Griffin Maxwell

10/23/17

CSE 5441

Programming Assignment 2 – Results and Summary

1) Serial program at AFFECT_RATE=0.15 and EPSILON=0.15:

In the last programming assignment, the values `AFFECT_RATE=0.15` and `EPSILON=0.15` were chosen to get the total runtime with `testgrid_400_12206` to be 3-6 minutes. The console output from those tests is below:

```
[maxwell.362@beta CSE5441-AMR]$ time ./amr 0.15 0.15 < /class/cse5441/testgrid_400_12206

********************************************************************************
temperature dissipation converged in 43234 iterations
    with max DSV = 0.088556 and min DSV = 0.075273
    AFFECT_RATE = 0.150000;      EPSILON = 0.150000
    Num boxes = 12206;    Num rows = 400;     Num columns = 400

elaspsed convergence loop time:
    using clock():          277260000 clicks (277.260010 s)
    using time():           279 s
    using clock_gettime():  278701.000 ms
********************************************************************************

real   4m38.793s
user   4m37.086s
sys    0m0.259s
```

This serial program was tested again at the same time as testing the pthread parallel versions of the code to make sure that the system load of the stdlinux server was similar to when the tests for Assignment 1 were run. The console output is below:

```
[maxwell.362@eta cse5441_lab1]$ time amr 0.15 0.15 < /class/cse5441/testgrid_400_12206

********************************************************************************
temperature dissipation converged in 43234 iterations
    with max DSV = 0.088556 and min DSV = 0.075273
    AFFECT_RATE = 0.150000;      EPSILON = 0.150000
    Num boxes = 12206;    Num rows = 400;     Num columns = 400
```

```
elaspsed convergence loop time:
    using clock():          376680000 clicks (376.679993 s)
    using time():           378 s
    using clock_gettime():  378039.000 ms
********************************************************************************

real   6m18.146s
user   6m16.561s
sys    0m0.220s
```

The timing results of these two tests are not very close, indicating that the system loads during the most recent tests were a bit higher than the previous tests. Good comparisons can still be made between the timings of the serial and newly parallelized versions of the code, but the parallelized timings should be considered slightly inflated because of these system loads.

2) Parallel Program Part One – Disposable pthreads

The first parallelization of the serial code from the last assignment was made using disposable pthreads. The following timing results were gathered with `AFFECT_RATE=0.15`, `EPSILON=0.15`, and the input grid `testgrid_400_12206` to compare with the previous serial tests. As can be seen by the data in the following pages, the number of iterations to converge for each test matches that of the original serial program, demonstrating that the underlying algorithm has not changed. Timing results for this parallelized version were gathered by running the program many times on the same input data and specifying the number of threads as a command line argument: 2, 8, 16, and 32 threads.

<u>2 threads</u>

```
[maxwell.362@eta cse5441_lab2]$ time disposable 0.15 0.15 2 <
/class/cse5441/testgrid_400_12206

********************************************************************************
temperature dissipation converged in 43234 iterations
    with number of (disposable) pthreads = 2
    with max DSV = 0.088556 and min DSV = 0.075273
    AFFECT_RATE = 0.150000;      EPSILON = 0.150000
    Num boxes = 12206;    Num rows = 400;     Num columns = 400
```

```
elaspsed convergence loop time:
    using clock():          492120000 clicks (492.119995 s)
    using time():           335 s
    using clock_gettime():  335749.000 ms
********************************************************************************

real   5m35.856s
user   7m53.870s
sys    0m18.355s
```

8 threads

```
[maxwell.362@eta cse5441_lab2]$ time disposable 0.15 0.15 8 <
/class/cse5441/testgrid_400_12206

********************************************************************************
temperature dissipation converged in 43234 iterations
    with number of (disposable) pthreads = 8
    with max DSV = 0.088556 and min DSV = 0.075273
    AFFECT_RATE = 0.150000;      EPSILON = 0.150000
    Num boxes = 12206;    Num rows = 400;     Num columns = 400

elaspsed convergence loop time:
    using clock():          766450000 clicks (766.450012 s)
    using time():           632 s
    using clock_gettime():  631738.000 ms
********************************************************************************

real   10m31.859s
user   11m13.103s
sys    1m33.455s
```

16 threads

```
[maxwell.362@eta cse5441_lab2]$ time disposable 0.15 0.15 16 <
/class/cse5441/testgrid_400_12206

********************************************************************************
temperature dissipation converged in 43234 iterations
    with number of (disposable) pthreads = 16
    with max DSV = 0.088556 and min DSV = 0.075273
    AFFECT_RATE = 0.150000;      EPSILON = 0.150000
    Num boxes = 12206;    Num rows = 400;     Num columns = 400

elaspsed convergence loop time:
    using clock():          939010000 clicks (939.010010 s)
    using time():           771 s
    using clock_gettime():  770192.000 ms
********************************************************************************

real   12m50.317s
user   12m38.083s
sys    3m1.036s
```

<u>32 threads</u>

```
[maxwell.362@eta cse5441_lab2]$ time disposable 0.15 0.15 32 <
/class/cse5441/testgrid_400_12206

********************************************************************************
temperature dissipation converged in 43234 iterations
    with number of (disposable) pthreads = 32
    with max DSV = 0.088556 and min DSV = 0.075273
    AFFECT_RATE = 0.150000;      EPSILON = 0.150000
    Num boxes = 12206;    Num rows = 400;     Num columns = 400

elaspsed convergence loop time:
    using clock():          1193260000 clicks (1193.260010 s)
    using time():           972 s
    using clock_gettime():  972265.000 ms
********************************************************************************

real   16m12.457s
user   14m4.530s
sys    5m48.865s
```

3) Parallel Program Part Two – Persistent threads

The second parallelization of the serial code from the last assignment was made using persistent pthreads. Otherwise, all the same testing parameters as “Parallel Program Part One – Disposable pthreads” were used. The console output of each of these tests is below:

<u>2 threads</u>

```
[maxwell.362@eta cse5441_lab2]$ time persistent 0.15 0.15 2 <
/class/cse5441/testgrid_400_12206

********************************************************************************
temperature dissipation converged in 43234 iterations
    with number of (disposable) pthreads = 2
    with max DSV = 0.088556 and min DSV = 0.075273
    AFFECT_RATE = 0.150000;      EPSILON = 0.150000
    Num boxes = 12206;    Num rows = 400;     Num columns = 400

elaspsed convergence loop time:
    using clock():          538080000 clicks (538.080017 s)
    using time():           430 s
    using clock_gettime():  430171.000 ms
********************************************************************************

real   7m10.297s
```

```
user   8m46.842s
sys    0m11.348s
```

<u>8 threads</u>

```
[maxwell.362@eta cse5441_lab2]$ time persistent 0.15 0.15 8 <
/class/cse5441/testgrid_400_12206

********************************************************************************
temperature dissipation converged in 43234 iterations
    with number of (disposable) pthreads = 8
    with max DSV = 0.088556 and min DSV = 0.075273
    AFFECT_RATE = 0.150000;      EPSILON = 0.150000
    Num boxes = 12206;    Num rows = 400;     Num columns = 400

elaspsed convergence loop time:
    using clock():          537790000 clicks (537.790039 s)
    using time():           292 s
    using clock_gettime():  291579.000 ms
********************************************************************************

real   4m51.749s
user   8m32.864s
sys    0m25.053s
```

<u>16 threads</u>

```
[maxwell.362@eta cse5441_lab2]$ time persistent 0.15 0.15 16 <
/class/cse5441/testgrid_400_12206

********************************************************************************
temperature dissipation converged in 43234 iterations
    with number of (disposable) pthreads = 16
    with max DSV = 0.088556 and min DSV = 0.075273
    AFFECT_RATE = 0.150000;      EPSILON = 0.150000
    Num boxes = 12206;    Num rows = 400;     Num columns = 400

elaspsed convergence loop time:
    using clock():          646710000 clicks (646.710022 s)
    using time():           505 s
    using clock_gettime():  505526.000 ms
********************************************************************************

real   8m25.653s
user   10m13.617s
sys    0m33.213s
```

<u>32 threads</u>

```
[maxwell.362@eta cse5441_lab2]$ time persistent 0.15 0.15 32 <
/class/cse5441/testgrid_400_12206

********************************************************************************
```

```
temperature dissipation converged in 43234 iterations
    with number of (disposable) pthreads = 32
    with max DSV = 0.088556 and min DSV = 0.075273
    AFFECT_RATE = 0.150000;      EPSILON = 0.150000
    Num boxes = 12206;     Num rows = 400;     Num columns = 400

elaspsed convergence loop time:
    using clock():          769850000 clicks (769.849976 s)
    using time():           582 s
    using clock_gettime():  581613.000 ms
********************************************************************************

real   9m41.839s
user   11m40.876s
sys    1m9.115s
```

4) Summary of Timing Results:

General timing results from this experiment are specified in the table below. Sections 1 – 3 above detail the parameters used to run the code, as well as the full console output of each run. "Real", clock(), and time() times are highlighted in this table because they each seem to come from a different time source, or at least give a different perspective on how the execution time of the programs can be measured.

Number of threads	**Serial**			**Disposable pthreads**			**Persistent pthreads**		
	"real"	clock()	time()	"real"	clock()	time()	"real"	clock()	time()
1	4m38.793s	277.26 s	279 s	-	-	-	-	-	-
2	-	-	-	5m35.856s	492.1 s	335 s	7m10.297s	538.1 s	430 s
8	-	-	-	10m31.859s	766.5 s	632 s	4m51.749s	537.8 s	292 s
16	-	-	-	12m50.317s	939.0 s	771 s	8m25.653s	646.7 s	505 s
32	-	-	-	16m12.457s	1193.3 s	972 s	9m41.839s	769.8 s	582 s

5) Questions

- As can be seen by the table in section 4, surprisingly, the serial program seemed to run faster than the parallelized versions of the code. This may be an anomaly due to higher system loads on the stdlinux servers, as discussed in section 1. However, stdlinux may not be entirely to blame. Seeing how the "system" times for the parallel programs shot up dramatically compared to the serial program might be a sign that the overhead of asking the operating system for threads might be significant enough to outweigh the benefits of parallelizing the dissipation calculations. This is further evidenced by the fact that the "system" times of the disposable thread version of the code were often around a minute greater than those of the persistent thread version. Since the disposable thread version requested many more threads overall, this is a clear indication that on these stdlinux systems, requesting and using pthreads can be an expensive operation. Further tests on a supercomputer or different systems with a lighter load can confirm that whether it is the overhead of pthreads or an algorithmic difference that is causing this increase in execution time.
- Based on the averages of the timings of the disposable and persistent versions of the programs, 2 threads seem to be the most effective number of threads. Since the average times for 8 threads were fairly close to the times for 2 threads, the actual most effective number of threads might be somewhere in the middle, but more tests at all numbers of threads 2-8 would have to be completed to determine this for sure.
- The persistent version of the code was definitely more effective than the disposable version because the run times were almost always shorter. For instance, for 32 threads, the disposable version took a whopping 16 minutes to run, while the persistent version only took about 10 minutes. This is likely due to the fact that the disposable version wasted a lot of extra time trying to communicate with the operating system to create threads and then soon after destroying them. The persistent version only requested threads once and used barriers to synchronize, so there was much less overhead communicating with the operating system.
- Overall, these results conflicted with my expectations. Before running these tests, I expected both parallel versions to be significantly faster, on the order of N times faster, where N is the number of threads. I expected the impact of creating threads to be insignificant, and the threads to work in parallel all the time, so that threads almost never spent time waiting in the queue. I also expected

the timing difference between disposable and persistent threads to be marginal, because I expected the thread creation and scheduling to add very little time overall. Instead, the results showed that more threads often times took significantly more time than having fewer threads, and that persistent threads are noticeably faster than disposable threads.

- The only sorts of “anomalies” in the timing results were where the timing of the disposable or persistent program changed directions. For instance, the persistent program with 8 threads ran a lot faster than the persistent program with 2 threads, yet this trend did not continue because the persistent program with 16 and 32 threads both ran the slowest. Since the scheduling and executing of programs and their threads is very dependent on the operating system the program is running on, these anomalies can be explained away as “luck of the draw” as the operating system is trying to schedule all the threads from the many users trying to use the system at a given time.
- The three time stats given by the time command (real, user, and sys) seem to be the best method to figure out the timing of the parallel programs. I say this because user time gives a good sense of the sum amount of processor time the threads actually spent running, sys time gives a good sense of how long the threads were waiting to be created or waiting in the queue, and contrasting these two with real time gives a good sense of how much server congestion is affecting the prompt running of the program. If user time divided by the number of threads plus the sys time is not close to the real time, then it is likely that there is a lot of congestion on the system that could be affecting timing results. This is what I expected, because the time command is designed to be the one user-friendly command that gives a programmer all the information he or she needs about timing. Meanwhile, the other timing methods seem to be based on these data structures accumulating time in the background, and it is harder to verify that they are counting time in the way the user expects.
- As discussed in section 2, the number of iterations to converge for each program under each number of threads exactly matches the serial program, indicating that the underlying algorithm and dissipation calculations have not changed after parallelization.