Shiny Patel CSE 5441 Lab 3 Report

Timing Results (pthreads vs OpenMP)

Disposable threads on testgrid_400_12206 with epsilon and affect rate = .1

#threads	posix clock (pthreads)	posix clock (OpenMP)	clock ticks (pthreads)	clock ticks (OpenMP)	time (pthreads)	time (OpenMP)
2	47521028.00	49592359.00	76260000	98110000	47	49
8	46694213.00	46194953.00	94490000	96130000	47	47
16	73580036.00	64955902.00	145090000	99620000	74	65
32	305715125.00	73365587.00	389580000	106410000	305	73

(the columns below are a continuation from the above table for the same runs)

#threads	real (pthreads)	real (OpenMP)	user (pthreads)	user (OpenMP)	sys (pthreads)	sys (OpenMP)
2	47.63	49.61	72.56	97.99	3.71	0.14
8	46.71	46.21	78.69	92.82	15.81	3.32
16	73.59	64.97	88.19	94.89	56.91	4.74
32	305.73	73.38	102.30	98.70	287.29	7.73

#iterations: 75197, max DSV: 0.086671 and min DSV: 0.078004 for every run

The slowest of the four programs was the run with 32 threads. The value of epsilon and affect rate were selected as .05 and .05 respectively to give the following result:

```
---------------------------------------------------------------------
elapsed convergence loop time (posix clock): 193994379.00
elapsed convergence loop time (clock ticks): 280040000
elapsed convergence loop time (clock seconds): 280.04
elapsed convergence loop time (time): 194

number of iterations: 197312
max DSV: 0.085244 and min DSV: 0.080982
epsilon: 0.05 and affect rate: 0.05
---------------------------------------------------------------------
real 194.01
user 259.40
sys 20.66
```

The above run took real time: 194 s (i.e. 3 min and 14 s)

Persistent threads on testgrid_400_12206 with epsilon and affect rate = .1

#threads	posix clock (pthreads)	posix clock (OpenMP)	clock ticks (pthreads)	clock ticks (OpenMP)	time (pthreads)	time (OpenMP)
2	40623443.00	47544371.00	67820000	94420000	41	48
8	41373636.00	59982159.00	85280000	96830000	41	60
16	61756259.00	62442619.00	99110000	102410000	62	62
32	82449914.00	71913331.00	107380000	114250000	82	72

(the columns below are a continuation from the above table for the same runs)

#threads	real (pthreads)	real (OpenMP)	user (pthreads)	user (OpenMP)	sys (pthreads)	sys (OpenMP)
2	40.64	47.56	66.07	94.31	1.76	0.12
8	41.39	60.00	75.38	92.53	9.91	4.32
16	61.77	62.46	80.01	95.37	19.11	7.06
32	82.46	71.93	81.34	101.02	26.05	13.25

#iterations: 75197, max DSV: 0.086671 and min DSV: 0.078004 for every run

The slowest of the four programs was the run with 32 threads. The value of epsilon and affect rate were selected as .04 and .04 respectively to give the following result:

```
---------------------------------------------------------------------
elapsed convergence loop time (posix clock): 264660959.00
elapsed convergence loop time (clock ticks): 426920000
elapsed convergence loop time (clock seconds): 426.92
elapsed convergence loop time (time): 265

number of iterations: 278494
max DSV: 0.085141 and min DSV: 0.081735
epsilon: 0.04 and affect rate: 0.04
---------------------------------------------------------------------
real 264.68
user 374.40
sys 52.53
```

The above run took real time: 265 s (i.e. 4 min and 25 s)

The program always created the requested number of threads for every run for both disposable (during every iteration) and persistent versions.

If the requested number of threads are not created then the program will print the actual number of threads created.

(The program is designed to check the number of threads created during every iteration for the disposable version and will print the actual number of threads along with the iteration number for which the requested number of threads are not created.)

NOTE: The wall clock time is the 'real' time in the above results and all times are in seconds.

Q&A

- **How would you characterize the computational work-load of our sample program?**
  The work-load in the program was distributed using block distribution.

- **Which threading mechanism, pthreads or OpenMP, provided the best results in your case?**
  OpenMP provided best results (in terms of real, user and sys times) for both disposable and persistent versions for the slowest run i.e. run with 32 threads. For all other runs, OpenMP provided better or equivalent results in terms of real and sys times.

- **Which threading mechanism, pthreads or OpenMP, was the easiest to implement?**
  OpenMP was the easiest to implement. There was no need to create or destroy threads and setup the workload distribution.

- **Which threading mechanism, pthreads or OpenMP, would you be most likely to select for a similar application?**
  I would select OpenMP due to its easier implementation.

- **Under what circumstances would you speculate that the other mechanism, pthreads or OpenMP, would be preferable?**
  pthreads would be preferable when the program required maximal control over thread management (such as thread create, join, locks and mutexes etc.) since it provides more flexibility than OpenMP.

- **Were there any surprises you encountered in this exercise?**
  There were no surprises. Parallelization using OpenMP was extremely straightforward compared to using pthreads.