HPC Spring 2017 – Homework 1

Robert van Engelen

Due date: January 31, 2017

1. Consider the characteristics of these two architectures:

Architecture	L1 cache		L2 cache		memory
	size	access cycles	size	access cycles	access cycles
1	32KB	2	2MB	8	100
2	64KB	3	1MB	6	90

   where access time is measured in cycles and both architectures have the same cycle time and same instruction execution speeds. Suppose that an application has a data set of 1MB and the load/store access window ranges from 16KB to 1MB. After profiling the application's memory access, it is determined that for architecture 1, 75% of these accesses are in L1 cache, 23% are in L2 cache, and 2% are in memory. For architecture 2, 93% of these accesses are in L1 cache, 2% are in L2 cache, and 5% are in memory.

   (a) Given these statistics, for each architecture compute the average number of cycles to access the application data. Based on this comparison, which architecture is faster for this application?

   (b) Suppose we rewrite our application such that we optimized the data access for the 1MB/2MB L2 caches. We further assume that all data is preloaded in L2 cache and no memory access overhead will incur. However, with this algorithmic change we observe 95% L1 (capacity and conflict) misses! For each architecture compute the average number of cycles to access the application data. Based on this comparison, which architecture is faster for this "optimized" application?

2. Consider the following code with four assignment instructions (see also slides 19, 23 of Architecture part I):

```
x = a+1
y = 2*x
z = a+b
x = z+1
```

   (a) Indicate the flow dependences (RAW) and anti dependences (WAR) between the statements.

(b) Assume that each operation has a throughput of 1 cycle (i.e. a new operation can start the next cycle) and operations have a latency of 3 cycles. When you take the dependences into account and assume that instructions are executed strictly in order and execution is delayed when operand values are not available, how many cycles does it take to execute this code? Draw the execution schedule (similar as shown in slide 19).

(c) Now assume that a dynamic scheduler is used to optimize instruction scheduling. The scheduler fetches four instructions simultaneously and reorders their execution to minimize the total latency. Renaming is used to break WAR dependences. But only one instruction can start execution per cycle (slide 23 assumes two instructions can start if the functional units used permit this). What is the minimum number of cycles required to execute the code?

3. Which compiler optimizations reduce memory latencies/memory bandwidth? List the loop and non-loop optimizations.

4. Consider the following program fragment

```
DO I = 1, N
   B(I) = T(I) * X(I)
ENDDO
DO I = 2, N
   A(I) = B(I) - B(I-1)
ENDDO
```

(a) Apply loop peeling to the first loop and fuse the loops. Explain why the fused loops yield a program that is computationally equivalent to the former, despite the flow dependences for B. In other words, are the values of B set correctly?

(b) What is the potential benefit of fusing these loops?

(c) Suppose that array B is no longer used after these loops. Can you eliminate array B from the fused loops and replace the B(I) and B(I-1) accesses with something else without having to recompute T(I)*X(I)? That is, ensure that the number of multiplications is 1 per loop iteration.

(d) Suppose the body of the second loop is changed to A(I) = B(I+1) - B(I-1), such as with a different finite differencing stencil computation. Now there is a dependence that prevents loop fusion, since after fusion B(I+1) is not computed until one iteration later. Can you think of a way to set the B values in the first loop such that it will be possible to fuse the loops? Hint: it is generally valid to add a constant to the loop counter index in the loop as long as you also adjust the loop start and end values by that constant.

5. Consider the following program fragment

```
DO I = 1, N
   T(I) = sqrt(B(I-1))
   A(I) = X(I) - B(I-1)
   B(I) = T(I) * X(I)
ENDDO
```

   (a) Show the dependence graph (see also slides 31 and 32 of Compiler Optimizations). Note that edges in the graph are labeled < and =, where = means a flow dependence in the same iteration between two statements and < means a cross-iteration flow or anti dependence.

   (b) Determine the strongly-connected components.

   (c) From this dependence analysis apply loop fission.

6. Find out which compiler optimizations are applied by GCC 4.x from the tables on the presentation notes "Compilers" pages 3 and 20. That is, which of the optimizations in these tables does GCC support and with what compiler option? You can check the GCC manual pages for available compiler options, such as `-funroll-loops`. List these options with the entries of the two tables or list "N/A" if the optimization has no specific option. Do not list `-O3` as an option, only list the GCC `-f` options.