1. Tell what machine you ran this on

I ran the code on my own Macbook pro (2.3GHz dual-core Intel Core i5).

1. Create a table with your results

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| NUM  NUMT | 0 | 1 | 2 | 3 | 4 | 5 |
| 1 | 344.71 | 338.59 | 325.93 | 344.31 | 325.27 | 318.01 |
| 2 | 337.51 | 365.60 | 409.06 | 682.43 | 629.65 | 661.28 |
| 4 | 333.73 | 372.41 | 413.05 | 593.78 | 545.88 | 534.73 |
| NUM  NUMT | 6 | 7 | 8 | 9 | 10 | 11 |
| 1 | 331.39 | 339.19 | 330.74 | 328.77 | 324.89 | 331.76 |
| 2 | 632.72 | 702.82 | 645.59 | 664.45 | 625.25 | 664.55 |
| 4 | 536.05 | 622.66 | 533.27 | 571.23 | 539.80 | 617.42 |
| NUM  NUMT | 12 | 13 | 14 | 15 | 16 |  |
| 1 | 330.33 | 321.31 | 325.50 | 342.43 | 328.49 |  |
| 2 | 641.72 | 651.25 | 629.84 | 690.40 | 643.46 |  |
| 4 | 586.57 | 907.19 | 949.64 | 1065.53 | 982.19 |  |

Table 1 Fix#1 Results

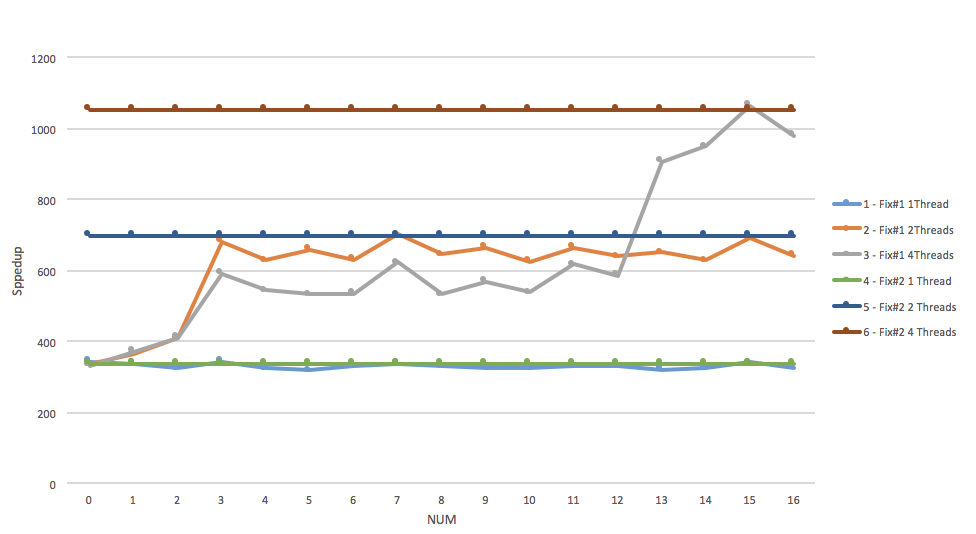
|  |  |
| --- | --- |
| NUMT |  |
| 1 | 338.98 |
| 2 | 700.90 |
| 4 | 1055.57 |

Table 2 Fix#2 Results

Table 1 shows the results using the Fix #1 solution (padding) with 1, 2 and 4 threads. Table 2 shows the results using the Fix #2 solution (private variable) with 1, 2 and 4 threads.

1. Draw a graph.

The X axis is the number of pads and the Y axis is the performance in units of “MegaTimes Compared Per Second”. Curve 1-3 uses the fix #1 solution (padding) with 1, 2, and 4 threads. Curve 4-6 uses the fix #2 solution (private value) with 1, 2, and 4 threads.



1. What patterns are you seeing in the performance?
   1. The speed of “coarse-grained” increases as the number of threads increases (but not will increase infinitely).
   2. The speed of “fine-grained” only slightly increases as the number of threads increase, and even slows down when the number of threads is 16.
   3. For the “coarse-grained” program, the speed using dynamic schedule is faster than static schedule when the number of threads is small (2 and 4), and the speed using static schedule is faster than dynamic schedule when the number of threads is large (8 and 16).
   4. For the “fine-grained” program, the speed using dynamic schedule is always slower than using static schedule.
2. Why do you think it is behaving this way?
   1. It is not hard to understand. Because there are more threads are processing data at the same time, the speed of ‘coarse’ program will increase.
   2. “#pragma omp parallel for” will creates a team of threads from the thread pool and divides the for-loop passes up among those threads. The process of creating and distributing threads are executed 200\*100 times in the ‘fine’ version and only 200 times in the ‘coarse’ version. Obviously it will cost much more time on creating and distributing threads in the ‘fine’ version than in the ‘coarse’ version. In both ‘fine’ and ‘coarse’ program, the number of bodies is set to 100 and the parallel programing don’t really save a lot time. Especially in the ‘fine’ version, the time saved may be covered by the cost of creating and distributing threads.
   3. Dynamic scheduling is better when the iteration may take very different amounts of time. The dynamic scheduling doesn’t save so much time in the ‘coarse’ version because the iteration may take very similar amounts of time. However, there is some overheads to dynamic scheduling. When the number of threads is small, the overheads are less than the time saved by dynamic scheduling and when the number of threads is getting larger (8 and 16), the overheads are more than the time saved.
   4. As the time cost on creating and distributing thread is too much in the ‘fine’ version, there is no obvious speedup using more threads. The overheads of dynamic scheduling make it slower comparing to static scheduling.