Multiprocessor Scheduling

Annalise Tarhan

February 3, 2021

1 Run the first job on one simulated CPU as follows: ./multi.py -n 1 -L a:30:200. How long will it take to complete?

Job a's run time is 30 time units, or three time slices.

2 Now increase the cache size so as to make the job's working set fit into the cache. Can you predict how fast the job will run once it fits in cache?

This depends on the warm time, which is given as 10 time units, and the warm rate, whose default can be calculated to be 2. If it is set to 2, then after the warm time it will run twice as fast. Since the run time is 30, it will actually run in 20 time units. If the warm rate s set to 4, the last 20 time units of the run time will run in 5 time units, for a total run time of 15.

3 Use the time_left trace to see how much run-time the job has left after each tick. What do you notice about how the second column decreases?

With the default warm rate of two, the run-time remaining decreases by one each tick before the cache is warm and by two each afterwards.

4 At what point does the cache become warm in the previous example? What happens as you change the warmup_time parameter?

The default warmup time is 10, so the cache becomes warm at the 10th time unit. If the warmup_time parameter changes to n, the cache becomes warm at the nth time unit.

5 Run the following three jobs on a two-CPU system: -n 2 -L a:100:100,b:100:50,c:100:50. Can you predict how long this will take, given a round-robin centralized scheduler?

In this example, jobs are assigned to CPUs' time slices, not the CPUs themselves. There are two CPUs and three jobs, so each job will split time evenly between CPUs (since 2 and 3 have no common factors larger than 1.) It takes a full time slice for a cache to warm, and a job never runs on th e same CPU twice in a row. This would be okay for jobs b and c, since their working set sizes of 50 units each are small enough to share a CPU's 100 unit cache. Job a, though, clears out b and c's sets with its 100 units and never sticks around to enjoy it. Since a always comes along before b or c reuses a cache, there is effectively no caching at all, and it takes the full (100+100+100)/2 = 150 time units to complete all three jobs.

6 Use the cache affinity flag to limit which CPUs the scheduler can place a particular job upon. Place jobs b and c on CPU 1, while restricting a to CPU 0. Can you predict how fast this version will run? Why does it do better? Will other combinations of a, b, and c on the two processors run faster or slower?

This version appropriately utilizes cache resources, allowing them to stay warm for the most time possible. Job a will warm its cache after 10 time units and will run twice as fast, finishing at time 55. Jobs b and c each run with a cold cache for 10 time units each, then share the warm cache for another 90 time units, finishing at time 110. Other combinations will suffer from a's cache hogging, although allowing either b or c to run on CPU 0 after a finishes would be even more efficient.

7 Super-linear speedup occurs when you run a set of jobs on N CPUs and achieve a speedup of more than a factor of N. Run -L a:100:100,b:100:100,c:100:100 with a small cache to create three jobs. Run this on systems with 1, 2, and 3 CPUs. Now do the same, but with a larger per-CPU cache of size 100. What do you notice about performance as the number of CPUs scales?

M = 50 300, 150, 100
M = 100 300, 150, 55

For one and two CPU systems, the difference in cache size makes no difference. For a 3 CPU system, however, the larger cache size cuts the run-time almost in half relative to a 3 CPU system with a smaller cache. This occurs because each job gets to run with a fully warmed cache for the longest time possible.

8 Run with two CPUs again, and the three job configuration -L a:100:100,b:100:50,c:100:50. How does adding the per-CPU scheduling option do, as opposed to the version with the hand-controlled affinity limits? How does performance change as you alter the peek interval to lower or higher values? How does this per-CPU approach work as the number of CPUs scales?

This version allows better CPU use, reducing the total run-time to 100 time units. Changing the peek time to anything below seven reduces the total run-time to 90 time units. Increasing it above 10 causes a much longer run-time, 130 time units or more. Adding a third CPU allows the very fastest result: 55 time units.