Nick Pepperling
4/21/2014
CS475

# HW1: Numeric Integration with OpenMP - Writeup

**1.)** Machine Configuration

I ran this on the machine intel gave us for our senior design project. It contains:

- Intel Xeon E5-2690
- 8-Cores @ 2.90GhZ
- Hyperthreaded (16 total cores)
- 20Mb CPU cache
- 64 GB RAM
- SSD

**2.)** Show the performances you achieved in tables and graphs?

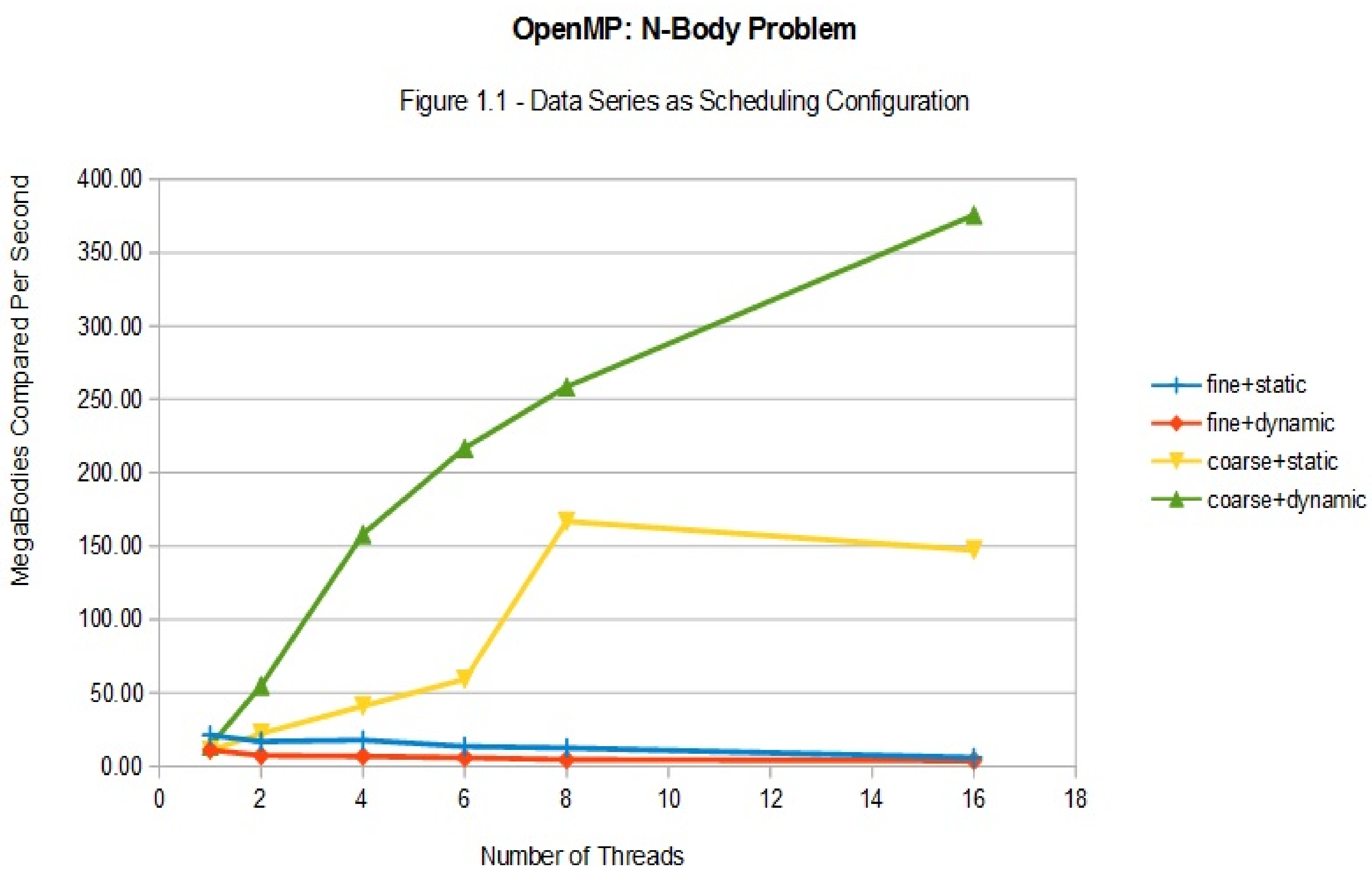
**Table Data for Multiple Run Throughs**

| | **1** | **2** | **4** | **6** | **8** | **16** |
|---|---|---|---|---|---|---|
| **Coarse+Static** | 21.23 | 16.81 | 17.79 | 13.49 | 12.51 | 5.91 |
| **Coarse+Dynamic** | 11.15 | 7.27 | 6.82 | 5.83 | 4.50 | 3.86 |
| **Fine+Static** | 10.74 | 22.40 | 41.03 | 59.23 | 166.86 | 147.32 |
| **Fine+Dynamic** | 13.97 | 54.94 | 158.14 | 216.86 | 258.65 | 375.58 |

The above table shows the scheduling configuration in the leftmost column and the number of threads in the topmost row (NUMTHREADS). The data incident to each row-column pair is the calculated number of MegaBody Comparisons per Second made; calculated via:

$$\frac{NUMBODIES*NUMBODIES*NUMSTEPS}{(EndTime - StartTime)/1000000}$$

**3.)** Graph



**4.)** What patterns are you seeing in the speeds?

The most obvious pattern would be the difference between fine and coarse grain parallelism. Both static and dynamic scheduling types under fine grain parallelism run at a fraction of the speed than that of coarse grain. Among fine grain, we see a slightly faster computational speed when run with static as opposed to dynamic, but each scheduling type demonstrates a slow and steady speed decrease as the number of threads is increased. Among coarse grain, the dynamically schedule run through went the fastest with a continual increase in speed as threads are added. Meanwhile, the statically scheduled runs significantly faster than both fine grain trails, however it seems to lose performance after 8 threads.

**5.)** Why do you think it is behaving this way?

One clear observation is the nature of the problem, in relationship with how the open omp parallel for loop pragmas are breaking up the data. Each loop iterates NUMBODIES (100) times.  Since fine grain parallelism parallelizes the inner-most for loop, all of the overhead required to spawn and manage threads is happening more times by a factor of  NUMBODIES.  This associated overhead is clearly why fine grain parallelism is less computationally efficient.

As for the differences in static and dynamic in the case of coarse grain parallelism, we must first consider the architecture.  The PC configuration used to run the trials contains 8 physical threads and 8 logical threads (hyperthreading).  When looking at the statically  scheduled trial, we see a slight performance drop between 8, and 16 threads.  This is likely due to the fact that static scheduling (with a chunksize of 1) assigns a single iteration (task) to each thread at runtime.  Thus when the number of threads grows, some threads may have to wait on work to become available if their task contains no instructions.  When dynamically scheduled, we see a performance increase all the way  up to the maximum number of available threads on the machine (16).  Dynamic scheduling operates on a master-slave paradigm. (i.e. one thread is assigned to be the master and control work flow to all other threads).  Therefore, we would expect to see performance increase synonymous with the number of threads simply because the overhead of having a reserved master thread would lessen, relatively,  while the master would also be able to delegate work and not have any of the slave threads become idle.