浙江大学

本科生课程作业

MPI 并行排序

课程名称:	并行计算与多核编程
姓名:	陈卓
学院:	计算机科学与技术学院
专业:	计算机科学与技术
学号:	3170101214
指导老师:	楼学庆

2020 年 5 月 2 日

1 算法

本次实验使用 MPI 实现了三种并行排序算法。

1.1 并行归并排序

本实验并行归并排序的想法十分平凡，即：

i) P_0 将原数据均分到不同进程 P_i，$0 \le i < p$；

ii) P_i 各自进行串行快速排序；

iii) 将排序后的数据发送到 P_0；

iv) P_0 对这些有序数列进行归并。

1.2 并行奇偶排序

奇偶排序是冒泡排序的一个变种。串行奇偶排序的算法步骤如下：

i) **偶阶段**：比较并交换 $(a[0], a[1]), (a[2], a[3]), \ldots$；

ii) **奇阶段**：比较并交换 $(a[1], a[2]), (a[3], a[4]), \ldots$；

iii) 重复进行 $0, 1, \ldots, n$ 阶段。

奇偶排序算法的并行化相对容易，因为它在每个阶段的比较交换是可并行的，但由于不可能提供 n 个处理器进行排序，因此需要将串行算法中比较交换的最小单位更改为一个处理器 P_i，其中每个处理器处理 $\frac{n}{p}$ 的数据。并行算法步骤如下：

i) P_0 将原数据均分到不同进程 P_i，$0 \le i < p$；

ii) **偶阶段**：比较并交换 $(P_0, P_1), (P_2, P_3), \ldots$；

iii) **奇阶段**：比较并交换 $(P_1, P_2), (P_3, P_4), \ldots$；

iv) **比较交换**：将 P_i, P_{i+1} 的数据归并，P_i/P_{i+1} 取归并后较小/大（大/小）的一半；

v) 重复进行 $0, 1, \ldots, n$ 阶段。

1.3 PSRS 排序

PSRS 排序在上次实验中已经通过 C++ 线程库实现，这次采用 MPI 实现，算法步骤如下：

i) P_0 将原数据均分到不同进程 P_i，$0 \le i < p$；

ii) P_i 将各自数据排序，并选择 p 各**样本**（第 $0, \frac{n}{p^2}, \ldots$ 个元素）发送给 P_0；

iii) P_0 对 p 各处理器的样本归并，并选择 $p-1$ 个元素作为**主元**，广播给所有处理器；

iv) P_i 根据 $p-1$ 个主元将各自数据分为 p 段，并将第 i 段发送给 P_i；

v) P_i 对数据段归并，并发送给 P_0。

2 实现

本实验选用的 MPI 实现是 MPICH，版本 3.3，环境 macOS 10.15.3，C 编译器 Apple clang version 11.0.3。

三种算法通过 MPI 接口实现所属算法步骤，实现代码详见 `merge.c`、`oddeven.c`、`psrs.c` 文件，或在 GitHub 中查看。

3 测试

本实验测试分别测试比较以上三种并行排序算法与串行快速排序，测试平台 2.3 GHz 双核 Intel Core i5。测试脚本见 `test.py`。测试结果如下。

表 3.1: 并行归并算法

data size	2	4	6	8
1000	0.318	0.4144	0.2014	0.0637
2000	0.6028	0.5869	0.2145	0.1704
4000	0.9279	0.8163	0.5669	0.2439
8000	0.936	0.7147	0.6749	0.1723
10000	1.1766	0.8111	0.5276	0.3672
20000	1.1211	1.4503	1.1332	0.4771
40000	1.0878	1.2525	1.2845	0.6683
80000	1.5046	1.3068	1.3158	0.9683
100000	1.4502	1.2389	1.2877	0.9347
200000	1.5111	1.2989	1.1516	0.8237
400000	1.711	1.9033	1.4051	1.2541
800000	1.5593	1.9909	1.7643	1.4167
1000000	1.6864	1.9013	1.5149	1.495
2000000	1.6305	1.9528	1.8392	1.3802
4000000	1.6983	2.1558	1.7956	1.0306
8000000	1.7761	2.2193	1.8555	1.4662
10000000	1.7989	2.2553	1.8771	1.5491

表格中，每行代表不同的数据量大小，每列代表 MPI 进程数量，每个元素代表加速比[1]。

[1]取三次平均。

表 3.2: Odd-even 和 PSRS

(a) Odd-even

data size	2	4	6	8
1000	0.628	0.2763	0.1673	0.1116
2000	0.58	0.7009	0.3847	0.1702
4000	1.1253	0.4046	0.5251	0.1501
8000	1.0995	0.9898	0.6476	0.4847
10000	1.1438	1.6084	0.7593	0.6653
20000	1.6753	1.1987	1.2355	0.6608
40000	0.9572	1.5913	1.0018	1.088
80000	1.3408	1.0079	1.9049	1.4625
100000	1.1573	1.2169	0.6721	1.2137
200000	1.4405	1.6818	1.3073	0.9121
400000	1.6733	2.21	1.8663	1.4839
800000	1.6691	2.2688	2.3497	1.9017
1000000	1.7499	2.3117	2.0255	2.0206
2000000	1.6462	2.5063	2.2737	2.5497
4000000	1.7047	2.5242	2.5939	2.3107
8000000	1.8165	2.67	2.182	2.5189
10000000	1.8473	2.6753	2.5478	2.4851

(b) PSRS

data size	2	4	6	8
1000	0.2361	0.1523	0.0685	0.0181
2000	0.4711	0.1109	0.1176	0.0139
4000	0.7284	0.2762	0.4648	0.156
8000	0.8773	0.4807	0.3726	0.4408
10000	0.8842	0.9156	0.455	0.3384
20000	0.8289	0.8443	0.6542	0.5443
40000	1.0907	1.0901	0.7088	0.5826
80000	1.3272	0.727	1.3212	0.9155
100000	1.2827	0.6154	0.7847	0.879
200000	1.5992	0.9787	1.089	0.8136
400000	1.5221	1.732	1.3438	1.1163
800000	1.4456	1.6909	1.5921	1.3406
1000000	1.3622	1.9132	1.641	1.3945
2000000	1.44	1.9633	1.705	1.3566
4000000	1.4199	1.9663	1.6449	1.3913
8000000	1.4382	2.0683	1.7433	1.43
10000000	1.6067	2.076	1.8918	1.5807

4 讨论

根据测试结果针对不同算法横向比较，我们可以看出并行奇偶排序的加速效果最好，这是因为并行奇偶排序的并行度较高，所有处理器在每个阶段都参与归并运算。而并行归并排序只通过 P_0 归并，并行度较低。同时，PSRS 算法在处理器广播片段时，由于预先不能预测片段的长度，因此在实现上先广播 p 个片段的长度，再广播 p 个片段，造成性能的损失。

由此我们可以提出针对这两种算法的改进方法。并行归并排序中，可以通过 log 递减的方式归并来提高并行度，即第一轮 $(P_0, P_1), (P_2, P_3), \ldots$ 两两归并，第二轮 $(P_0, P_2), (P_4, P_6), \ldots$ 两两归并，这样可以提高并行度。PSRS 排序中，可以为每个片段申请 $\frac{n}{p}$ 大小的空间，在广播片段时可以不用顾虑片段长度问题。

针对处理器数量比较，以并行奇偶排序算法为例，结果趋近于 $p = 4 > p = 6 > p = 8 > p = 2$，4 进程的效率最高是因为测试机器是 4 线程 CPU。其它算法由于 MPI 进程开销暂时无法准确评估，因此顺序略有变化，但是 $p = 4$ 效率最高。

纵向比较，$p = 2$ 时三种算法的加速比随着数据量的增加，稳定在 $1.6 \sim 1.8$ 之间，有一定的可扩放性。