CSC458 HW1

Chi-Chun, Chen

Ex 1

1. csug.cycle2
2. Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz
3. avx2
4. 256 bits

Ex 2

To achieve scalability, I would map each three consecutive tasks of processing one image to one core, in addition, each core should process the same number of images. By mapping this way, if there are n cores in the system, each core can process three images at once (pipeline of three stages) and can process $3 * n$ images at the same time.

Ex 3

Processor doing the prediction by assuming that the branch is not taken, and it could execute the instructions before the MEM phase of the branch instruction is finished which is how speculative processor improve its performance. However, if the speculative processor can only prove whether the branch should be taken after the MEM phase of the branch instruction is finishing executed. Therefore, if mis-speculation happens, a deeper pipeline has more stages to execute before MEM phase which is inevitable to consume more time to finish than a pipeline with fewer stages.

Ex 4

1. Four distinct loop dependences in the loop. Three true-dependences, one loop-independent dependency.
2. Seven distinct distances: 6, 4, 2, 0, -2, -4, -6.
3. Three direction vectors: $<$, $=$, $>$.

Ex 5

1. Since the swap statements has dependences to the inner for-loop, such as the true dependence of $a[j] = a[j] + 1$ and $a[i] = a[i + 1]$ where i equals to j. According to Definition 2.8 in AK, moving swap statements before the inner for-loop does not preserve the dependency direction so it is prohibited by the dependence-based framework.

2. Yes, according to Definition 2.5 in AK, two program is equivalent if they produce the same value. Moving the swap statements produce the same result as not moving the swap statements, which is adding every elements in array a by nine. Therefore, I suppose the program is equivalent whether moving the swap statement.

Ex 6

Assumptions

1. The compiler we use in this example is able to figure out data dependences between each instructions.

2. The hardware is able to track the data dependences of load and store at the runtime.[1]

3. Load and Store do not cause interruptions which takes unpredictable amount of cycles.

Scheme for Data Speculation

To take advantage of the data dependences that our compiler give us from assumption number one, we move all the load instructions that have no true dependences to the start of the program. If the load instruction have true dependences, then we should only execute it after the store instruction has finished executing. However, some load and store instructions might depend on runtime execution. In this sort of context, we have to use the assumption number two.

Assumption one and two avoid the opportunity of mis-speculation, since we have all the data dependences information for all the instructions and there is no any ambiguity for doing speculation[2]. If we have all the data dependences then we can move the instruction with correct order without stalling for load and store. In this scheme, the processor buffers the speculative results until we know that the speculation is correct. After proving the speculation is correct and have the buffer be written to memory does the instruction is completed.

References

[1] Advanced Load Address Table (ALAT)
`https://en.wikipedia.org/wiki/Advanced_load_address_table`

[2] Andreas Moshovos, Scott E. Breach, T. N. Vijaykumar, Gurindar S. Sohi: Dynamic Speculation and Synchronization of Data Dependences,
http://ftp.cs.wisc.edu/sohi/papers/1997/isca.data-dep-spec.pdf