Lab 1

Oscar Galindo

Advanced Computer Architecture

Dr. Shirley Moore

**2.1**

1. What should be the minimum size of the cache to take advantage of blocked execution?

The minimum size requires the use of at least the entire cache line, in such line there are 64 bytes, which means there is enough space for 8 float numbers. If the intention is to use blocks, then we should have 64\*64 blocks of space or 4096 bytes which is 212 bytes per array, given that we have two arrays it would be ideal to have at least the two block-copies of the input and output arrays, so the minimum size should be equal to 2\*212 or 213 bytes which is around 8KB.

1. How do the relative number of misses in the blocked and unblocked versions compare in the preceding minimum-sized cache?

In the blocked version the number of misses per block is exactly equal to 2\*B since we have B lines to bring from main memory into cache per array involved in the transpose algorithm. Furthermore, we have exactly n2/B2 blocks. So, the asymptotic “complexity” of the number of misses is described by (n2/B2) \* (2\*B) or 2\* n2/B. In comparison, the unblocked version creates more misses due to the manner of accessing the src array in a column-major order. Accessing an array in column-major order while in memory arrays are saved in row-major order means that no locality can be “exploited” to perform the transpose operation faster because every access to read or write to the array will incur in a cache miss, meaning a line of array elements must be brought from main memory with a high cost. In addition, due to size issues, bringing elements from different rows at every single iteration will saturate the cache of new lines to store, meaning at some point if we are reading the line of cache that includes element [n][1] most likely the lines that include the range of values [1…n/2][1] would have been evicted, meaning more requests are made to main memory. This last means, in the worst case every single access to obtain values from src would inquire in a cache miss, hence the number of misses only for the src array are asymptotically close to n2 -which already is at the same complexity of the blocked algorithm in terms of misses. Finally, to access the elements of the row-major stored-as array we would inquire in n2/B misses, or a combined number of misses described as n2+n2/B, where n2+n2/B > 2\* n2/B.

1. Write code to perform a transpose with a block size parameter B that uses B\*B blocks.

Submitted alongside this assignment

A screenshot of a computer

Description automatically generated with medium confidence

1. What is the minimum associativity required of the L1 cache for consistent performance independent of both arrays’ position in memory?

According to the table of cache associativity and cache size located in Appendix B.8 of the book the minimum associativity needed is 2 because 1 has a higher rate of miss and a higher penalty while 4 and 8 have the same numbers than an associativity of 2.

1. Try out blocked and nonblocked 256\*256 matrix transpositions on a computer. How closely do the results match your expectations based on what you know about the computer’s memory system? Explain any discrepancies if possible.

The results do not match my expectations:

Text

Description automatically generated

This I believe is since most likely the 256\*256 fits in cache, so if that is true the blocking won’t help instead more evaluations occur because of this set up and that hurts the execution time.

**2.2.** For best performance given a non-unit stride prefetcher, in the steady state of the inner loop, how many prefetches must be outstanding at a given time?

Two prefetch instructions must be missing, one to ask for the input entry that will be used in so many iterations and another that asks for the output sections that will be used in so many iterations

**2.3**

* 1. Create a blocked version of the matrix transpose with software prefetching.

Attempting Column pre-fetch:

**Text

Description automatically generated**

Attempting only row pre-fetching (one-by-one):

Text

Description automatically generated

* 1. Estimate and compare the performance of the blocked and unblocked transpose codes both with and without software prefetching

Note: This analysis is somewhat qualitative as the numbers output by PAPI are a little confusing.

In terms of direct comparison blocked vs. unblocked should offer a simple conclusion, the fact that not as many misses are incur at because of the block locality should give block implementation a clear advantage. In terms of performance the performance should be noticeably different between the implementation as the size of the array (n) increase if B is fixed. Here are some actual execution times:

Text

Description automatically generated

In terms of performing pre-fetching between unblocked and blocked implementations the blocked implementation gains less from prefetching if the block is small because a miss could occur while prefetching is being performed. In addition, in the blocked implementation pre-fetching only makes sense when a blocksize is big because otherwise the performance gains are minimal. So, in terms of the unblocked implementation pre-fetching should allow to ask “with time” for data that, predictably, would be accessed. In terms of the implementations, according to the resulting times, pre-fetching hurts performance probably because latency due to too many cache pre-fetches occurring, and because the size of n being reasonably small.

Furthermore, I have concluded if one is going to choose an implementation among the four possible one should try to implement the blocked code without pre-fetching.