## Parallel Sorting by Regular Sampling

浙工大 姜楠

### 实验原理：

设有n个数据，P个处理器，以及均匀分布在P个处理器上的n个数据。 则PSRS算法可描述如下：

**输入：**n个待排序的数据，均匀地分布在P个处理器上

**输出：**分布在各个处理器上，得到全局有序的数据序列

**Begin**

(1) 每个处理器将自己的n/P个数据用串行快速排序(Quicksort)，得到一个排好序的

序列；

(2) 每个处理器从排好序的序列中选取第w，2w，3w，…，(P-1)w个共P-1个数据作为

代表元素，其中w=n/P2；

(3) 每个处理器将选好的代表元素送到处理器P0中，并将送来的P段有序的数据序列

做P路归并，再选择排序后的第P-1，2(P-1)，…，(P-1)(P-1)个共P-1个主元；

(4) 处理器P0将这P-1个主元播送到所有处理器中；

(5) 每个处理器根据上步送来的P-1个主元把自己的n/P个数据分成P段，记为处

理器Pi的第j+1段，其中i=0，…，P-1，j=0，…，P-1；

(6) 每个处理器送它的第i+1段给处理器Pi，从而使得第i个处理器含有所有处理器的

第i段数据(i=0，…，P-1)；

(7) 每个处理器再通过P路归并排序将上一步的到的数据排序；从而这n个数据便是有

序的。

### 实验结果：

输入：

/usr/mpi/intel/openmpi-1.4.3/bin/mpicc -o psra psra.c

/opt/openmpi/bin/mpirun -np 11 ./psra

可以看到：结果是成功的，单数还存在些许不足，尚待完善

进程i发来数据<进程(i+1)发来数据 i=1,2,3...9

### 实验体会：

花了整整两天时间调试，测试，逐步完善代码，感受颇深。当最后结果出来时，还是很欣慰的。虽然网上能找到很多版本的PSRA算法的代码，但是都是不忍卒读，代码风格较差，为此，故自己参照他们的大致思路，完成了一份思路较为清晰的代码，花费了很多精力在MPI\_Send和MPI\_Recv这两个通信函数上面，解决了一些较为常见的问题，对并行编程有了更为深刻的体验。

这次由于初次尝试编写较为功能完整的作业，主要时间都花费在调试上面，作业迟交，还望老师谅解，谢谢！

### 附录

试验结果：

|  |
| --- |
| tmp[0]: 1 tmp[1]: 5 tmp[2]: 6 tmp[3]: 7 tmp[4]: 8 tmp[5]: 9 tmp[6]: 9 tmp[7]: 11 tmp[8]: 11 tmp[9]: 11 tmp[10]: 12 tmp[11]: 1 tmp[12]: 5 tmp[13]: 6 tmp[14]: 7 tmp[15]: 8 tmp[16]: 9 tmp[17]: 9 tmp[18]: 11 tmp[19]: 11 tmp[20]: 11 tmp[21]: 12 tmp[22]: 13 tmp[23]: 13 tmp[24]: 14 tmp[25]: 9 tmp[26]: 11 tmp[27]: 11 tmp[28]: 11 tmp[29]: 12 tmp[30]: 13 tmp[31]: 13 tmp[32]: 14 tmp[33]: 17 tmp[34]: 18 tmp[35]: 19 tmp[36]: 9 tmp[37]: 11 tmp[38]: 11 tmp[39]: 11 tmp[40]: 13 tmp[41]: 13 tmp[42]: 14 tmp[43]: 19 tmp[44]: 19 tmp[45]: 20 tmp[46]: 20 tmp[47]: 25 tmp[48]: 27 tmp[49]: 11 tmp[50]: 13 tmp[51]: 13 tmp[52]: 19 tmp[53]: 20 tmp[54]: 27 tmp[55]: 9 tmp[56]: 11 tmp[57]: 13 tmp[58]: 20 tmp[59]: 27 tmp[60]: 28 tmp[61]: 29 tmp[62]: 37 tmp[63]: 20 tmp[64]: 27 tmp[65]: 28 tmp[66]: 29 tmp[67]: 37 tmp[68]: 39 tmp[69]: 41 tmp[70]: 43 tmp[71]: 59 tmp[72]: 61 tmp[73]: 1000 tmp[74]: 1000 tmp[75]: 1000 tmp[76]: 1000 tmp[77]: 1000 tmp[78]: 1000 tmp[79]: 1000 tmp[80]: 1000 tmp[81]: 1000 tmp[82]: 1000 tmp[83]: 1000  I got the final result!  tmp[0]: 66 tmp[1]: 66 tmp[2]: 70 tmp[3]: 71 tmp[4]: 71 tmp[5]: 74 tmp[6]: 75 tmp[7]: 75 tmp[8]: 76 tmp[9]: 79 tmp[10]: 80 tmp[11]: 80 tmp[12]: 80 tmp[13]: 81 tmp[14]: 66 tmp[15]: 71 tmp[16]: 71 tmp[17]: 75 tmp[18]: 75 tmp[19]: 76 tmp[20]: 79 tmp[21]: 80 tmp[22]: 80 tmp[23]: 80 tmp[24]: 81 tmp[25]: 66 tmp[26]: 70 tmp[27]: 71 tmp[28]: 71 tmp[29]: 75 tmp[30]: 75 tmp[31]: 76 tmp[32]: 79 tmp[33]: 80 tmp[34]: 80 tmp[35]: 80 tmp[36]: 81 tmp[37]: 66 tmp[38]: 70 tmp[39]: 71 tmp[40]: 71 tmp[41]: 75 tmp[42]: 75 tmp[43]: 76 tmp[44]: 79 tmp[45]: 80 tmp[46]: 80 tmp[47]: 80 tmp[48]: 81 tmp[49]: 81 tmp[50]: 82 tmp[51]: 71 tmp[52]: 75 tmp[53]: 75 tmp[54]: 76 tmp[55]: 79 tmp[56]: 80 tmp[57]: 80 tmp[58]: 81 tmp[59]: 81 tmp[60]: 82 tmp[61]: 82 tmp[62]: 84 tmp[63]: 71 tmp[64]: 75 tmp[65]: 75 tmp[66]: 76 tmp[67]: 79 tmp[68]: 80 tmp[69]: 80 tmp[70]: 82 tmp[71]: 82 tmp[72]: 84 tmp[73]: 85 tmp[74]: 87 tmp[75]: 90 tmp[76]: 90 tmp[77]: 99 tmp[78]: 100 tmp[79]: 101 tmp[80]: 101 tmp[81]: 90 tmp[82]: 90 tmp[83]: 99 tmp[84]: 101 tmp[85]: 104 tmp[86]: 104 tmp[87]: 112 tmp[88]: 112 tmp[89]: 113 tmp[90]: 115 tmp[91]: 99 tmp[92]: 101 tmp[93]: 104 tmp[94]: 112 tmp[95]: 113 tmp[96]: 115 tmp[97]: 119 tmp[98]: 125 tmp[99]: 134 tmp[100]: 138 tmp[101]: 128 tmp[102]: 134 tmp[103]: 139 tmp[104]: 143 tmp[105]: 128 tmp[106]: 134 tmp[107]: 139 tmp[108]: 143 tmp[109]: 1000 tmp[110]: 1000 tmp[111]: 1000 tmp[112]: 1000 tmp[113]: 1000 tmp[114]: 1000 tmp[115]: 1000 tmp[116]: 1000 tmp[117]: 1000 tmp[118]: 1000 tmp[119]: 1000 tmp[120]: 1000  I got the final result!  tmp[0]: 159 tmp[1]: 162 tmp[2]: 163 tmp[3]: 163 tmp[4]: 164 tmp[5]: 166 tmp[6]: 166 tmp[7]: 168 tmp[8]: 169 tmp[9]: 171 tmp[10]: 171 tmp[11]: 172 tmp[12]: 173 tmp[13]: 175 tmp[14]: 175 tmp[15]: 176 tmp[16]: 176 tmp[17]: 163 tmp[18]: 164 tmp[19]: 166 tmp[20]: 166 tmp[21]: 169 tmp[22]: 171 tmp[23]: 171 tmp[24]: 172 tmp[25]: 173 tmp[26]: 175 tmp[27]: 175 tmp[28]: 176 tmp[29]: 176 tmp[30]: 176 tmp[31]: 177 tmp[32]: 163 tmp[33]: 166 tmp[34]: 169 tmp[35]: 171 tmp[36]: 171 tmp[37]: 172 tmp[38]: 173 tmp[39]: 175 tmp[40]: 175 tmp[41]: 176 tmp[42]: 176 tmp[43]: 176 tmp[44]: 177 tmp[45]: 178 tmp[46]: 179 tmp[47]: 183 tmp[48]: 166 tmp[49]: 169 tmp[50]: 171 tmp[51]: 171 tmp[52]: 172 tmp[53]: 173 tmp[54]: 175 tmp[55]: 176 tmp[56]: 177 tmp[57]: 178 tmp[58]: 179 tmp[59]: 183 tmp[60]: 185 tmp[61]: 162 tmp[62]: 163 tmp[63]: 166 tmp[64]: 169 tmp[65]: 171 tmp[66]: 172 tmp[67]: 173 tmp[68]: 175 tmp[69]: 176 tmp[70]: 177 tmp[71]: 178 tmp[72]: 179 tmp[73]: 183 tmp[74]: 185 tmp[75]: 193 tmp[76]: 162 tmp[77]: 163 tmp[78]: 166 tmp[79]: 171 tmp[80]: 172 tmp[81]: 173 tmp[82]: 175 tmp[83]: 176 tmp[84]: 177 tmp[85]: 178 tmp[86]: 179 tmp[87]: 183 tmp[88]: 185 tmp[89]: 193 tmp[90]: 194 tmp[91]: 198 tmp[65 158 249 356 480 562 663 770 866 this is threads:7  \*\*\*I put an egg\*\*\*  65 158 249 356 480 562 663 770 866 this is threads:10  \*\*\*I put an egg\*\*\*  \*\*\*I put an egg\*\*\*  65 158 249 356 480 562 663 770 866 this is threads:4  \*\*\*I put an egg\*\*\*  65 158 249 356 480 562 663 770 866 this is threads:6  \*\*\*I put an egg\*\*\*  65 158 249 356 480 562 663 770 866 this is threads:8  \*\*\*I put an egg\*\*\*  92]: 200 tmp[93]: 200 tmp[94]: 201 tmp[95]: 176 tmp[96]: 177 tmp[97]: 178 tmp[98]: 179 tmp[99]: 183 tmp[100]: 194 tmp[101]: 198 tmp[102]: 200 tmp[103]: 200 tmp[104]: 201 tmp[105]: 201 tmp[106]: 204 tmp[107]: 206 tmp[108]: 208 tmp[109]: 208 tmp[110]: 179 tmp[111]: 183 tmp[112]: 198 tmp[113]: 200 tmp[114]: 201 tmp[115]: 201 tmp[116]: 204 tmp[117]: 206 tmp[118]: 208 tmp[119]: 208 tmp[120]: 208 tmp[121]: 210 tmp[122]: 216 tmp[123]: 228 tmp[124]: 228 tmp[125]: 232 tmp[126]: 233 tmp[127]: 208 tmp[128]: 216 tmp[129]: 228 tmp[130]: 228 tmp[131]: 238 tmp[132]: 241 tmp[133]: 1000 tmp[134]: 1000 tmp[135]: 1000 tmp[136]: 1000 tmp[137]: 1000 tmp[138]: 1000 tmp[139]: 1000 tmp[140]: 1000 tmp[141]: 1000 tmp[142]: 1000 tmp[143]: 1000 tmp[144]: 1000 tmp[145]: 1000 tmp[146]: 1000 tmp[147]: 1000  I got the final result!  tmp[0]: 250 tmp[1]: 250 tmp[2]: 250 tmp[3]: 250 tmp[4]: 251 tmp[5]: 251 tmp[6]: 253 tmp[7]: 254 tmp[8]: 254 tmp[9]: 254 tmp[10]: 255 tmp[11]: 255 tmp[12]: 258 tmp[13]: 260 tmp[14]: 260 tmp[15]: 250 tmp[16]: 250 tmp[17]: 250 tmp[18]: 251 tmp[19]: 251 tmp[20]: 253 tmp[21]: 254 tmp[22]: 254 tmp[23]: 254 tmp[24]: 255 tmp[25]: 258 tmp[26]: 260 tmp[27]: 260 tmp[28]: 250 tmp[29]: 250 tmp[30]: 250 tmp[31]: 250 tmp[32]: 251 tmp[33]: 251 tmp[34]: 253 tmp[35]: 254 tmp[36]: 254 tmp[37]: 254 tmp[38]: 255 tmp[39]: 258 tmp[40]: 260 tmp[41]: 260 tmp[42]: 250 tmp[43]: 250 tmp[44]: 250 tmp[45]: 250 tmp[46]: 251 tmp[47]: 251 tmp[48]: 253 tmp[49]: 254 tmp[50]: 254 tmp[51]: 254 tmp[52]: 255 tmp[53]: 258 tmp[54]: 260 tmp[55]: 260 tmp[56]: 261 tmp[57]: 261 tmp[58]: 262 tmp[59]: 262 tmp[60]: 266 tmp[61]: 268 tmp[62]: 255 tmp[63]: 258 tmp[64]: 260 tmp[65]: 261 tmp[66]: 261 tmp[67]: 262 tmp[68]: 266 tmp[69]: 268 tmp[70]: 271 tmp[71]: 274 tmp[72]: 274 tmp[73]: 254 tmp[74]: 255 tmp[75]: 258 tmp[76]: 260 tmp[77]: 261 tmp[78]: 261 tmp[79]: 266 tmp[80]: 268 tmp[81]: 271 tmp[82]: 274 tmp[83]: 274 tmp[84]: 275 tmp[85]: 280 tmp[86]: 284 tmp[87]: 287 tmp[88]: 260 tmp[89]: 261 tmp[90]: 261 tmp[91]: 266 tmp[92]: 271 tmp[93]: 274 tmp[94]: 275 tmp[95]: 280 tmp[96]: 284 tmp[97]: 287 tmp[98]: 290 tmp[99]: 296 tmp[100]: 298 tmp[101]: 306 tmp[102]: 307 tmp[103]: 308 tmp[104]: 275 tmp[105]: 280 tmp[106]: 284 tmp[107]: 287 tmp[108]: 290 tmp[109]: 296 tmp[110]: 306 tmp[111]: 308 tmp[112]: 337 tmp[113]: 340 tmp[114]: 287 tmp[115]: 290 tmp[116]: 296 tmp[117]: 306 tmp[118]: 316 tmp[119]: 320 tmp[120]: 332 tmp[121]: 340 tmp[122]: 1000 tmp[123]: 1000 tmp[124]: 1000 tmp[125]: 1000 tmp[126]: 1000 tmp[127]: 1000 tmp[128]: 1000 tmp[129]: 1000 tmp[130]: 1000 tmp[131]: 1000 tmp[132]: 1000 tmp[133]: 1000 tmp[134]: 1000 tmp[135]: 1000 tmp[136]: 1000 tmp[137]: 1000 tmp[138]: 1000 tmp[139]: 1000 tmp[140]: 1000 tmp[141]: 1000 tmp[142]: 1000 tmp[143]: 1000 tmp[144]: 1000 tmp[145]: 1000 tmp[146]: 1000 tmp[147]: 1000  I got the final result!  tmp[0]: 357 tmp[1]: 360 tmp[2]: 361 tmp[3]: 363 tmp[4]: 365 tmp[5]: 365 tmp[6]: 366 tmp[7]: 366 tmp[8]: 367 tmp[9]: 368 tmp[10]: 371 tmp[11]: 372 tmp[12]: 373 tmp[13]: 374 tmp[14]: 375 tmp[15]: 376 tmp[16]: 379 tmp[17]: 379 tmp[18]: 380 tmp[19]: 361 tmp[20]: 365 tmp[21]: 365 tmp[22]: 366 tmp[23]: 366 tmp[24]: 367 tmp[25]: 368 tmp[26]: 371 tmp[27]: 372 tmp[28]: 373 tmp[29]: 374 tmp[30]: 375 tmp[31]: 379 tmp[32]: 379 tmp[33]: 380 tmp[34]: 380 tmp[35]: 380 tmp[36]: 381 tmp[37]: 381 tmp[38]: 381 tmp[39]: 382 tmp[40]: 372 tmp[41]: 373 tmp[42]: 374 tmp[43]: 375 tmp[44]: 379 tmp[45]: 379 tmp[46]: 380 tmp[47]: 380 tmp[48]: 381 tmp[49]: 381 tmp[50]: 382 tmp[51]: 382 tmp[52]: 382 tmp[53]: 383 tmp[54]: 383 tmp[55]: 367 tmp[56]: 368 tmp[57]: 372 tmp[58]: 373 tmp[59]: 374 tmp[60]: 375 tmp[61]: 379 tmp[62]: 380 tmp[63]: 381 tmp[64]: 381 tmp[65]: 382 tmp[66]: 383 tmp[67]: 383 tmp[68]: 395 tmp[69]: 398 tmp[70]: 365 tmp[71]: 366 tmp[72]: 367 tmp[73]: 368 tmp[74]: 373 tmp[75]: 374 tmp[76]: 375 tmp[77]: 379 tmp[78]: 381 tmp[79]: 381 tmp[80]: 382 tmp[81]: 383 tmp[82]: 383 tmp[83]: 395 tmp[84]: 398 tmp[85]: 399 tmp[86]: 401 tmp[87]: 401 tmp[88]: 403 tmp[89]: 368 tmp[90]: 373 tmp[91]: 375 tmp[92]: 379 tmp[93]: 381 tmp[94]: 382 tmp[95]: 383 tmp[96]: 383 tmp[97]: 395 tmp[98]: 398 tmp[99]: 399 tmp[100]: 401 tmp[101]: 403 tmp[102]: 404 tmp[103]: 404 tmp[104]: 406 tmp[105]: 408 tmp[106]: 409 tmp[107]: 412 tmp[108]: 415 tmp[109]: 383 tmp[110]: 383 tmp[111]: 398 tmp[112]: 399 tmp[113]: 403 tmp[114]: 404 tmp[115]: 406 tmp[116]: 408 tmp[117]: 412 tmp[118]: 415 tmp[119]: 415 tmp[120]: 424 tmp[121]: 425 tmp[122]: 404 tmp[123]: 406 tmp[124]: 412 tmp[125]: 415 tmp[126]: 424 tmp[127]: 425 tmp[128]: 425 tmp[129]: 420 tmp[130]: 430 tmp[131]: 441 tmp[132]: 444 tmp[133]: 446 tmp[134]: 458 tmp[135]: 1000 tmp[136]: 1000 tmp[137]: 1000 tmp[138]: 1000 tmp[139]: 1000 tmp[140]: 1000 tmp[141]: 1000 tmp[142]: 1000 tmp[143]: 1000 tmp[144]: 1000 tmp[145]: 1000 tmp[146]: 1000 tmp[147]: 1000 tmp[148]: 1000 tmp[149]: 1000 tmp[150]: 1000 tmp[151]: 1000  I got the final result!  tmp[0]: 482 tmp[1]: 483 tmp[2]: 483 tmp[3]: 485 tmp[4]: 487 tmp[5]: 488 tmp[6]: 490 tmp[7]: 490 tmp[8]: 491 tmp[9]: 491 tmp[10]: 492 tmp[11]: 494 tmp[12]: 497 tmp[13]: 498 tmp[14]: 498 tmp[15]: 499 tmp[16]: 483 tmp[17]: 483 tmp[18]: 485 tmp[19]: 487 tmp[20]: 488 tmp[21]: 490 tmp[22]: 490 tmp[23]: 491 tmp[24]: 491 tmp[25]: 492 tmp[26]: 494 tmp[27]: 497 tmp[28]: 498 tmp[29]: 499 tmp[30]: 499 tmp[31]: 483 tmp[32]: 483 tmp[33]: 485 tmp[34]: 487 tmp[35]: 490 tmp[36]: 490 tmp[37]: 491 tmp[38]: 491 tmp[39]: 492 tmp[40]: 494 tmp[41]: 497 tmp[42]: 498 tmp[43]: 499 tmp[44]: 499 tmp[45]: 499 tmp[46]: 501 tmp[47]: 487 tmp[48]: 490 tmp[49]: 490 tmp[50]: 491 tmp[51]: 491 tmp[52]: 492 tmp[53]: 494 tmp[54]: 497 tmp[55]: 498 tmp[56]: 499 tmp[57]: 499 tmp[58]: 499 tmp[59]: 501 tmp[60]: 503 tmp[61]: 504 tmp[62]: 506 tmp[63]: 491 tmp[64]: 491 tmp[65]: 492 tmp[66]: 494 tmp[67]: 497 tmp[68]: 498 tmp[69]: 499 tmp[70]: 499 tmp[71]: 501 tmp[72]: 503 tmp[73]: 504 tmp[74]: 506 tmp[75]: 510 tmp[76]: 513 tmp[77]: 518 tmp[78]: 519 tmp[79]: 494 tmp[80]: 497 tmp[81]: 498 tmp[82]: 499 tmp[83]: 503 tmp[84]: 504 tmp[85]: 506 tmp[86]: 510 tmp[87]: 513 tmp[88]: 518 tmp[89]: 519 tmp[90]: 521 tmp[91]: 524 tmp[92]: 526 tmp[93]: 527 tmp[94]: 497 tmp[95]: 503 tmp[96]: 504 tmp[97]: 506 tmp[98]: 510 tmp[99]: 513 tmp[100]: 518 tmp[101]: 519 tmp[102]: 521 tmp[103]: 524 tmp[104]: 527 tmp[105]: 531 tmp[106]: 532 tmp[107]: 533 tmp[108]: 535 tmp[109]: 536 tmp[110]: 537 tmp[111]: 542 tmp[112]: 544 tmp[113]: 548 tmp[114]: 549 tmp[115]: 550 tmp[116]: 550 tmp[117]: 1000 tmp[118]: 1000 tmp[119]: 1000 tmp[120]: 1000 tmp[121]: 1000 tmp[122]: 1000 tmp[123]: 1000 tmp[124]: 1000 tmp[125]: 1000 tmp[126]: 1000 tmp[127]: 1000 tmp[128]: 1000 tmp[129]: 1000 tmp[130]: 1000 tmp[131]: 1000 tmp[132]: 444 tmp[133]: 446 tmp[134]: 458 tmp[135]: 1000 tmp[136]: 1000 tmp[137]: 1000 tmp[138]: 1000 tmp[139]: 1000 tmp[140]: 1000 tmp[141]: 1000 tmp[142]: 1000 tmp[143]: 1000 tmp[144]: 1000 tmp[145]: 1000 tmp[146]: 1000 tmp[147]: 1000 tmp[148]: 1000 tmp[149]: 1000 tmp[150]: 1000 tmp[151]: 1000  I got the final result!  tmp[0]: 563 tmp[1]: 565 tmp[2]: 565 tmp[3]: 567 tmp[4]: 568 tmp[5]: 568 tmp[6]: 569 tmp[7]: 569 tmp[8]: 570 tmp[9]: 574 tmp[10]: 574 tmp[11]: 574 tmp[12]: 574 tmp[13]: 574 tmp[14]: 575 tmp[15]: 575 tmp[16]: 575 tmp[17]: 575 tmp[18]: 576 tmp[19]: 579 tmp[20]: 580 tmp[21]: 580 tmp[22]: 582 tmp[23]: 583 tmp[24]: 583 tmp[25]: 574 tmp[26]: 574 tmp[27]: 574 tmp[28]: 575 tmp[29]: 575 tmp[30]: 575 tmp[31]: 579 tmp[32]: 583 tmp[33]: 583 tmp[34]: 584 tmp[35]: 584 tmp[36]: 567 tmp[37]: 568 tmp[38]: 568 tmp[39]: 569 tmp[40]: 570 tmp[41]: 574 tmp[42]: 574 tmp[43]: 574 tmp[44]: 575 tmp[45]: 575 tmp[46]: 579 tmp[47]: 583 tmp[48]: 583 tmp[49]: 584 tmp[50]: 584 tmp[51]: 586 tmp[52]: 586 tmp[53]: 588 tmp[54]: 568 tmp[55]: 568 tmp[56]: 570 tmp[57]: 574 tmp[58]: 574 tmp[59]: 574 tmp[60]: 575 tmp[61]: 579 tmp[62]: 583 tmp[63]: 583 tmp[64]: 584 tmp[65]: 584 tmp[66]: 586 tmp[67]: 588 tmp[68]: 590 tmp[69]: 565 tmp[70]: 568 tmp[71]: 568 tmp[72]: 570 tmp[73]: 574 tmp[74]: 574 tmp[75]: 574 tmp[76]: 575 tmp[77]: 579 tmp[78]: 583 tmp[79]: 583 tmp[80]: 584 tmp[81]: 584 tmp[82]: 586 tmp[83]: 588 tmp[84]: 590 tmp[85]: 591 tmp[86]: 592 tmp[87]: 599 tmp[88]: 601 tmp[89]: 602 tmp[90]: 574 tmp[91]: 574 tmp[92]: 579 tmp[93]: 584 tmp[94]: 584 tmp[95]: 586 tmp[96]: 590 tmp[97]: 591 tmp[98]: 592 tmp[99]: 599 tmp[100]: 601 tmp[101]: 602 tmp[102]: 604 tmp[103]: 604 tmp[104]: 606 tmp[105]: 608 tmp[106]: 618 tmp[107]: 579 tmp[108]: 584 tmp[109]: 586 tmp[110]: 590 tmp[111]: 592 tmp[112]: 599 tmp[113]: 602 tmp[114]: 604 tmp[115]: 604 tmp[116]: 606 tmp[117]: 608 tmp[118]: 618 tmp[119]: 620 tmp[120]: 623 tmp[121]: 625 tmp[122]: 628 tmp[123]: 579 tmp[124]: 584 tmp[125]: 586 tmp[126]: 590 tmp[127]: 592 tmp[128]: 599 tmp[129]: 604 tmp[130]: 604 tmp[131]: 608 tmp[132]: 620 tmp[133]: 623 tmp[134]: 625 tmp[135]: 628 tmp[136]: 638 tmp[137]: 647 tmp[138]: 622 tmp[139]: 626 tmp[140]: 638 tmp[141]: 639 tmp[142]: 661 tmp[143]: 663 tmp[144]: 1000 tmp[145]: 1000 tmp[146]: 1000 tmp[147]: 1000 tmp[148]: 1000 tmp[149]: 1000 tmp[150]: 1000 tmp[151]: 1000 tmp[152]: 1000 tmp[153]: 1000 tmp[154]: 1000 tmp[155]: 1000 tmp[156]: 1000 tmp[157]: 1000 tmp[158]: 1000 tmp[159]: 1000 tmp[160]: 1000  I got the final result!  tmp[0]: 664 tmp[1]: 664 tmp[2]: 668 tmp[3]: 668 tmp[4]: 669 tmp[5]: 670 tmp[6]: 671 tmp[7]: 671 tmp[8]: 672 tmp[9]: 672 tmp[10]: 674 tmp[11]: 675 tmp[12]: 676 tmp[13]: 678 tmp[14]: 664 tmp[15]: 664 tmp[16]: 668 tmp[17]: 668 tmp[18]: 669 tmp[19]: 670 tmp[20]: 671 tmp[21]: 671 tmp[22]: 672 tmp[23]: 672 tmp[24]: 674 tmp[25]: 675 tmp[26]: 676 tmp[27]: 678 tmp[28]: 679 tmp[29]: 664 tmp[30]: 668 tmp[31]: 669 tmp[32]: 670 tmp[33]: 671 tmp[34]: 671 tmp[35]: 672 tmp[36]: 672 tmp[37]: 674 tmp[38]: 675 tmp[39]: 676 tmp[40]: 678 tmp[41]: 679 tmp[42]: 680 tmp[43]: 680 tmp[44]: 681 tmp[45]: 671 tmp[46]: 672 tmp[47]: 672 tmp[48]: 674 tmp[49]: 675 tmp[50]: 676 tmp[51]: 678 tmp[52]: 679 tmp[53]: 680 tmp[54]: 680 tmp[55]: 681 tmp[56]: 682 tmp[57]: 683 tmp[58]: 684 tmp[59]: 671 tmp[60]: 672 tmp[61]: 674 tmp[62]: 675 tmp[63]: 676 tmp[64]: 678 tmp[65]: 679 tmp[66]: 680 tmp[67]: 682 tmp[68]: 683 tmp[69]: 684 tmp[70]: 685 tmp[71]: 685 tmp[72]: 688 tmp[73]: 699 tmp[74]: 674 tmp[75]: 676 tmp[76]: 678 tmp[77]: 679 tmp[78]: 680 tmp[79]: 682 tmp[80]: 683 tmp[81]: 684 tmp[82]: 685 tmp[83]: 699 tmp[84]: 699 tmp[85]: 702 tmp[86]: 702 tmp[87]: 705 tmp[88]: 678 tmp[89]: 679 tmp[90]: 680 tmp[91]: 682 tmp[92]: 683 tmp[93]: 684 tmp[94]: 685 tmp[95]: 699 tmp[96]: 702 tmp[97]: 705 tmp[98]: 705 tmp[99]: 676 tmp[100]: 678 tmp[101]: 679 tmp[102]: 680 tmp[103]: 683 tmp[104]: 684 tmp[105]: 699 tmp[106]: 702 tmp[107]: 705 tmp[108]: 709 tmp[109]: 710 tmp[110]: 718 tmp[111]: 725 tmp[112]: 730 tmp[113]: 734 tmp[114]: 1000 tmp[115]: 1000 tmp[116]: 1000 tmp[117]: 1000 tmp[118]: 1000 tmp[119]: 1000 tmp[120]: 1000 tmp[121]: 1000 tmp[122]: 1000 tmp[123]: 1000 tmp[124]: 1000 tmp[125]: 1000 tmp[126]: 1000 tmp[127]: 1000 tmp[128]: 599 tmp[129]: 604 tmp[130]: 604 tmp[131]: 608 tmp[132]: 620 tmp[133]: 623 tmp[134]: 625 tmp[135]: 628 tmp[136]: 638 tmp[137]: 647 tmp[138]: 622 tmp[139]: 626 tmp[140]: 638 tmp[141]: 639 tmp[142]: 661 tmp[143]: 663 tmp[144]: 1000 tmp[145]: 1000 tmp[146]: 1000 tmp[147]: 1000 tmp[148]: 1000 tmp[149]: 1000 tmp[150]: 1000 tmp[151]: 1000 tmp[152]: 1000 tmp[153]: 1000 tmp[154]: 1000 tmp[155]: 1000 tmp[156]: 1000 tmp[157]: 1000 tmp[158]: 1000 tmp[159]: 1000 tmp[160]: 1000  I got the final result!  tmp[0]: 771 tmp[1]: 772 tmp[2]: 775 tmp[3]: 776 tmp[4]: 776 tmp[5]: 777 tmp[6]: 777 tmp[7]: 779 tmp[8]: 780 tmp[9]: 781 tmp[10]: 782 tmp[11]: 783 tmp[12]: 783 tmp[13]: 784 tmp[14]: 785 tmp[15]: 786 tmp[16]: 786 tmp[17]: 787 tmp[18]: 772 tmp[19]: 775 tmp[20]: 776 tmp[21]: 776 tmp[22]: 777 tmp[23]: 777 tmp[24]: 780 tmp[25]: 781 tmp[26]: 782 tmp[27]: 783 tmp[28]: 783 tmp[29]: 784 tmp[30]: 785 tmp[31]: 786 tmp[32]: 786 tmp[33]: 787 tmp[34]: 787 tmp[35]: 787 tmp[36]: 789 tmp[37]: 790 tmp[38]: 792 tmp[39]: 777 tmp[40]: 777 tmp[41]: 783 tmp[42]: 783 tmp[43]: 785 tmp[44]: 786 tmp[45]: 786 tmp[46]: 787 tmp[47]: 787 tmp[48]: 789 tmp[49]: 790 tmp[50]: 792 tmp[51]: 794 tmp[52]: 794 tmp[53]: 796 tmp[54]: 776 tmp[55]: 777 tmp[56]: 777 tmp[57]: 783 tmp[58]: 785 tmp[59]: 786 tmp[60]: 787 tmp[61]: 787 tmp[62]: 789 tmp[63]: 790 tmp[64]: 792 tmp[65]: 794 tmp[66]: 796 tmp[67]: 776 tmp[68]: 777 tmp[69]: 777 tmp[70]: 785 tmp[71]: 786 tmp[72]: 787 tmp[73]: 787 tmp[74]: 789 tmp[75]: 790 tmp[76]: 792 tmp[77]: 794 tmp[78]: 796 tmp[79]: 797 tmp[80]: 800 tmp[81]: 802 tmp[82]: 804 tmp[83]: 804 tmp[84]: 805 tmp[85]: 809 tmp[86]: 796 tmp[87]: 797 tmp[88]: 800 tmp[89]: 802 tmp[90]: 804 tmp[91]: 804 tmp[92]: 809 tmp[93]: 811 tmp[94]: 776 tmp[95]: 777 tmp[96]: 785 tmp[97]: 786 tmp[98]: 792 tmp[99]: 794 tmp[100]: 796 tmp[101]: 797 tmp[102]: 800 tmp[103]: 802 tmp[104]: 804 tmp[105]: 809 tmp[106]: 811 tmp[107]: 811 tmp[108]: 816 tmp[109]: 820 tmp[110]: 821 tmp[111]: 822 tmp[112]: 825 tmp[113]: 831 tmp[114]: 832 tmp[115]: 835 tmp[116]: 838 tmp[117]: 840 tmp[118]: 1000 tmp[119]: 1000 tmp[120]: 1000 tmp[121]: 1000 tmp[122]: 1000 tmp[123]: 1000 tmp[124]: 1000 tmp[125]: 1000 tmp[126]: 1000 tmp[127]: 1000 tmp[128]: 1000 tmp[129]: 1000 tmp[130]: 1000 tmp[131]: 1000 tmp[132]: 1000 tmp[133]: 1000 tmp[134]: 1000 tmp[135]: 628 tmp[136]: 638 tmp[137]: 647 tmp[138]: 622 tmp[139]: 626 tmp[140]: 638 tmp[141]: 639 tmp[142]: 661 tmp[143]: 663 tmp[144]: 1000 tmp[145]: 1000 tmp[146]: 1000 tmp[147]: 1000 tmp[148]: 1000 tmp[149]: 1000 tmp[150]: 1000 tmp[151]: 1000 tmp[152]: 1000 tmp[153]: 1000 tmp[154]: 1000 tmp[155]: 1000 tmp[156]: 1000 tmp[157]: 1000 tmp[158]: 1000 tmp[159]: 1000 tmp[160]: 1000 |

### 代码

|  |
| --- |
| #include"mpi.h"  #include<stdio.h>  #include<stdlib.h>  #include<time.h>  #define MAXN 1000  int tag=333;  int max\_length=500;  MPI\_Status status;  /\*qsort自定义比较函数\*/  int comp(const void\*p,const void\*q){  return(\*(int\*)p - \*(int\*)q);  }  //用于生成需要排序的数据  void generate\_data(int \*data\_list,int size){  int i;  srand(time(NULL));//给它一个种子，开始生成随机数  for(i=0;i<size;i++){  data\_list[i]=rand()%MAXN;//生成1000个随机数  }  }  //进程0分发数据  void send\_data(int\* source,int p,int len){  int i;  for(i=1;i<=p;i++){  //source随机生成的数据，len每个进程能分到的数据长度，逐个分发给其他进程  MPI\_Send(source+(i-1)\*len,len,MPI\_INT,i,tag,MPI\_COMM\_WORLD);  }  }  //进程1-（numprocs-1）开始接收数据  void receive\_data(int \*sample,int sample\_len){  //sample：接受到的样本数据，sample\_len:样本数据集大小  MPI\_Recv(sample,sample\_len,MPI\_INT,0,tag,MPI\_COMM\_WORLD,&status);  //对接收到的数据进行快速排序，按照从小到大顺序  qsort(sample,sample\_len,sizeof(int),comp);  //表示当前进程接收数据成功。  printf("data received\n");  }  //进程1-（numprocs-1）开始挑选p个样本元素  void select\_represent(int \*sample,int len,int w,int p){  int i=0,count=0;  //存放挑选出来的样本数据  int\* tmp=(int\*)malloc(p\*sizeof(int));  for( i=0;count<=p&&i<len;i+=w){  tmp[count]=sample[i];  count++;  }  //将p个数据集传送给进程0  MPI\_Send(tmp,p,MPI\_INT,0,tag+1,MPI\_COMM\_WORLD);  }  //进程0获取其他进程传给他的p\*p个代表数组  int\* fetch\_represent(int p){  int len=p\*p,i,\*tmp;  int \*main\_item;  //生成的主元的记录数组，共p-1个。  main\_item=(int\*)malloc((p-1)\*sizeof(int));  int count=0;  //存放其他进程传给他的p个数据集，大小为p\*p  tmp=(int\*)malloc(len\*sizeof(int));  printf("debuging\n");  for( i=1;i<=p;i++){  //接受其他进程传给他的数据,用数组偏移的方法，节省空间，便于下面操作。  MPI\_Recv(tmp+p\*(i-1),p,MPI\_INT,i,tag+1,MPI\_COMM\_WORLD,&status);  }  //获取主元之前，必须要进行排序，这里还是采用快速排序，qsort  qsort(tmp,len,sizeof(int),comp);  for(i=p;i<len;i+=p){  //采用步进的方式，从排序后的数组内读取p-1个主元  main\_item[count++]=tmp[i];  /\*printf("main\_items[%d]:%d\n",count-1,main\_item[count-1]);\*/  }  return main\_item;  }  //由于在使用MPI\_Bcast时出现一些故障，故采用MPI\_Send和MPI\_Recv组合，将p-1个主元发给各个进程。  void send\_main\_item(int \*main\_item,int len,int p){  int i=0;  for(i=1;i<=p;i++){  //main\_item：包含p-1个主元  MPI\_Send(main\_item,len,MPI\_INT,i,tag+1,MPI\_COMM\_WORLD);  }  }  void receive\_main\_item(int \*items,int len){  int i;  MPI\_Recv(items,len,MPI\_INT,0,tag+1,MPI\_COMM\_WORLD,&status);  //接受之后，查看是否正确，输出到屏幕，以便检查。  for(i=0;i<len;i++){  printf("%d ",items[i]);  }  }  //对于进程0-（numprocs-1），当接受完主元后，开始划分样本数据  void split(int\*sample,int len,int\*items,int len\_p){  int start=0,end=0,i,count=0;  for(i=0;i<len;i++){  /\*printf("sample[%d]:%d\n",i,sample[i]);\*/  }  for(i=0;i<len\_p;i++){  /\*printf("items[%d]:%d\n",i,items[i]);\*/  }  //len\_p:p-1个主元；items：主元存储的数组；sample：样本数据存储的数组；  //start：划分的起点，end:划分的终点  for(i=0;i<len\_p&&end<len;){ /\*printf("items[%d]:%d;sample[%d]:%d\n",i,items[i],end,sample[end]);\*/  //满足这个条件，就表示找到新的一块划分  if(items[i]<sample[end]){  int j;  //将第i块划分的数据传送给进程i  MPI\_Send(sample+start,end-start,MPI\_INT,i+1,tag+2,MPI\_COMM\_WORLD);  start=end;//更新首尾位置  i++;  }  else{  end++;  }  }    //最后一块数据，不满足上述性质，需要单独处理  if(end<len){  int i;  MPI\_Send(sample+end,len-end,MPI\_INT,len\_p+1,tag+2,MPI\_COMM\_WORLD);  }  }  //进程1-（numprocs-1）获取被主元分块后的数据，每个进程都要接受来自剩余全部进程的数据。  void fetch\_merge(int numprocs){  int len=0,count=0,curlen=0,fin\_cnt=0;  int \*sorted,\*tmp;  int i,j;  int\* splited\_items=(int\*)malloc(max\_length\*sizeof(int));  sorted=(int\*)malloc(max\_length\*sizeof(int));  tmp=(int\*)malloc(max\_length\*sizeof(int));  for(i=0;i<max\_length;i++){  tmp[i]=MAXN;//初始化  }    printf("\*\*\*I put an flag\*\*\*\n");  for(i=1;i<numprocs;i++){  int a=0,b=0;  //由于无法确定传给当前进程的数据量的大小，故需要预设一个最大值；初始化该数组，数为了保证能获取传送给我的数据量；splited\_items:存储来自各进程分块数据。  memset(splited\_items,0,sizeof(splited\_items));  MPI\_Recv(splited\_items,max\_length,MPI\_INT,i,tag+2,MPI\_COMM\_WORLD,&status);  for(j=0,count=0;j<max\_length&&splited\_items[j]!=0;j++)  count++;//只有满足非零元素的数字才能被记为有效数字，统计传给我的数组实际大小。    /\*printf("current count:%d\n",count);\*/  for(a=0,b=0,fin\_cnt=0;a<count&&b<curlen;){  //sotred：存放当前已排序的数据，tmp：上一次已排序的数组，splited\_item:刚刚读取的已排序的数组，这里采用归并排序，如果采用qsort会节省代码量，但是却提高了排序复杂度，故选用归并排序。  sorted[fin\_cnt++]=(tmp[a]<splited\_items[b]?tmp[a++]:splited\_items[b++]);  /\*printf("sorted[%d]:%d\n",fin\_cnt-1,sorted[fin\_cnt-1]);\*/  }  //我们还需要确保剩余的数组内的数据也要存入sorted内  while(a<count) sorted[fin\_cnt++]=tmp[a++];  while(b<curlen) sorted[fin\_cnt++]=tmp[b++];  for(j=0;j<curlen;j++)  tmp[j]=sorted[j];//更新tmp数组  curlen+=count;//更新当前已排序数组sorted长度  }  MPI\_Barrier(MPI\_COMM\_WORLD);  //将已排序的数据发给进程0，到这里所有工作完成  MPI\_Send(sorted,curlen,MPI\_INT,0,tag+3,MPI\_COMM\_WORLD);  }  //进程0接受来自其他进程已经排序后的数据  //  void collect\_result(int p){  int i,j;  int \*tmp=(int\*)malloc(max\_length\*sizeof(int));  for(i=1;i<=p;i++){  memset(tmp,0,sizeof(tmp));  //tmp:存储其他进程发来的数据  MPI\_Recv(tmp,max\_length,MPI\_INT,i,tag+3,MPI\_COMM\_WORLD,&status);  printf("\nI got the final result!\n");  for(j=0;j<max\_length&&tmp[j]!=0;j++)  printf("tmp[%d]: %d ",j,tmp[j]);//打印结果  }  }    int main(int argc,char\* argv[]){  int \*source,\*sample;  int size=1000,p;  int myid,numprocs;  int sample\_len;  int w;  int \* main\_item,\*items;  MPI\_Status status;  MPI\_Init(&argc,&argv);  MPI\_Comm\_rank(MPI\_COMM\_WORLD,&myid);  MPI\_Comm\_size(MPI\_COMM\_WORLD,&numprocs);  p=numprocs-1;  sample\_len=size/p;  w=sample\_len/p;    if(myid==0){  source=(int \*)malloc(size\*sizeof(int));  generate\_data(source,size);//主进程生成一组数据  send\_data(source,p,sample\_len);//发送这些生成的数据  }else{  sample=(int\*)malloc(sample\_len\*sizeof(int));//  receive\_data(sample,sample\_len);//从进程接受样本数据  }  MPI\_Barrier(MPI\_COMM\_WORLD);//同步  if(myid==0){  int i=0;  main\_item=fetch\_represent(p);//接受p\*p个代表元素  /\*printf("debugging main\_items returned failed?\n");  for(i=0;i<p-1;i++){  printf("%d ",main\_item[i]);  }\*/  send\_main\_item(main\_item,p-1,p);//挑选p-1个主元，再发给个个从进程  }else{  items=(int\*)malloc((p-1)\*sizeof(int));  select\_represent(sample,sample\_len,w,p);//从进程挑选p个数据，间隔为w  /\* printf("\*\*\*this is procs:%d\*\*\*\n",myid);\*/  receive\_main\_item(items,p-1);//获取p-1个主元  }  MPI\_Barrier(MPI\_COMM\_WORLD);  if(myid!=0){  split(sample,sample\_len,items,p-1);//从进程划分样本数组  printf("this is threads:%d",myid);  /\*fetch\_merge(numprocs);\*/  printf("\n");  }  MPI\_Barrier(MPI\_COMM\_WORLD);  if(myid!=0){  fetch\_merge(numprocs);//从进程开始获取被主元分块后的数据，接着，各个进程开始归并排序  MPI\_Barrier(MPI\_COMM\_WORLD);    }  MPI\_Barrier(MPI\_COMM\_WORLD);  if(myid==0){  printf("I am your admin\n");  collect\_result(p);//打印结果  }    MPI\_Finalize();  } |