Deqing Qu
qud@oregonstate.edu

1. Tell what machine you ran this on

   I ran the code on Flip2.

2. Create a table with your results

   Table 1 shows the raw data of all experiments. In Table 1, the first column shows which program is executed. For example, “Coarse+Static+1Thread” means the running program is the ‘coarse-grained parallelism’ program with static schedule using 1 thread. Each program is executed 5 times and all raw results are saved in Table 1. Other cells show the speed in units of “MegaBodies Compared Per Second”.

   Table 2 shows the average of the raw data in Table1. In Table 2, the first row shows how many threads are used and the first column shows which program is run.

Coarse+Static+1Thread	9.698505	9.688897	9.685148	9.671711	9.685338
Coarse+Static+2Thread	18.957109	18.86906	18.902381	18.915978	18.769959
Coarse+Static+4Thread	36.198225	36.411964	36.196487	36.292568	36.329673
Coarse+Static+8Thread	64.139873	64.995322	64.455246	64.788446	64.527855
Coarse+Static+16Thread	72.391805	73.381285	74.494753	73.257948	74.103968
Coarse+Dynamic+1Thread	9.684014	9.664761	9.687152	9.61	9.670293
Coarse+Dynamic+2Thread	24.976186	24.837047	24.168928	25.1424	24.748808
Coarse+Dynamic+4Thread	39.901699	38.640633	39.693684	38.799342	39.822513
Coarse+Dynamic+8Thread	57.09551	58.460174	58.491893	56.321232	58.772416
Coarse+Dynamic+16Thread	66.602477	67.564179	66.521863	66.935555	67.14147
Fine+Static+1Thread	9.269249	9.259642	9.225926	9.216211	9.262445
Fine+Static+2Thread	13.298901	13.605212	13.479351	13.450713	13.891476
Fine+Static+4Thread	18.298517	18.270255	18.237305	18.224088	18.316177
Fine+Static+8Thread	16.468409	16.476834	16.556433	16.633249	16.494057
Fine+Static+16Thread	8.236913	8.675417	8.778599	8.644933	8.724847
Fine+Dynamic+1Thread	6.712297	6.701518	6.63432	6.670818	6.687786
Fine+Dynamic+2Thread	7.887356	7.381963	7.89607	7.684492	7.821564
Fine+Dynamic+4Thread	9.172953	9.177053	8.679947	9.070675	8.983213
Fine+Dynamic+8Thread	8.16635	8.171674	8.49829	8.14961	8.09906
Fine+Dynamic+16Thread	6.185383	6.14015	6.022579	6.098527	5.860883

Table 1. Raw Data

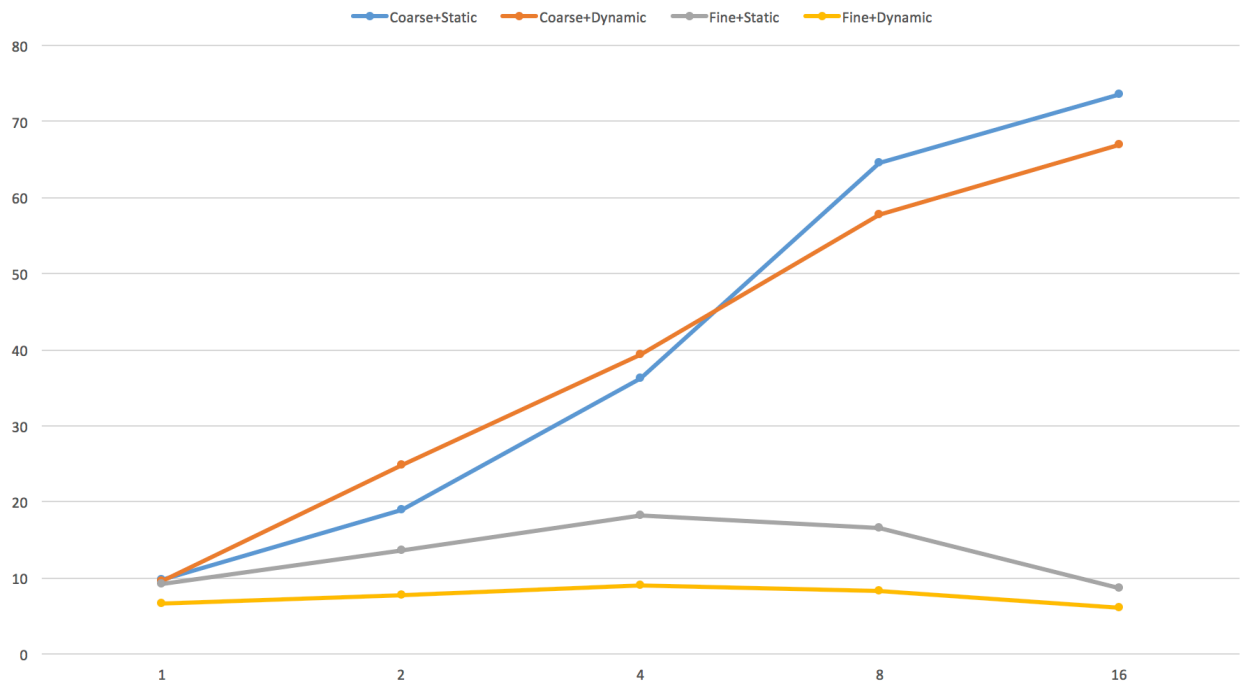
Deqing Qu
qud@oregonstate.edu

Threads Number	1	2	4	8	16
Coarse+Static	9.6859198	18.8828974	36.2857834	64.5813484	73.5259518
Coarse+Dynamic	9.663244	24.7746738	39.3715742	57.828245	66.9531088
Fine+Static	9.2466946	13.5451306	18.2692684	16.5257964	8.6121418
Fine+Dynamic	6.6813478	7.734289	9.0167682	8.2169968	6.0615044

Table 2. Average Data

3. Draw a graph.

The X axis is the number of threads and the Y axis is the performance in units of “MegaBodies Compared Per Second”. The different color lines indicate the different combinations of the programs and schedules.



4. What patterns are you seeing in the speeds?

(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

Deqing Qu
qud@oregonstate.edu

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.

5. 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.