MYE023: Homework #2

Due on Monday, May 1, 2017

Vassilios V. Dimakopoulos

George Z. Zachos

April 30, 2017

Contents

1 Exercise #1

1.1 About

This exercise is about the multiplication of integer NxN arrays using the `OpenMP` specification. The serial calculation consists of three (3) nested for-loops and the purpose of this exercise is to parallelize all three, one at a time. The three resulting programs will be executed using both `static` and `dynamic` scheduling policies.

1.2 Experiment details

The calculation consists of N^3 loop iterations (N=1024), while the number of threads used in `parallel` regions is four (4) and chunk size is automatically set to default values.

1.2.1 System Specifications

The experiments were conducted on a Dell OptiPlex 7020:

- CPU: Intel® Core^{TM} i5-4590 CPU @ 3.30GHz (64 bit)
- RAM: 2 DIMMs x4GiB @ 1600MHz DDR3
- Cache line size: 64B (in all levels)
- Cache associativity:
  - L1, L2: 8-way set associative
  - L3: 12-way set associative

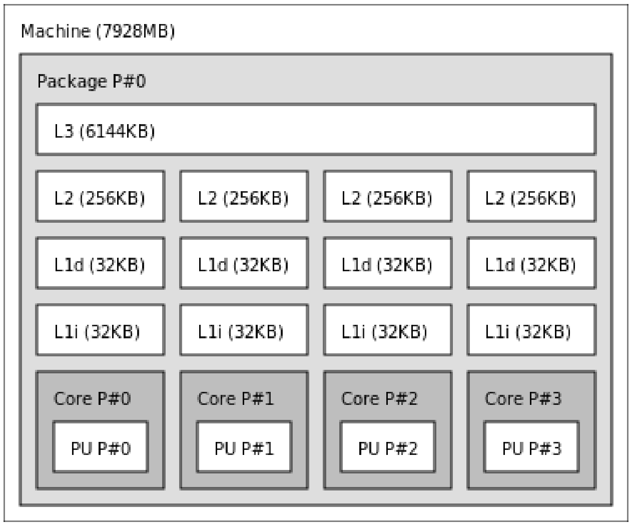

Figure 1: Topology information of a Dell OptiPlex 7020

1.3 Timing Results

In the following table and plot the recorded execution times are displayed. Note that X axis is plotted on a linear scale while Y axis on a (base 10) logarithmic scale.

Timing results of matrix multiplication (Time unit: seconds) Array size: 1024x1024, Number of threads: 4			
	Parallelized loop nesting level		
Scheduling Policy	0	1	2
Static	0.9228565	1.0030725	1.988372
Dynamic	0.87141825	1.049885	28.30894775

Table 1: Timing results of 2D matrix multiplication

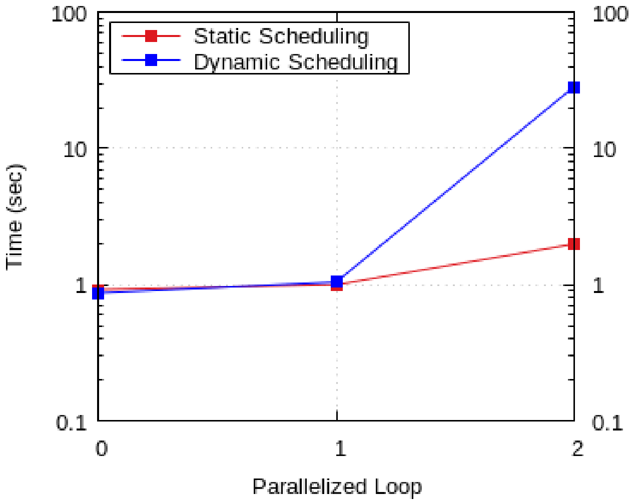

Figure 2: Timing results of 2D matrix multiplication

1.4 Conclusion

Based on the results presented above and given that the average execution time of the serial program is 3,82147025 seconds, we conclude that:

- The best program performance[1] is achieved by parallelizing the outermost for-loop as only one parallel region is invoked. Parallelizing the middle and the innermost loop will cause N and N^2 invocations respectively and the granularity of the tasks being dispatched to the team threads to decrease. Due to these continuous invocations, execution time is increased as overheads are introduced by thread management (creation, synchronization[2], destruction etc.).
- Both `dynamic` and `static` schedules result in approximately the same execution time, except for the case of parallelizing the innermost for-loop. During `static` schedule, the iteration space is divided into chunks that are approximately equal in size, and at most one chunk is distributed to each thread. In constrast, during `dynamic` schedule, default chunk size equals to one iteration and in total N^3 dispatches take place[3]. For this reason, `dynamic` schedule exponentially increases program time.

[1] About 86% speedup.

[2] There is an implied barrier at the end of every `parallel` region.

[3] N dispatches every time the `parallel` construct is encountered