SIT315 M2.T2C: Complex Threading With OpenMP

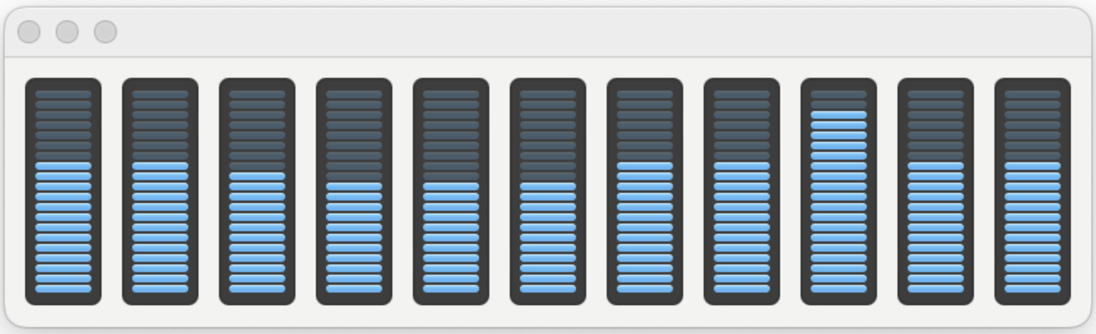
Codey Funston | S222250824

Decomposition and Performance Comparison

I decided to implement a quicksort algorithm that used recursion as that seemed to work better conceptually with parallelising the program. The function runs its body and then has two recursive calls, splitting the data in two based on a random pivot element. I used OpenMP tasks to assign threads to the recursive calls since their dynamic scheduling is very good for this sort of task, especially for load balancing when quicksort() calls are done with unbalanced data partitions.

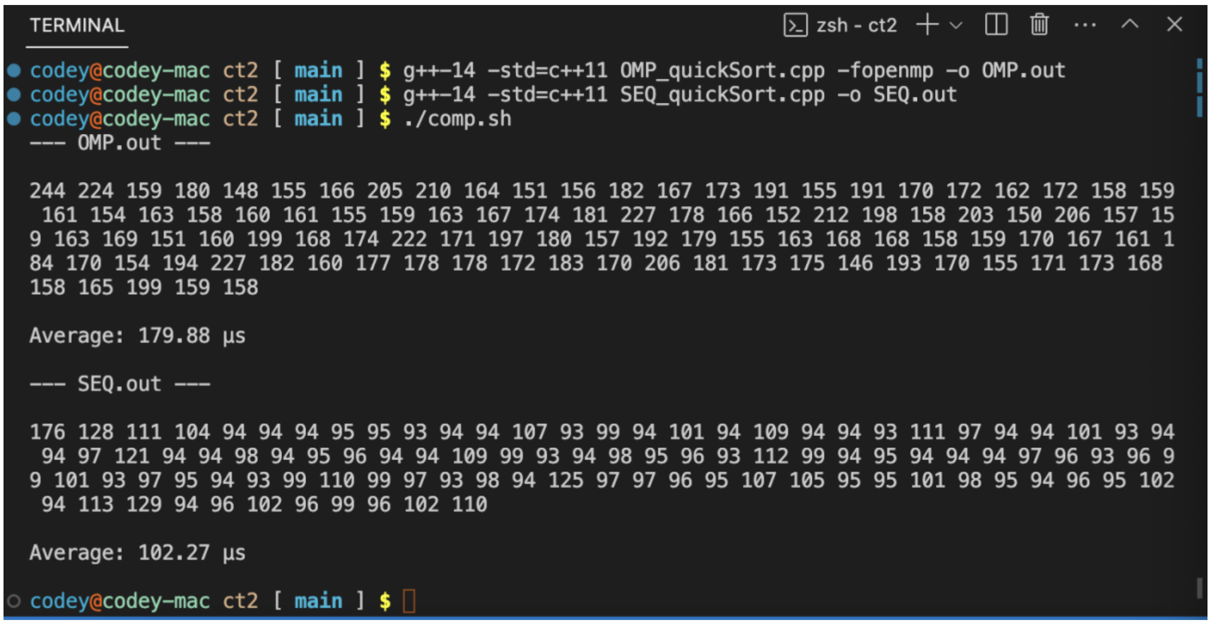
The runtime of the sequential program isn't that bad given that the time complexity for quicksort is $O(nlog(n))$. For very small data input the sequential version runs faster, for example with a vector of 2000 integers, over 100 tests for each program the average for the sequential version was 102.27 microseconds, compared to the OMP version's average of 179.88 microseconds. This is because the threading overhead is much too high. To help reduce the overhead I added a condition based on testing output to only assign a thread to a recursive call if the section of data being worked on consisted of more than 1000 integers.

The core distribution was nice and even throughout each test, usually looking similar to this clipping from a test of the OpenMP version with 20 million integers:



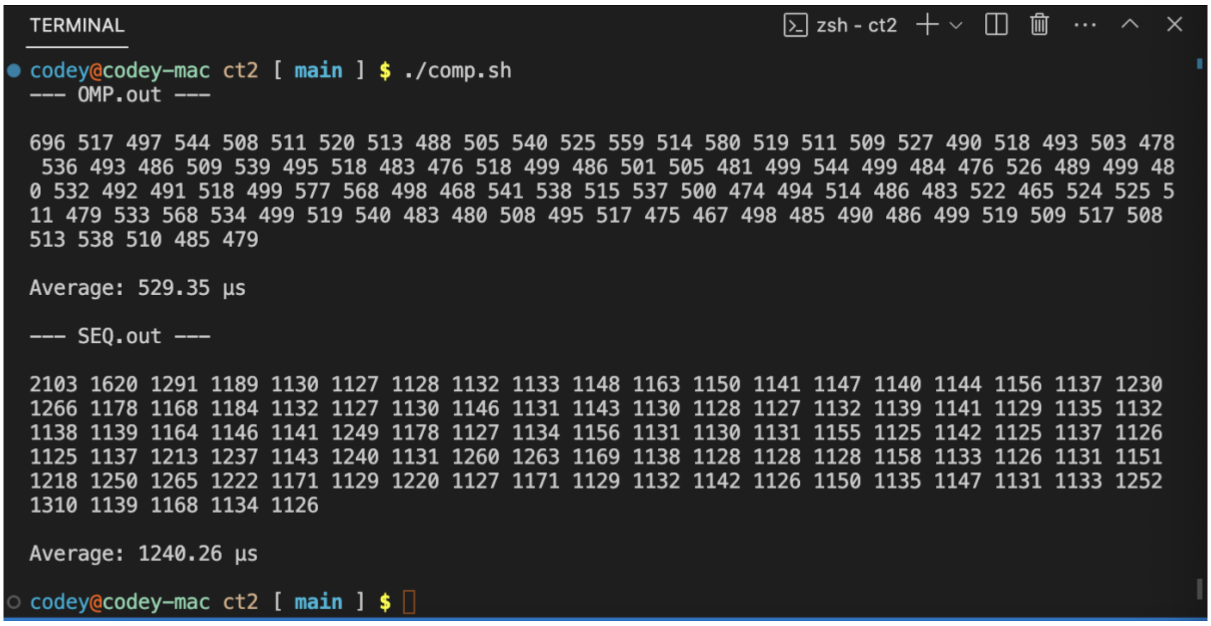
Some Results and Corresponding Input Data Size

- 2,000



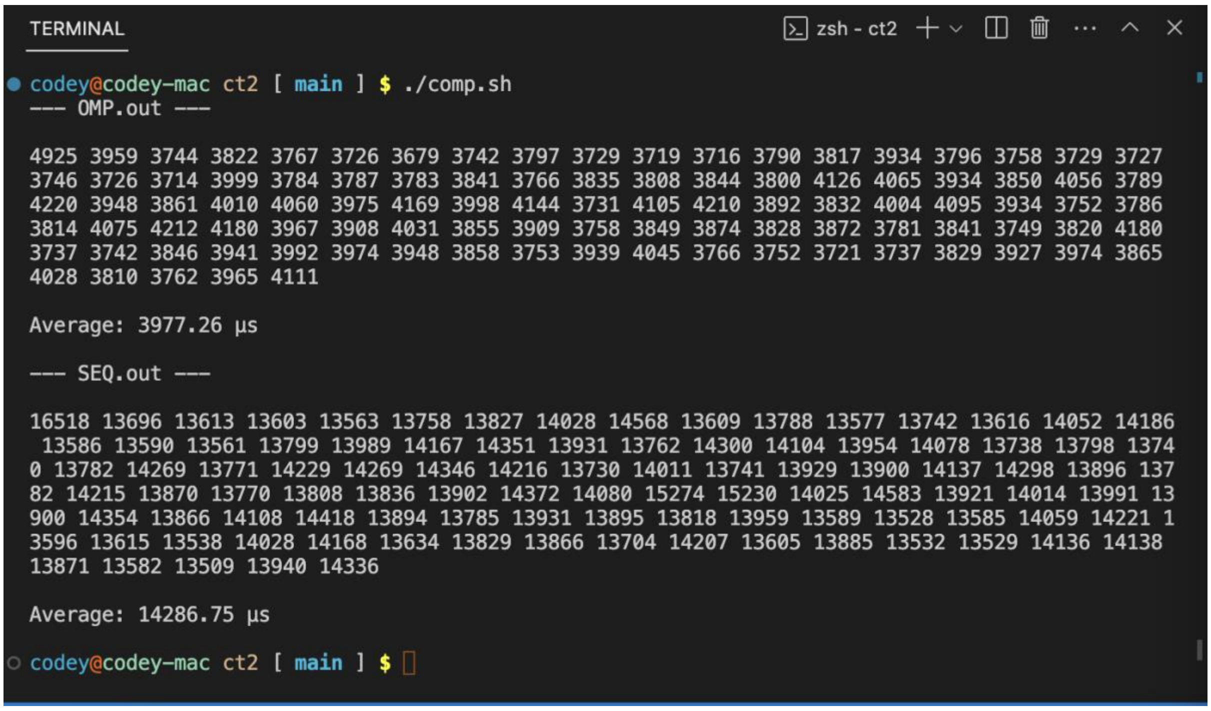
```
codey@codey-mac ct2 [ main ] $ g++-14 -std=c++11 OMP_quickSort.cpp -fopenmp -o OMP.out
codey@codey-mac ct2 [ main ] $ g++-14 -std=c++11 SEQ_quickSort.cpp -o SEQ.out
codey@codey-mac ct2 [ main ] $ ./comp.sh
--- OMP.out ---

244 224 159 180 148 155 166 205 210 164 151 156 182 167 173 191 155 191 170 172 162 172 158 159
 161 154 163 158 160 161 155 159 163 167 174 181 227 178 166 152 212 198 158 203 150 206 157 15
9 163 169 151 160 199 168 174 222 171 197 180 157 192 179 155 163 168 168 158 159 170 167 161 1
84 170 154 194 227 182 160 177 178 178 172 183 170 206 181 173 175 146 193 170 155 171 173 168
158 165 199 159 158

Average: 179.88 µs

--- SEQ.out ---

176 128 111 104 94 94 94 95 95 93 94 94 107 93 99 94 101 94 109 94 94 93 111 97 94 94 101 93 94
 94 97 121 94 94 98 94 95 96 94 94 109 99 93 94 98 95 96 93 112 99 94 95 94 94 94 97 96 93 96 9
9 101 93 97 95 94 93 99 110 99 97 93 98 94 125 97 97 96 95 107 105 95 95 101 98 95 94 96 95 102
 94 113 129 94 96 102 96 99 96 102 110

Average: 102.27 µs

codey@codey-mac ct2 [ main ] $
```

- 20,000 (OMP is now more than twice as fast)



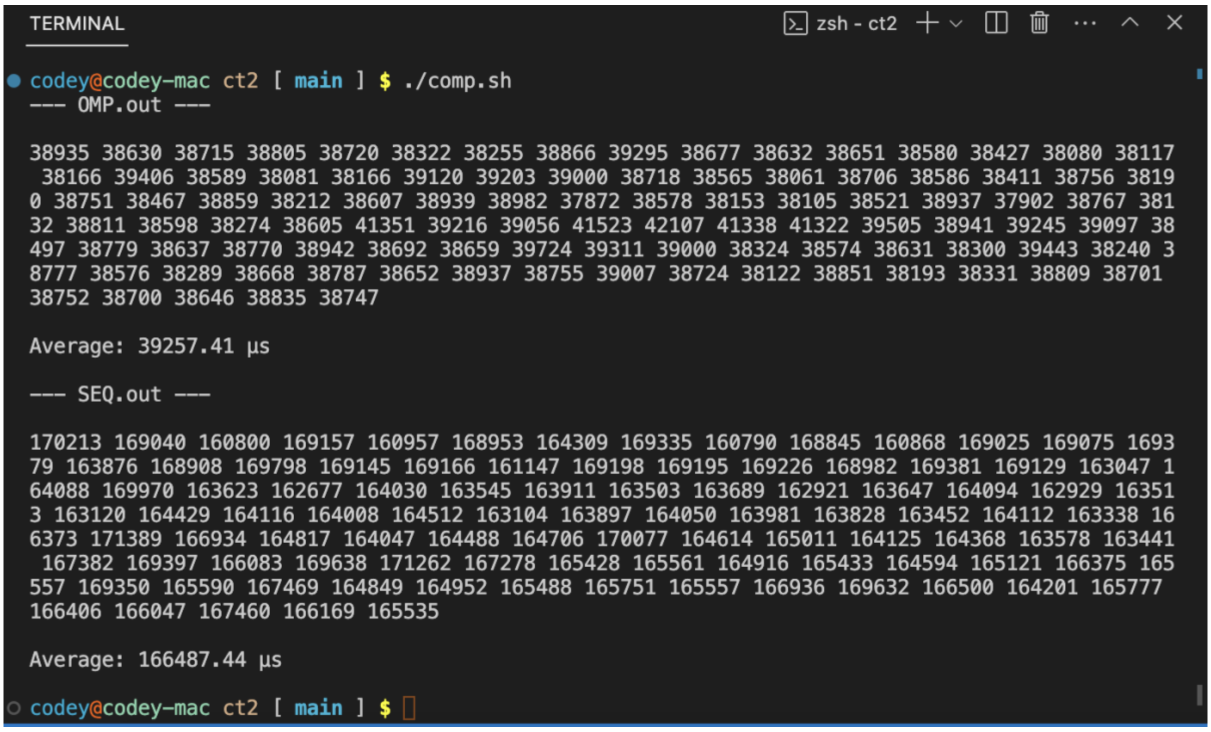
```
codey@codey-mac ct2 [ main ] $ ./comp.sh
--- OMP.out ---

696 517 497 544 508 511 520 513 488 505 540 525 559 514 580 519 511 509 527 490 518 493 503 478
 536 493 486 509 539 495 518 483 476 518 499 486 501 505 481 499 544 499 484 476 526 489 499 48
0 532 492 491 518 499 577 568 498 468 541 538 515 537 500 474 494 514 486 483 522 465 524 525 5
11 479 533 568 534 499 519 540 483 480 508 495 517 475 467 498 485 490 486 499 519 509 517 508
513 538 510 485 479

Average: 529.35 µs

--- SEQ.out ---

2103 1620 1291 1189 1130 1127 1128 1132 1133 1148 1163 1150 1141 1147 1140 1144 1156 1137 1230
1266 1178 1168 1184 1132 1127 1130 1146 1131 1143 1130 1128 1127 1132 1139 1141 1129 1135 1132
1138 1139 1164 1146 1141 1249 1178 1127 1134 1156 1131 1130 1131 1155 1125 1142 1125 1137 1126
1125 1137 1213 1237 1143 1240 1131 1260 1263 1169 1138 1128 1128 1128 1158 1133 1126 1131 1151
1218 1250 1265 1222 1171 1129 1220 1127 1171 1129 1132 1142 1126 1150 1135 1147 1131 1133 1252
1310 1139 1168 1134 1126

Average: 1240.26 µs

codey@codey-mac ct2 [ main ] $
```

- 200,000 (OMP has an incredible jump, at more than 3 times faster)



```
● codey@codey-mac ct2 [ main ] $ ./comp.sh
--- OMP.out ---

4925 3959 3744 3822 3767 3726 3679 3742 3797 3729 3719 3716 3790 3817 3934 3796 3758 3729 3727
3746 3726 3714 3999 3784 3787 3783 3841 3766 3835 3808 3844 3800 4126 4065 3934 3850 4056 3789
4220 3948 3861 4010 4060 3975 4169 3998 4144 3731 4105 4210 3892 3832 4004 4095 3934 3752 3786
3814 4075 4212 4180 3967 3908 4031 3855 3909 3758 3849 3874 3828 3872 3781 3841 3749 3820 4180
3737 3742 3846 3941 3992 3974 3948 3858 3753 3939 4045 3766 3752 3721 3737 3829 3927 3974 3865
4028 3810 3762 3965 4111

Average: 3977.26 µs

--- SEQ.out ---

16518 13696 13613 13603 13563 13758 13827 14028 14568 13609 13788 13577 13742 13616 14052 14186
 13586 13590 13561 13799 13989 14167 14351 13931 13762 14300 14104 13954 14078 13738 13798 1374
0 13782 14269 13771 14229 14269 14346 14216 13730 14011 13741 13929 13900 14137 14298 13896 137
82 14215 13870 13770 13808 13836 13902 14372 14080 15274 15230 14025 14583 13921 14014 13991 13
900 14354 13866 14108 14418 13894 13785 13931 13895 13818 13959 13589 13528 13585 14059 14221 1
3596 13615 13538 14028 14168 13634 13829 13866 13704 14207 13605 13885 13532 13529 14136 14138
13871 13582 13509 13940 14336

Average: 14286.75 µs

○ codey@codey-mac ct2 [ main ] $ 
```

- 2000000 (another speed increase with OMP now more than 4 times faster)



```
● codey@codey-mac ct2 [ main ] $ ./comp.sh
--- OMP.out ---

38935 38630 38715 38805 38720 38322 38255 38866 39295 38677 38632 38651 38580 38427 38080 38117
 38166 39406 38589 38081 38166 39120 39203 39000 38718 38565 38061 38706 38586 38411 38756 3819
0 38751 38467 38859 38212 38607 38939 38982 37872 38578 38153 38105 38521 38937 37902 38767 381
32 38811 38598 38274 38605 41351 39216 39056 41523 42107 41338 41322 39505 38941 39245 39097 38
497 38779 38637 38770 38942 38692 38659 39724 39311 39000 38324 38574 38631 38300 39443 38240 3
8777 38576 38289 38668 38787 38652 38937 38755 39007 38724 38122 38851 38193 38331 38809 38701
38752 38700 38646 38835 38747

Average: 39257.41 µs

--- SEQ.out ---

170213 169040 160800 169157 160957 168953 164309 169335 160790 168845 160868 169025 169075 1693
79 163876 168908 169798 169145 169166 161147 169198 169195 169226 168982 169381 169129 163047 1
64088 169970 163623 162677 164030 163545 163911 163503 163689 162921 163647 164094 162929 16351
3 163120 164429 164116 164008 164512 163104 163897 164050 163981 163828 163452 164112 163338 16
6373 171389 166934 164817 164047 164488 164706 170077 164614 165011 164125 164368 163578 163441
 167382 169397 166083 169638 171262 167278 165428 165561 164916 165433 164594 165121 166375 165
557 169350 165590 167469 164849 164952 165488 165751 165557 166936 169632 166500 164201 165777
166406 166047 167460 166169 165535

Average: 166487.44 µs

○ codey@codey-mac ct2 [ main ] $ 
```