Problem 4 part 1

Even Flørenæs

Fall 2016

TDT4200 Parallel Computing
Department of Computer Science

Contents

1 Theory

1.1 Types of cache misses

1.1.1 Cache conflict

Miss occurring because several memory blocks are mapped to the same area of the cache, and some data will be discarded to solve conflict.

To avoid cache conflict introducing associativity will reorder how the blocks are arranged and may decrease number of conflicts.

1.1.2 Cache capacity

Miss occurring because the program needs to reference a collection of memory blocks of a size which exceeds the capacity of the cache memory, and some of the data will be discarded from the cache.

To avoid misses due to capacity increase the capacity to match the required space for running necessary programs.

1.1.3 Cache compulsory

Miss occurring when the program references the first block of memory from an empty cache memory, for example after a start up. The program has to access other, slower, memory areas like the RAM to find the referenced data.

To avoid misses due to cache compulsory increase the block size in the cache.

1.2 Branching

Branches in computers will experience to come to a crossroad where it can continue in minimum two different directions. If the branch chooses one of the directions, and later it proves to be the wrong direction. The branch will have to start over again in the correct direction. Unfortunately the branches often don't know which direction to choose when it approaches the crossroad. One way to solve this issue is to build branch predictors which predicts the direction of the branch. If the predictor is right the system will experience improved performance, and no stop and start of branches in the middle of an instruction. If the predictor is wrong the system performance will decrease as the branch will need to restart. If one develops well functioning predictors the system will achieve much higher performance.

1.3 SIMD instructions

SIMD is short for single instructions multiple data which is one implementation of a parallel instruction set. SIMD can vectorize scalar code by performing a single instruction for multiple data streams in parallel. SIMD can not be used to perform different instructions in parallel as Single Instruction in SIMD suggest. SIMD can be used to speed up for loops performing the same operation every loop but for different data.

1.4 OpenMP schedules

In static scheduling the iterations of a for-loop are evenly divided between the available threads. In the case of static scheduling we need to make sure that the work load is evenly balanced between the different threads.

In dynamic scheduling the work load is evenly distributed among threads. Because when one thread is finished performing its task, it will not go idle but find work somewhere else. Which means every available thread will stay busy until all work is done. The distribution of work load is performed at run time with a queue system leading to large overhead at runtime.

Guided scheduling is somewhat a combination of static and dynamic scheduling. Where the threads dynamically grab blocks of iterations, where the blocks are decreasing in size as the calculation proceeds. With this schedule it would be beneficial that the calculation time increases as the iteration increases. Because the sizes of the blocks decrease one can increase the work in these blocks.