Programming Assignment 2

Name: Shuangsheng Lou
Email: lou.125@osu.edu

Introduction

In this programming, I implemented the simplified AMR dissipation problem by using two versions of posix threads. I listed my program output and timing results for the testgrid_400_12206 test file from my sequential program and both versions of my pthreads parallel program. Other than execution times, they achieve the same results. In addition, the line graph was used to show experimental results vividly. According to the output data, I did some careful analyses for the timing results and put forward my assumption for the best number of threads for the program.

Timing Result

Input file: testgrid_400_12206
Parameters: epsilon is 0.1, affect_rate is 0.1

(1) Serial program:

Iteration: 75272
Max DSV: 0.086672
Min DSV: 0.078004
Clock: 172000000
Time: 172.000000
Realtime: 172.00000

(2) Disposable threads：

2 threads：
Iteration: 75272
Max DSV: 0.086672

Min DSV: 0.078004
Clock: 186720000
Time: 105.000000
Realtime: 105.00000

4 threads:

Iteration: 75272
Max DSV: 0.086672
Min DSV: 0.078004
Clock: 196000000
Time: 83.000000
Realtime: 83.00000

8 threads：

Iteration: 75272
Max DSV: 0.086672
Min DSV: 0.078004
Clock: 212820000
Time: 89.000000
Realtime: 89.00000

16 threads：

Iteration: 75272
Max DSV: 0.086672
Min DSV: 0.078004
Clock: 266630000
Time: 104.000000
Realtime: 104.00000

24 threads:

Iteration: 75272
Max DSV: 0.086672
Min DSV: 0.078004
Clock: 330420000
Time: 134.000000
Realtime: 134.00000

36 threads:

Iteration: 75272
Max DSV: 0.086672
Min DSV: 0.078004
Clock: 383960000
Time: 165.000000
Realtime: 165.00000

(3) Persistent threads:

2 threads:

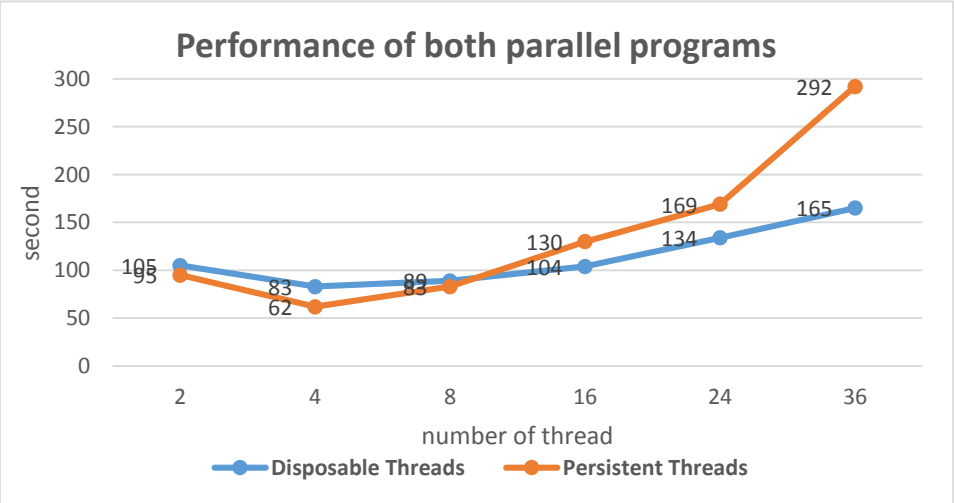
Iteration: 75272

Max DSV: 0.086672
Min DSV: 0.078004
Clock: 186290000
Time: 95.000000
Realtime: 95.00000
4 threads:
Iteration: 75272
Max DSV: 0.086672
Min DSV: 0.078004
Clock: 211500000
Time: 62.000000
Realtime: 62.00000
8 threads:
Iteration: 75272
Max DSV: 0.086672
Min DSV: 0.078004
Clock: 254670000
Time: 83.000000
Realtime: 83.00000
16 threads:
Iteration: 75272
Max DSV: 0.086672
Min DSV: 0.078004
Clock: 346080000
Time: 130.000000
Realtime: 130.00000
24 threads:
Iteration: 75272
Max DSV: 0.086672
Min DSV: 0.078004
Clock: 432280000
Time: 169.000000
Realtime: 169.00000
36 threads:
Iteration: 75272
Max DSV: 0.086672
Min DSV: 0.078004
Clock: 542850000
Time: 292.000000
Realtime: 292.00000

Summarizing Timing Result

1. Both versions' performance of the parallel program

In order to have a vivid expression of both versions' performance of this parallel program, I did a graph below by the output time data. From the graph, you could easily see that by using different number of threads, the parallel performance of this program differs from each other. When the number of threads increases from 2, the performance of both versions firstly improves and then reduces. Because the running real time of sequential program is 172 seconds, so the performance of using disposable threads in this program is better than sequential program. However, for the persistent threads, the program using 2, 4, 8, 16 or 24 threads in parallel performs better than the sequential program but the program using 36 threads in parallel performs worse than the sequential program.



Graph 1

2. Related questions

(a) Did this program perform better sequentially or in parallel?

As I discussed in former part, in most cases, this program performs better in parallel, except the case that uses 36 persistent threads.

(b) Which number of threads was most effective?

In this program, using 4 threads was most effective.

(c) Which parallel version was most effective?

If we just look the best performance of both versions, using a persistent threads model was more effective. When the number of threads is 2, 4 and 8, the pthread parallel version of using a persistent threads model was more effective than using a disposable threads model. However, when the number of threads is 16, 24 and 36, the pthread parallel version of using a disposable threads model was more effective.

(d) How did your results match or conflict with your expectations?

At first, the results of final convergence values and number of iterations matched my expectations. No matter it was executed sequentially or in parallel, these results remained the same. However, some parts of parallel results conflicted with my expectations. In my former perspective, if we use more threads in parallel program, we could get better results. In fact, when you use too many threads, the performance of the program would reduce.

(e) Were there any unexpected anomalies in the timing information collected?

Yes. When I used 36 persistent threads to run this program, the running

time of this program was even longer than the time of sequential program.

(f) Which timing methods seem best for parallel programs? How does this compare with your expectations?

From my perspective, when you have a large amount of computations, the real time method seems best for parallel programs and in other cases, the clock timing method seems best for parallel programs. It matches my expectations.

3. Assumption

From the output data, we could see that when the number of threads is 4, the running time of the parallel program of any thread version achieves the best. According to my experience on stdlinux and the resources of my own computer, I suppose that the best performance would be achieved when the number of threads used is equal to the number of CPU cores.