IPC Programming Assignment 2

jaspreetkaur.1

1. Objective

The objective of the project is to find out if parallelizing the serial version of our program done as part of lab1, gives any performance improvement, and i.e. takes lesser time to converge. We experiment with two different thread models- disposable and persistent and try to compare their performance for different thread numbers. Thread number varied from 2, 4, 8, 16, and 32.

2. Summary of results

All values computed with epsilon = 0.1 and affect_rate = 0.1

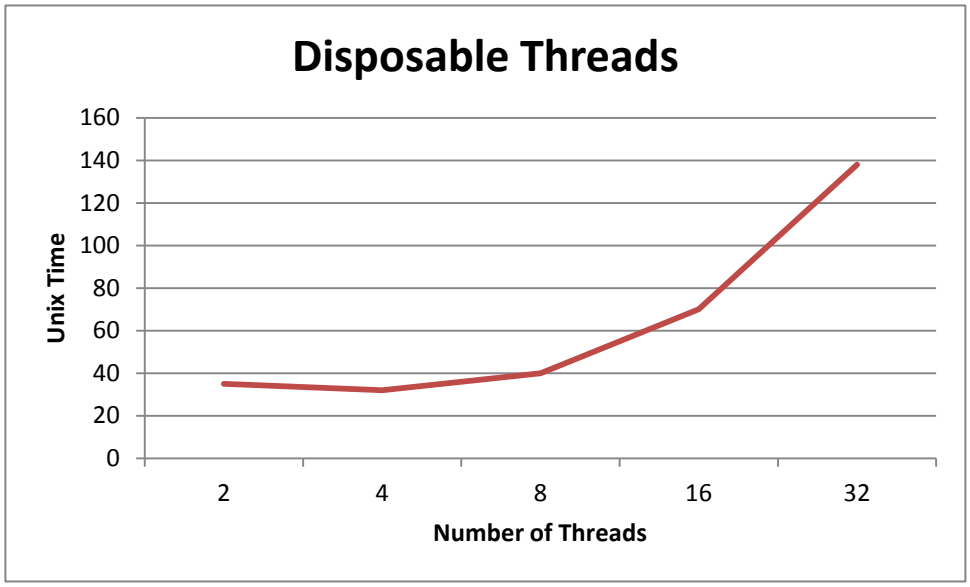
Serial:

Clock Time	Chrono Time	Unix time
41400000	41523436.000000	41

Disposable Threads

Number of Threads	Clock Time	Chrono Time	Unix time
2	53920000	34545634.000000	35
4	59050000	32082763.000000	32
8	70190000	39513884.000000	40
16	121010000	70507009.000000	70
32	195310000	137601934.000000	138

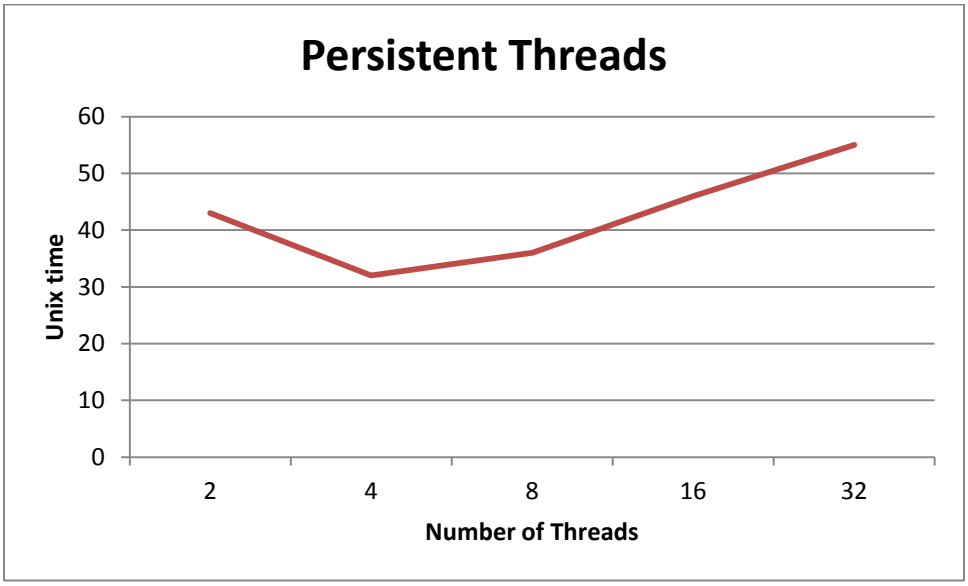
Below is a plot of unix times versus number of threads for epsilon = 0.1 and affect_rate = 0.1.



Persistent Threads:

Number of Threads	Clock Time	Chrono Time	Unix time
2	43790000	42889452.000000	43
4	48990000	32082763.000000	32
8	54460000	36295566.000000	36
16	55980000	46387274.000000	46
32	61240000	54901972.000000	55

Below is a plot of unix times versus number of threads for epsilon = 0.1 and affect_rate = 0.1.

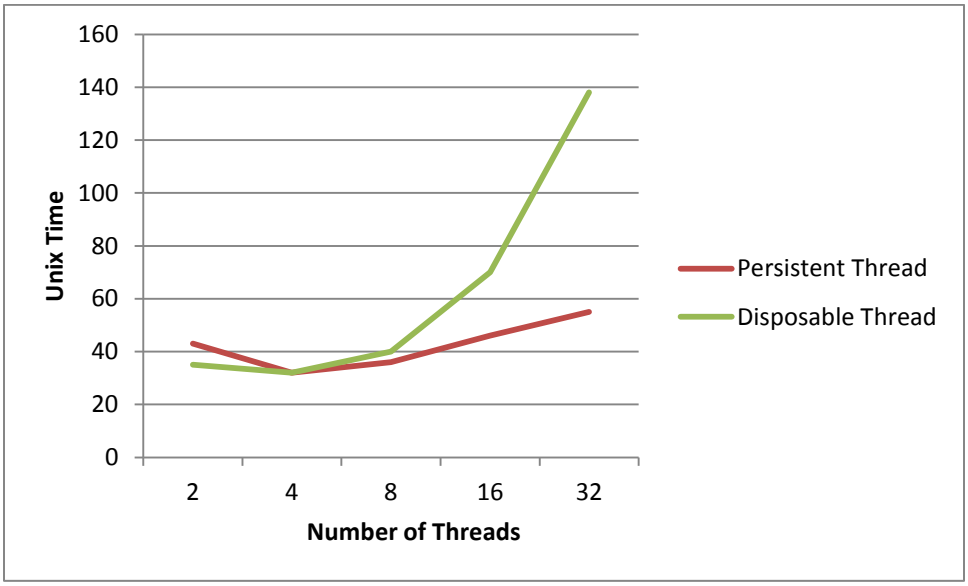


3. Questions:

1. **Did the program perform better sequentially or in parallel?**
   There was no clear winner of who performed better serial or parallel. It depended on the number of threads being used to parallelize the program and the threading strategy in place. Parallel versions of the program performed better than the serial version of the program for thread count 2 to 8. Using 4 threads in the persistent model reduced the execution time in half. It seems that since the stdlinux system has 4 cores the performance is best for 4 threads. For larger number of threads the benefit of parallelizing code gets offset by the overhead required in context switching by the CPU and for threads lesser than 4 the benefit of parallelization gets somewhat offset by thread creation overhead.

2. **Which number of threads was most effective?**
   The best number of threads for running the program is 4. Which is expected as the expectation is that best performance is expected when number of threads created is equal to number of cores.

3. **Which parallel version (disposable or persistent) was most effective?**
   Persistent version of thread creation is more efficient than the disposable version. This is on expected lines as disposable version has an additional overhead of thread creation in every convergence loop. See graph below for comparison:-



4. **How did your results match or conflict with your expectations?**
   The results were on an average as expected, I was however expecting a little more gain in terms of performance i.e. lesser execution time for threads between 2 to 4, but it seems that the gain obtained by parallelizing the thread got offset by the thread creation overhead. For larger number of threads since there was a lot of context switch, so performance degradation was on expected lines.

5. **Were there any unexpected anomalies in the timing information collected ?**
   a. I expected the code performance to be slightly better with 2 threads than it actually was. I expected that the code will speed up by 20-30 % instead of the final 10-15%. I did not realize that the thread creation overhead is so much.
   b. I even expected that more number of threads will lead to faster execution of code but it turned out that overhead was substantial and more than my expectation.
   c. As per my expectations, persistent threads performed better than disposable threads

6. **Which timing methods seem best for parallel programs? How does this compare with your expectations?**
   Unix time was very similar in the time they returned which was wall clock time of execution. CLOCK time was different from what I got for serial program where there was close correlation between CLOCK time and Chrono time. CLOCK time here was substantially larger from the other two as it returns the total clock cycles used by all threads running on different cores. Both clock and Chrono are better to evaluate the threaded programs.

4. Other Insight:-

To find out more about the thread creation overhead, I ran disposable and persistent version for a single thread where master thread would create new thread and execute all the computations in the new thread. And the results gave a clearer picture of the overhead in thread creation.

Execution Method	Time Taken (secs)
Serial	41
Disposable threads	57
Persistent threads	43

real 0m56.420s
user 0m50.380s
sys 0m2.608s
****
There is an additional 3.069 seconds that is used by the System which is I think is for the thread creation overhead like stack allocation, etc.

5. Code Structure :-

1. Used Block distribution to allocate boxes to the threads. Took care of the cases where there are extra boxes when the total number of boxes modulus number of threads is not zero.
2. Time was spent on deciding the access of the variables whether they are to be kept private or global.

6. Results:-

time ./amr_csr_serial 0.1 0.1 < /class/cse5441/testgrid_400_12206

****
Dissipation converged in 75197 iterations,
with max DSV = 0.086671 and min DSV = 0.078004
Affect rate = 0.100000 , epsilon = 0.100000
elapsed convergence loop time(clock) : 41400000
elapsed convergence loop time(time) : 41
elapsed convergence loop time (chrono): 41523436.000000
****

real 0m41.547s
user 0m41.393s
sys 0m0.023s

time ./kaur_jaspreet_disposable.c 0.1 0.1 2 < /class/cse5441/testgrid_400_12206

****

Dissipation converged in 75197 iterations,
with max DSV = 0.086671 and min DSV = 0.078004
Affect rate = 0.100000 , epsilon = 0.100000
elapsed convergence loop time(clock) : 53920000
elapsed convergence loop time(time) : 35
elapsed convergence loop time (chrono): 34545634.000000
****

real 0m34.583s
user 0m50.564s
sys 0m3.373s

time ./kaur_jaspreet_disposable.c 0.1 0.1 4 < /class/cse5441/testgrid_400_12206

****
Dissipation converged in 75197 iterations,
with max DSV = 0.086671 and min DSV = 0.078004
Affect rate = 0.100000 , epsilon = 0.100000
elapsed convergence loop time(clock) : 59050000
elapsed convergence loop time(time) : 35
elapsed convergence loop time (chrono): 35075603.000000
****

real 0m35.097s
user 0m51.912s
sys 0m7.152s

time ./kaur_jaspreet_disposable.c 0.1 0.1 8 < /class/cse5441/testgrid_400_12206
****
Dissipation converged in 75197 iterations,
with max DSV = 0.086671 and min DSV = 0.078004
Affect rate = 0.100000 , epsilon = 0.100000
elapsed convergence loop time(clock) : 70190000
elapsed convergence loop time(time) : 40
elapsed convergence loop time (chrono): 39513884.000000
****

real 0m39.536s
user 0m54.899s
sys 0m15.318s

time ./kaur_jaspreet_disposable.c 0.1 0.1 16 < /class/cse5441/testgrid_400_12206
****

Dissipation converged in 75197 iterations,
with max DSV = 0.086671 and min DSV = 0.078004
Affect rate = 0.100000 , epsilon = 0.100000
elapsed convergence loop time(clock) : 121010000
elapsed convergence loop time(time) : 70
elapsed convergence loop time (chrono): 70507009.000000
****

real 1m10.529s
user 1m5.798s
sys 0m55.238s

time ./kaur_jaspreet_disposable.c 0.1 0.1 32 < /class/cse5441/testgrid_400_12206
****
Dissipation converged in 75197 iterations,
with max DSV = 0.086671 and min DSV = 0.078004
Affect rate = 0.100000 , epsilon = 0.100000
elapsed convergence loop time(clock) : 195310000
elapsed convergence loop time(time) : 138
elapsed convergence loop time (chrono): 137601934.000000
****

real 2m17.624s
user 1m17.162s
sys 1m58.166s

time ./kaur_jaspreet_persistent.c 0.1 0.1 2 < /class/cse5441/testgrid_400_12206

****
Dissipation converged in 75197 iterations,
with max DSV = 0.086671 and min DSV = 0.078004
Affect rate = 0.100000 , epsilon = 0.100000 , number_of_threads = 2
elapsed convergence loop time(clock) : 43790000
elapsed convergence loop time(time) : 43
elapsed convergence loop time (chrono): 42889452.000000
****

real 0m42.911s
user 0m43.120s
sys 0m0.697s

time ./kaur_jaspreet_persistent.c 0.1 0.1 4 < /class/cse5441/testgrid_400_12206

****

Dissipation converged in 75197 iterations,
with max DSV = 0.086671 and min DSV = 0.078004
Affect rate = 0.100000 , epsilon = 0.100000 , number_of_threads = 4
elapsed convergence loop time(clock) : 48990000
elapsed convergence loop time(time) : 32
elapsed convergence loop time (chrono): 32082763.000000
****

real 0m32.104s
user 0m46.113s
sys 0m2.900s

time ./kaur_jaspreet_persistent.c 0.1 0.1 8 < /class/cse5441/testgrid_400_12206
****
Dissipation converged in 75197 iterations,
with max DSV = 0.086671 and min DSV = 0.078004
Affect rate = 0.100000 , epsilon = 0.100000 , number_of_threads = 8
elapsed convergence loop time(clock) : 54460000
elapsed convergence loop time(time) : 36
elapsed convergence loop time (chrono): 36295566.000000
****

real 0m36.317s
user 0m48.090s
sys 0m6.392s

time ./kaur_jaspreet_persistent.c 0.1 0.1 16 < /class/cse5441/testgrid_400_12206

****
Dissipation converged in 75197 iterations,
with max DSV = 0.086671 and min DSV = 0.078004
Affect rate = 0.100000 , epsilon = 0.100000 , number_of_threads = 16
elapsed convergence loop time(clock) : 55980000
elapsed convergence loop time(time) : 46
elapsed convergence loop time (chrono): 46387274.000000
****

real 0m46.434s
user 0m47.291s
sys 0m8.741s

time ./kaur_jaspreet_persistent.c 0.1 0.1 32 < /class/cse5441/testgrid_400_12206

****
Dissipation converged in 75197 iterations,
with max DSV = 0.086671 and min DSV = 0.078004
Affect rate = 0.100000 , epsilon = 0.100000 , number_of_threads = 32
elapsed convergence loop time(clock) : 61240000

elapsed convergence loop time(time) : 55
elapsed convergence loop time (chrono): 54901972.000000
****

real 0m54.923s
user 0m48.517s
sys 0m12.742s