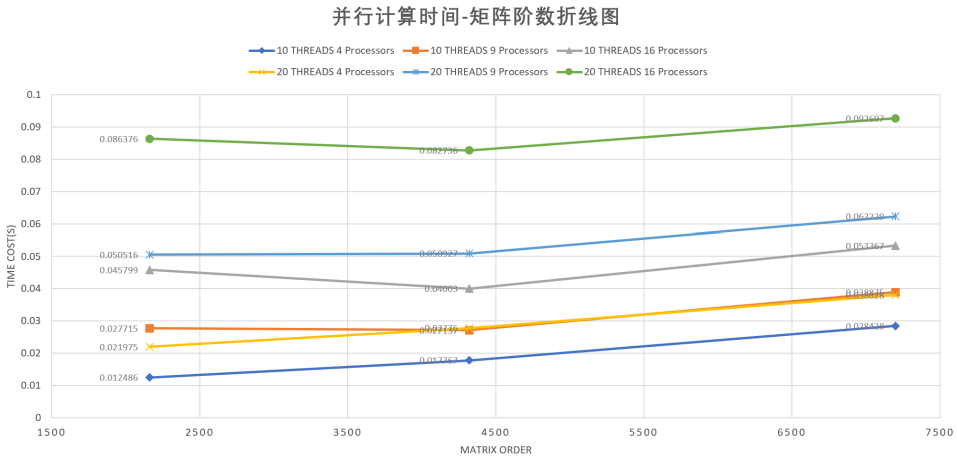

图 1: 并行计算时间与矩阵阶数折线图

3.2.2 二范数计算时间

表 3: 二范数计算时间表

	L2 Norm Calc Time		
	2160	4320	7200
SERIAL	0.000007	0.000013	0.000021
10 THREADS	0.000058	0.00006	0.00008
20 THREADS	0.000085	0.002227	0.006091

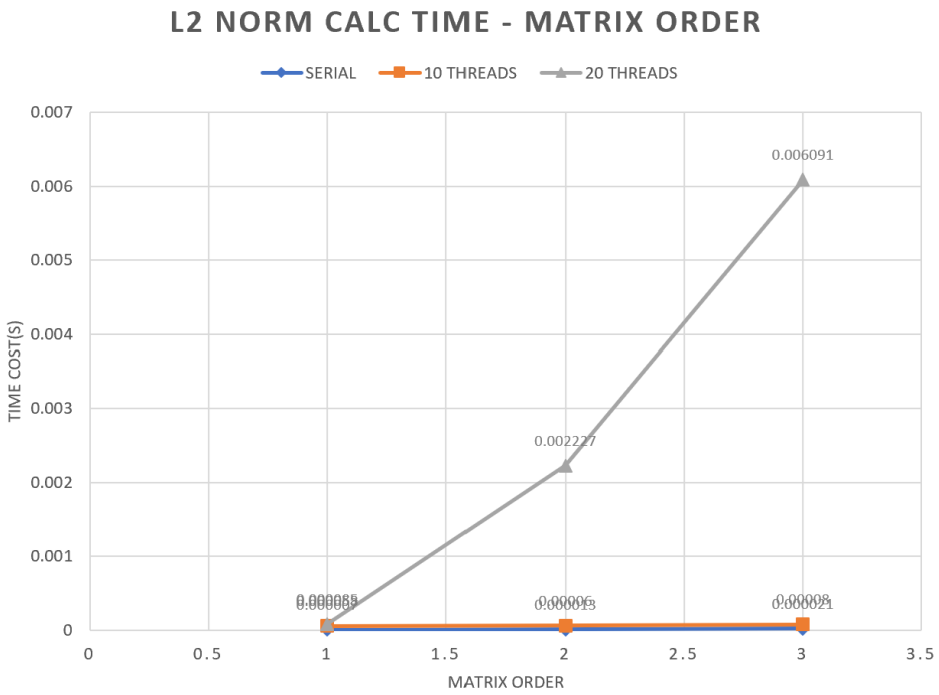

图 2: 二范数计算时间折线图

从图中可以明显看出，多线程对于二范数计算的加速并不明显。原因可能是二范数计算范围较小，线程的开辟和回收时间成本可能大于并行带来的时间收益。

3.3 并行总时间、计算时间、通信时间、加速比、并行效率

3.3.1 并行总时间

表 4: 并行总时间表

parallel all time				
		MATRIX ORDER		
	Processors	2160	4320	7200
10 THREADS	4	0.125707	0.343842	0.822637
	9	0.134293	0.368953	0.905315
	16	0.164941	0.39197	0.942639
20 THREADS	4	0.135145	0.36316	0.884623
	9	0.167472	0.386904	0.9313
	16	0.204671	0.432752	0.961524

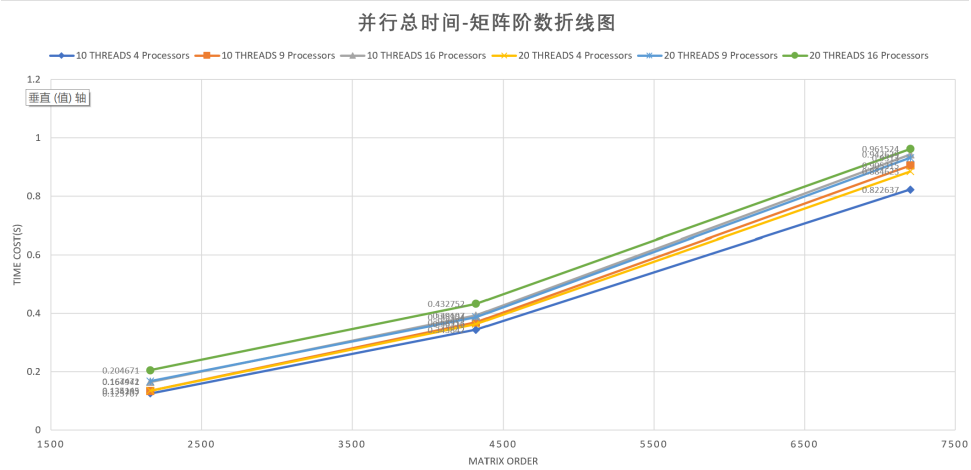

图 3: 并行总时间折线图

3.3.2 并行计算时间

并行计算时间的图表已经在 4.1 中分析了，在此不再重复。

3.3.3 并行通信时间

表 5: 并行通信时间表

parallel distribution time				
		MATRIX ORDER		
	Processors	2160	4320	7200
10 THREADS	4	0.113221	0.326075	0.794209
	9	0.106578	0.341816	0.866440
	16	0.119142	0.351940	0.889272
20 THREADS	4	0.11317	0.3354	0.846595
	9	0.116956	0.335977	0.869071
	16	0.118295	0.350016	0.868827

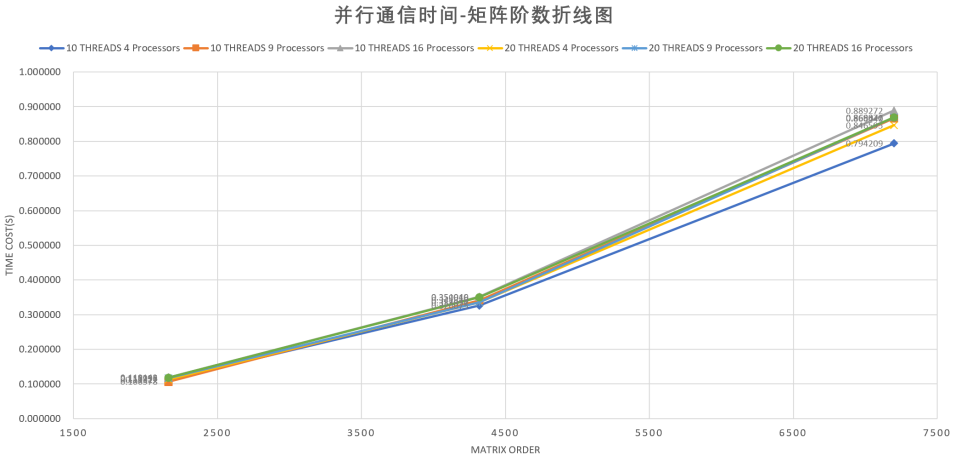

图 4: 并行通信时间折线图

3.3.4 并行计算时间加速比

表 6: 并行加速比表

parallel speed up ratio				
		MATRIX ORDER		
	Processors	2160	4320	7200
10 THREADS	4	2.607801	3.233748	5.584881
	9	1.051633	2.385046	4.616875
	16	0.353610	1.619435	3.384039
20 THREADS	4	1.403868	3.505728	5.378852
	9	0.508611	1.694916	2.885632
	16	0.188664	0.918923	2.113315

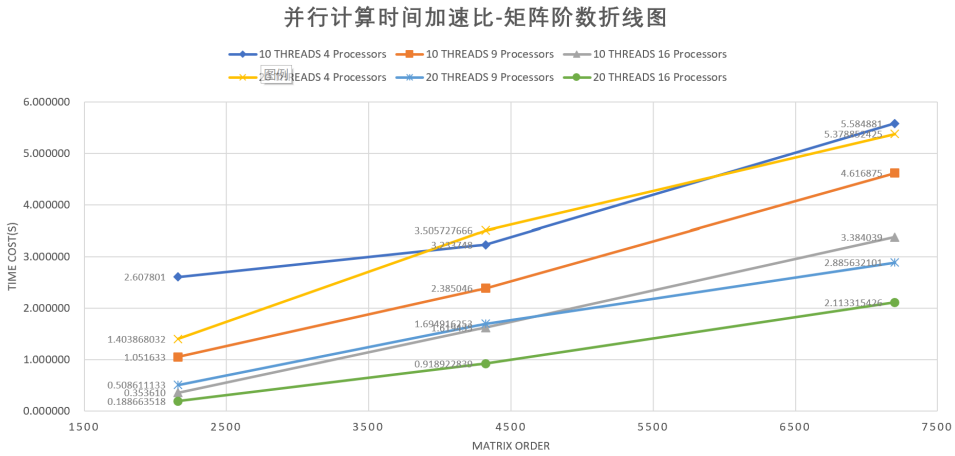

图 5: 并行通信时间折线图

3.3.5 并行计算时间效率

表 7: 并行计算时间效率表

efficiency				
		MATRIX ORDER		
	Processors	2160	4320	7200
10 THREADS	4	0.065195	0.080844	0.139622
	9	0.011685	0.026501	0.051299
	16	0.00221	0.010121	0.02115
20 THREADS	4	0.017548	0.043822	0.067236
	9	0.002826	0.009416	0.016031
	16	0.00059	0.002872	0.006604

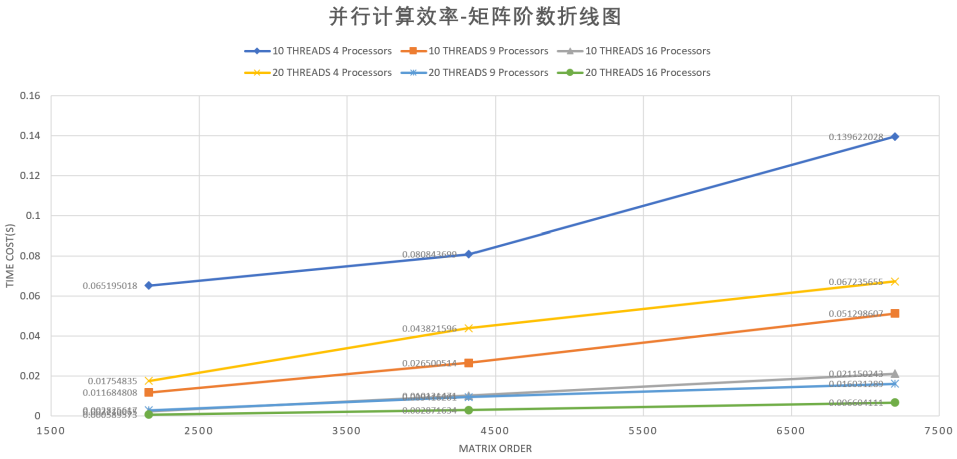

图 6: 并行效率折线图

3.4 两种算法对比

在矩阵阶数为 7200，处理器数为 9 的情况下，我对比了此前实现的算法与加入 OMP 后的算法。其中 Thread Count 为 1 的数据即为此前算法，Thread Count 不为 1 的是加入 OMP 后的算法。

表 8: 两种算法并行计算时间表

Thread Count	PARALLEL CALC TIME
1	0.020316
10	0.038875
20	0.062229

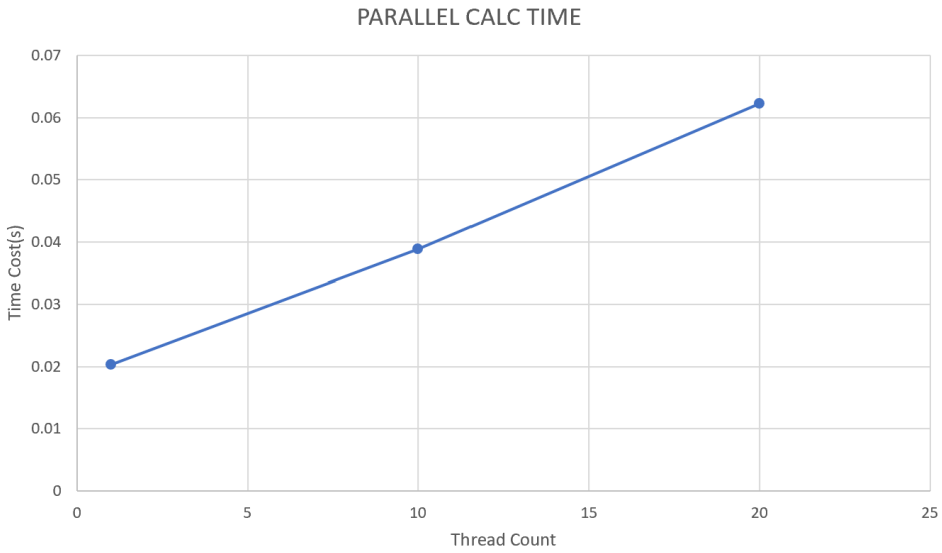

图 7: 两种算法并行计算时间折线图

表 9: 两种算法并行总时间表

Thread Count	PARALLEL ALL TIME
1	0.835079
10	0.905315
20	0.9313

可能是由于测试的矩阵规模不够大，OMP 在每个线程上并不能得到理想的加速结果，反而比不采用 OMP 的版本用时更多 (无论是计算时间还是总时间). 因此这也提醒了我，在实验规模不大时，无须采用过多的并行技术。

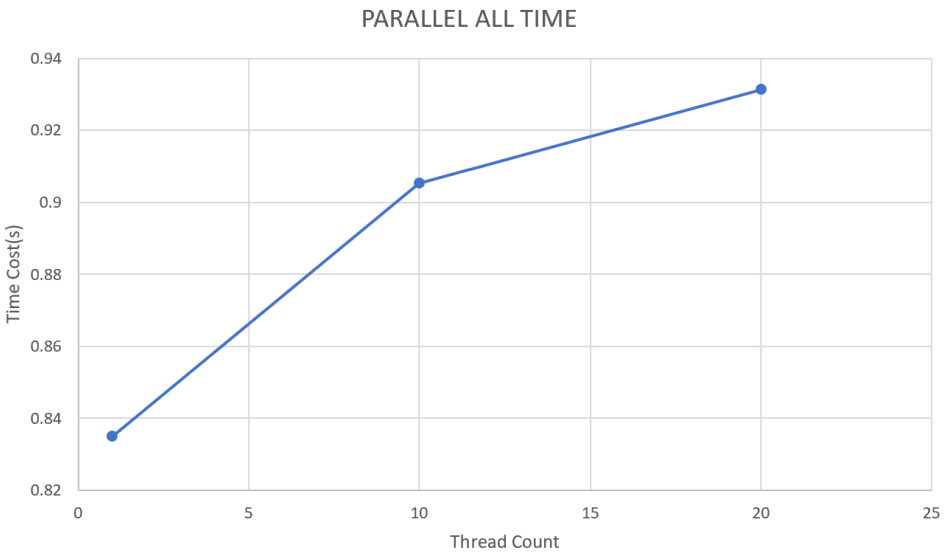

图 8: 两种算法并行总时间折线图

4 PA2

本题在集群上的目录为/home/2018011423/HW5/PA2,在该文件夹下输入 make 即可编译,make run 即可运行。

4.1

私有的变量有 i, j, count. 共享的变量有 a, n, temp.

4.2

没有循环依赖。每个线程之间彼此都使用私有的 i, j, count，对共享的 temp 数组的写操作也是彼此不重叠的，对 a 和 n 只存在读操作，因此不存在依赖和冲突。

4.3

可以并行化 memcpy 的调用。但是不一定能提高效率，因为 memcpy 自身效率已经很高，开辟新线程还会带来额外的时间开销，因此不一定能够提高效率。

4.4

程序的实现思路如下：我使用了题目中给出的计数排序的实现方式, 并利用 parallel for 语句对其加以改造使其并行化。

4.5 分析不同数据规模下，各排序算法的性能

在这一部分，我将对比不同数据规模下，串行计数排序、并行计数排序、快速排序的性能。此处的并行计数排序，我采用了 10 线程和默认调度方式。分析图表如下：

表 10: 各排序算法性能表

array size	100	500	1000	5000	10000	50000	100000
Serial Count Sort Time(s)	0.000120	0.003107	0.013448	0.202669	0.604671	13.615840	53.869850
Parallel Count Sort Time(s)	0.000254	0.000705	0.002144	0.038180	0.116571	1.759016	6.806380
QuickSort Time(s)	0.000020	0.000216	0.000296	0.000545	0.001082	0.010681	0.022842

表 11: 并行计数排序相较串行计数排序加速比

array size	100	500	1000	5000	10000	50000	100000
SpeedUpRatio	0.472440945	4.407092199	6.27238806	5.308250393	5.187147747	7.740600427	7.914610997

从图表中可以看出，相较串行计数排序，并行计数排序在数据规模较大时，表现出了较好的加速性能。但是由于题中所给的计数排序时间复杂度为 $O(n^2)$, 而快速排序时间复杂度为 $O(nlogn)$, 其中 n 为待排序的数组长度，因此两种计数排序的耗时都明显高于快速排序。

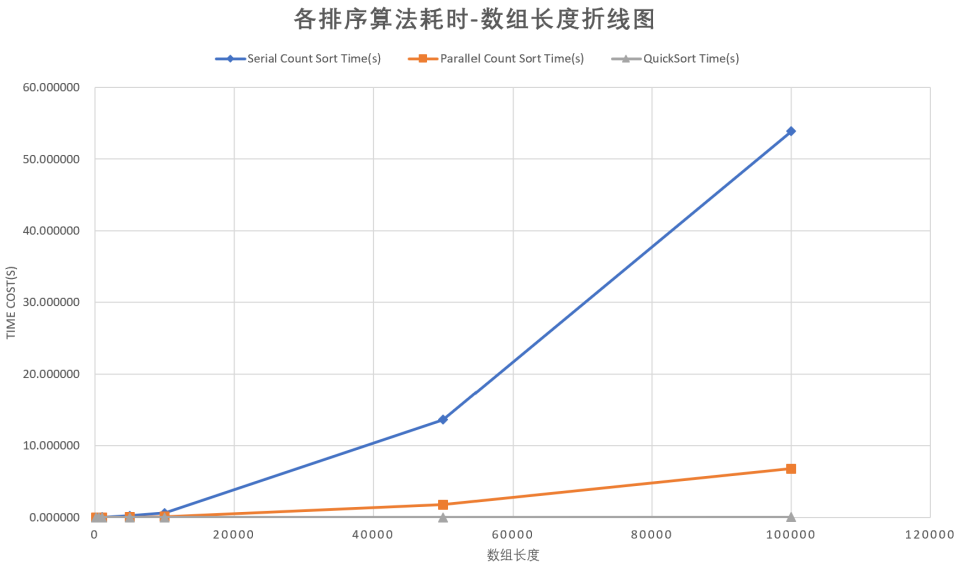

图 9: 各排序算法耗时-数组长度折线图

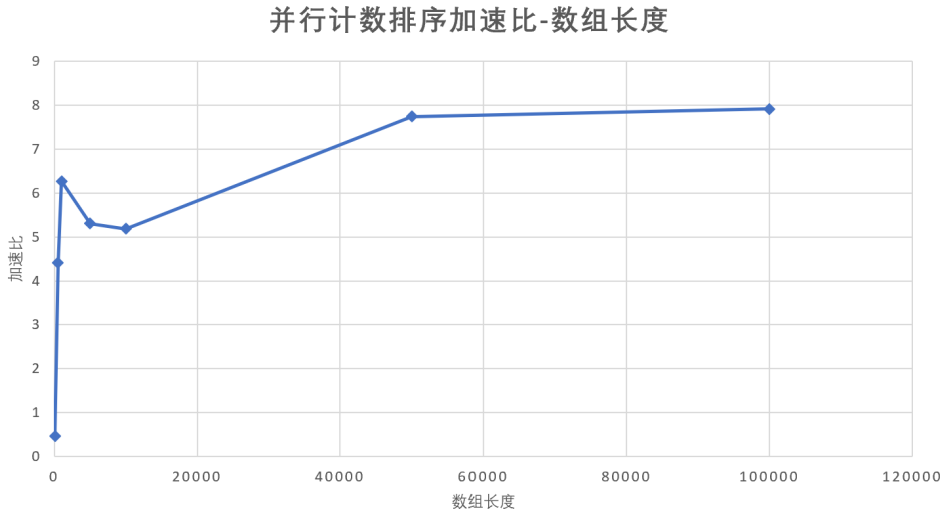

图 10: 并行计数排序加速比-数组长度折线图

4.6 不同调度策略下，程序运行效率

表 12: 不同调度策略，程序耗时表

	static					Dynamic					Guided				
Chunk Size	1	10	50	100	500	1	10	50	100	500	1	10	50	100	500
Time Cost(s)	0.118	0.115	0.117	0.117	0.117	0.109	0.108	0.110	0.114	0.118	0.108	0.108	0.110	0.111	0.122

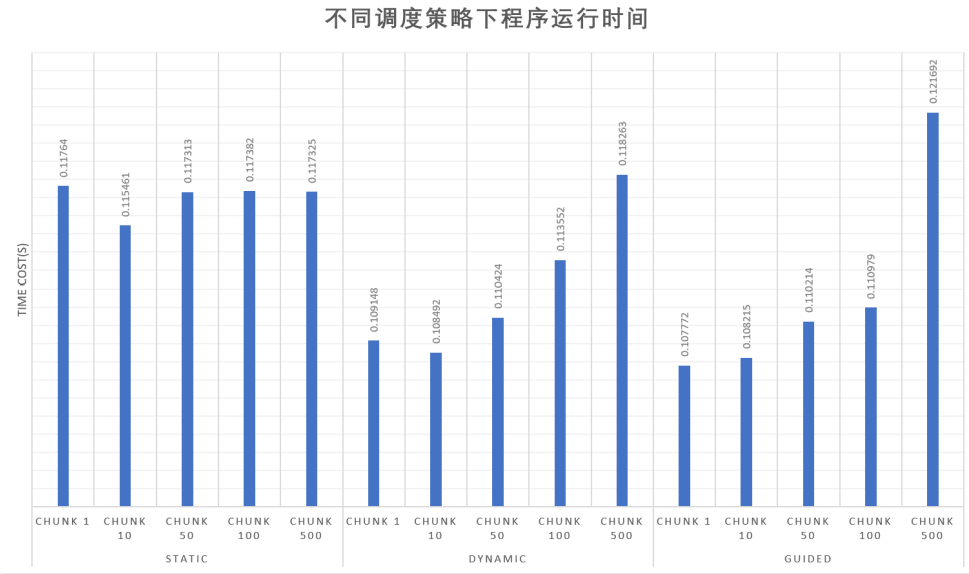

图 11: 不同调度策略下，程序运行时间柱状图

理论上讲，在本题的场景中，分配到数组中的元素进行计数排序的时间成本应该不会有很大区别。因此使用 Static 调度会是一个好的选择，Dynamic 调度可能会因为动态调度而比 Static 调度慢一些。

但是从实际的实验结果来看，Dynamic 调度耗时比 Static 调度略少，并且 ChunkSize 较小时运行速度更快。在 Static 调度方法中，ChunkSize 对耗时影响不大。而在 Guided 调度中，耗时随着 ChunkSize 的增大而增加。