CS 6290: High-Performance Computer Architecture

Fall 2015

Project 3
Due: December 4th 2015 at 5pm EST

This project is intended to help you understand multi-core performance. As with previous projects, you will need VirtualBox and our project virtual machine for this project.

You will fill in the form fields in this PDF document and submit it in T-Square (the submitted file name should still be PRJ3.pdf). Note that you will need to **save the filled-out form**, and that many PDF readers/viewers cannot save the document. One free online viewer that you can use to do this is http://www.pdfescape.com/. If you are having trouble filling (or saving) this PDF form, fill out and submit the provided PRJ3.txt form. If you do use the PRJ3.txt file, please note that the answers you put into PRJ3.txt should be **exactly** the same as what you would enter in the PDF form. In both PDF and TXT forms, you must first provide your answer to the question (e.g. a number), and then you can enter any explanations that the form is not asking for, but such not-asked-for parts of the answer should be enclosed in square brackets. E.g. answer 9.7102 may be entered as

9.7102 [Because 9.71+0.0002 is 9.7102]

Finally, if you use the PRJ3.txt file, **do not** modify any line that begins with a square-bracket. These lines are needed to quickly identify which question you are answering, and any ambiguity about which question you are answering will result in losing all points for that question or questions.

Additional files to upload are specified in each part of this document. Do **not** archive (zip, rar, or anything else) the files when you submit them – each file should be uploaded separately, with the file name specified in this assignment. You will lose up to 10 points for not following the file submission and naming guidelines (zipped files, files not named according to instructions, etc.). Furthermore, if it is not VERY clear which submitted file matches which requested file, we will treat the submission as missing that file. The same is true if you submit multiple files that appear to match the same requested file (e.g. several files with the same name). In short, if there is any ambiguity about which submitted file(s) should be used for grading, the grading will be done as if those ambiguous files were not submitted at all.

Most numerical answers should have **at least two decimals** of precision. Speedups should be computed to **at least 4 decimals** of precision, using the simulated time from the report.pl output. Do **not** use the IPC or the number of cycles from report.pl for speedup calculation – in multi-core simulations these do not account for all the time the processor is waiting on synchronization operations, so they are no longer proportional to

execution time. You lose points if you round to fewer decimals, or if truncate digits instead of rounding to the closer value.

This project can be done either individually or in groups of two students. If doing this project as a two-student group, you can do the simulations and programming work together, but each student is responsible for his/her own project report, and each student will be graded based solely on what that student submits. Finally, **no collaboration with other students or anyone else is allowed**. If you do have a partner you **have to** provide his/her name here _____ and his/her T-Square username here _____. Enter None in both of these if you have no partner.

Part 1 [40 points]: Running a parallel application

In this part of Project 3 we will be using the LU benchmark. We will also be using a processor with more (sixteen) cores (cmp16-noc.conf). So, for example, to simulate 4-threaded execution you would use a command like this (note the absence of spaces between -n and 128, and between -p and 4):

~/sesc/sesc.opt -c ~/sesc/confs/cmp16-noc.conf -olu.out -elu.err lu.mipseb –n128 -p4

To complete this part of the project, run the lu application with 1, 2, 4, 8, and 16 threads. Then fill in the blanks, taking into account all the runs you were asked to do:

A) Rename the simulation reports to sesc_lu.mipseb.Part1-p1, sesc_lu.mipseb.Part1-p2, sesc_lu.mipseb.Part1-p4, etc. and submit them in T-Square.

B) Fill out the execution time, parallel speedup, and parallel efficiency with 2, 4, etc. threads. Individual students can enter zeroes in the last two rows of the table, but students working in pairs need to enter these numbers (and submit the two additional simulation reports). Enter Sim Time with precision of at least three decimals, and speedup and efficiency with precision of at least two decimals.

	SimTime (in ms)	Parallel Speedup	Parallel Efficiency
-p1	_____ ms		
-p2	_____ ms		
-p4	_____ ms		
-p8	_____ ms		
-p16	_____ ms		

Note: Parallel speedup is the speedup of parallel execution over the single-thread execution with the same input size. Parallel efficiency is the parallel speedup divided by the number of threads used – ideally, the speedup would be equal to the number of threads used, so the efficiency would be 1. When computing the

speedup and efficiency we cannot use IPC or Cycles that are reported for each processor, because these do not account for the cycles where that core was idle (e.g. because the thread was waiting for something to happen). So we need to use the "Sim Time" we get from report.pl because it accounts for all cycles that elapse between the start and completion of the entire benchmark.

C) Our results indicate that parallel efficiency gets lower as we use more cores. Why do you think this is happening?

D) When we use two threads (-p2) instead of one (-p1), the IPC achieved by Core 0 (the first processor listed) got slightly lower because

E) Now look at the simulation reports for these simulations. Core 0 executes more than its fair share of all instructions because

Part 2 [20 points]: Cache miss behavior

In this part of Project 3, we will be focusing on the number of read misses in the DL1 (Data L1) cache of Core 0, using the same simulations that we already did for Part 1. In the report file generated by the simulator (sesc_lu.mipseb.something, not what you get from report.pl), the number of cache read misses that occur in each DL1 cache (one per processor core) is reported in lines that begin with "P(0)_DL1:readMiss=".

F) The total number of read misses that occur in the DL1 cache of Core 0 is

Simulation	-p1	-p2	-p4	-p8	-p16
Core 0's DL1 read misses					

Your answers here should be integer numbers.

G) The number of these misses changes this way as we go from one to two to four, etc. threads because

Part 3 [40 points]: Identifying accesses to shared data

You task in this part of the project is to determine how many read misses in each core's DL1 cache are compulsory (compMiss), replacement (capacity or conflict, the counter should be called replMiss), and coherence misses (coheMiss). Note that this classification is similar to the one you did for Project 2, except that you now need to identify coherence misses. To simplify classification, we will **not** follow the exact definition of coherence misses ("those misses that would have been hits were it not for coherence actions from other cores"). Instead, we will use a definition that allows much simpler implementation: a coherence miss is a miss that finds in the cache a line whose tag matches the block it wants, but that block has a coherence state that prevents such access. In the case of read misses, this means that the line was found in an "Invalid" coherence state. Note that this identification of coherence misses may not be trivial in the SESC simulator because of the way it handles tags during invalidation. If a miss is not a coherence miss, then you can classify it as either compulsory or replacement miss by checking if the block was ever in that cache. When checking whether the miss is a compulsory miss, be careful to track the "was previously in this cache" set of blocks for each cache separately.

H) Create a Changed.zip file with any simulator source code files that you have modified in Part 3 of the project, and submit this Changed.zip file together with your project.

I) With your changed simulator (that now counts compulsory, replacement, and coherence read misses), re-run the simulations from Part 1 and submit the resulting simulation report files as sesc_lu.mipseb.Part3-p1, sesc_lu.mipseb.Part3-p2, sesc_lu.mipseb.Part3-p4.

J) The number of all read misses, compulsory read misses, replacement read misses, and coherence read misses for the DL1 cache of Core 0 is:

	Core 0's DL1 readMiss	**Core 0's DL1 compMiss**	**Core 0's DL1 replMiss**	**Core 0's DL1 coheMiss**
-p1				
-p2				
-p4				
-p8				
-p16				

Note: readMiss numbers here should be the same as those you had in Part 2.