Computer Architecture Area

Fall 2011 PhD Qualifier Exam

November 1**8**th 2011

This exam has six (6) equally-weighted problems. **You must submit your answers to all six problems.** Write your answers/solutions clearly and legibly. Illegible answers will be treated as wrong answers. You should clearly mark on top of each page of your answers which problem the page corresponds to. For each problem, start your answer/solution on a new page, and use as many pages per problem as you need. Although there is no restriction to how long your answers should be, try to keep your answers and solutions short and to the point.

Good luck!

1. [Cache coherence] A particular multiprocessor has 8 processor cores nodes, C0-C7, connected by a shared bus. Coherence is maintained using the MESI snooping coherence protocol.

1.1 Consider the following sequence of accesses to a particular block B, which has not been accessed by any of the cores. First C0 writes to B, then C1 reads from B, then C2 reads from B. For each cache in C0, C1, C2, and C3 show the status of this block as it evolves over time. For each of these caches, you must specify after each of these accesses whether block B is present in that cache and, if present, what its state is.

1.2 Now assume that this system uses a switched interconnect and a directory-based coherence mechanism, and if core C7 is the home for block B. For each of the three accesses to block B, specify which messages are sent. For each message, you must specify the type of the message, its origin and destination, and whether it carries data or not.

1.3 In this 8-core system, describe at least one scenario in which the bus-based snooping coherence approach would significantly outperform the switched interconnect with directory-based coherence, and at least one scenario in which the bus-based snooping coherence would significantly underperform the switched interconnect with directory-based coherence.

1.4 To improve performance, a processor can use a write buffer and retire a store instruction when it deposits its value into the write buffer. The values in the write buffer can them be sent to the cache in the order in which they were written.

1.5 How would store-to-load dependencies be affected by this write-buffer approach. What needs to be done to achieve correct behavior of loads that read values written by recent stores.

1.6 In a multi-processor system, the write buffer approach can create consistency problems, even when each core gets correct handling of its store-to-load dependencies. Describe at least one scenario in which write buffering can result in a violation of sequential consistency.

2 Are there any consistency models that would not be affected by the write buffer? Which consistency model would you support for this system?

2. [Branch prediction] Branch predictors are used for predicting the outcome of the branches to improve the accuracy of fetching the right instructions to avoid the pipeline stalls. Traditionally branch predictors are history based predictors which record the history of the branch execution and classify it; some branch predictors learn the stochastic behavior of the branch and train the models using gathered data (such as outcomes of other branches) to arrive at the predictions. Please answer the following:

2.1 Compare three different predictors with regard to: (a) Type of data gathered and used in prediction, (2) The real estate or the chip area used for implementing the predictor (size of tables, logic, etc.), (3) Accuracy and (4) Energy efficiency based on activity performed

2.2 Discuss the root cause of poor(er) performance for one of the three predictors and suggest an improvement to increase its accuracy without significantly increasing its space needs or activity count.

2.3 It is proposed to devise a truly "predictive" branch predictor which is not based on history but based on evaluating the branch condition well ahead of time using the variable values participating in the predicate. Show how, with the right compiler support one could hoist the evaluation of branch predicates well ahead of where they occur. Show code excerpt and show the potential placement points for evaluation using a control flow graph. Doing such an approach purely in software would not be feasible; thus a hardware support mechanism is envisioned. Develop hardware support for such ahead of time outcome prediction scheme: discuss various overheads space, activity and latency overheads. Comment upon additional potential benefits that might be possible due to this approach over the best (pure) history based predictor.

3 [Multi-core performance] To take advantage of the computing power in multi-core processors, programmers often need to judge whether they want to parallelize their applications. Before the parallelization, programmers should estimate the performance benefits. Let's say a program is composed with 3 functions (FuncA, FuncB, and FuncC). The execution time in sequential code is as follows: FuncA:65%, FuncB:20%, and FuncC:15%. The total execution time of the sequential code is 1sec in this example. Assume that only FuncA and FuncB can be parallelized. We have the following three machines.

Machine A: 2GHz 32 CPU cores

Machine B: 4GHz 4 CPU cores + 500Mhz 512 GPU cores

Machine C: 2GHz 16 CPU cores + 1GHZ 128 GPU cores

3.1 Among the three machines, which one do you choose to run the program to get the best performance?

3.2 If there is an overhead of sending a job to GPU (100msec, this overhead includes sending data to GPU, invoking the kerenel, sending the data back to CPU) , what will be the results?

3.3 Let's say CPU cores also have a SIMD unit. Let's say 50% of Func A can be simizized. The width of SIMD is 16. In that case, what will be the answer? (There is no GPU-CPU communication overhead)

3.4 After you parallelize the code, you found out that the performance is not showing benefits as you expected. What kind of tests would you perform to find out the cause of performance degradations?

4 [GPUs] GPUs become very popular these days. We like to enhance the GPU architecture to be like a multiscalar processor. What hardware structures should we add? What will be the performance bottlenecks in the machine?

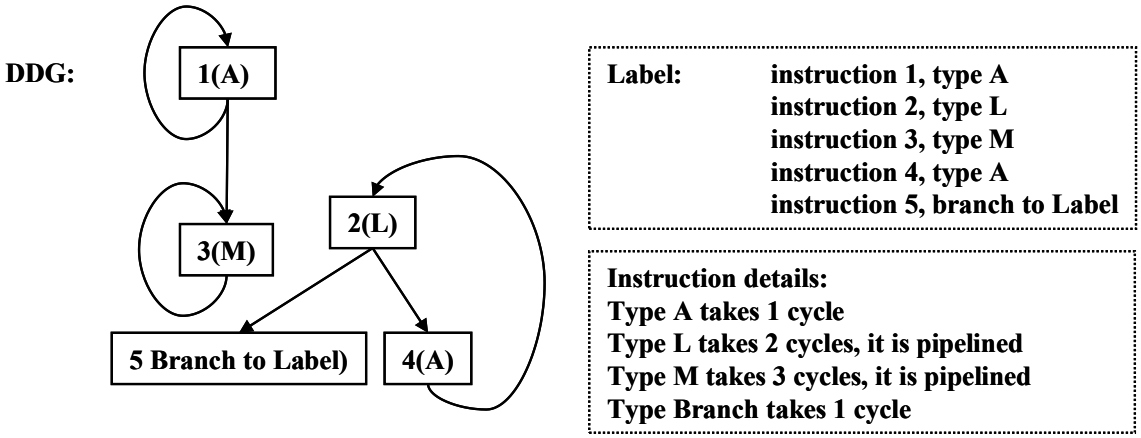
5 [Architectural support for synchronization] Synchronization plays an important role in computer architecture performance on parallel workloads. For each of the following primitives, give a potential *hardware* implementation for a cache-coherent, shared-memory multi-core system with a wormhole-routed, torus network-on-chip. Explain how your hardware approach is (a) high-performance, (b) scalable to many, many cores, and (c) implementable and hardware efficient. You will be graded dependent on the level of detail you provide in your answer.

5.1 Lock and Unlock semaphore: lock(x) requests lock "x" and does not return until permission is granted. unlock(x) frees (unlocks) lock "x" and thereby allows another waiting thread to lock "x"

5.2 Fetch and Add: FandA(I) is guaranteed to return a unique value of the global, integer variable "I." Thus if two threads perform FandA(I), thread 1 would receive "0" and thread 2 would receive "1," or vice-versa. But neither should receive the same value.

5.3 Barrier Synchronization: Barrier(Q, N) does not return until (N-1) other threads have reached point "Q".

6 [Software pipelining] Consider the following loop (note: there are no anti- nor output dependencies):



Assume there is one unit of each type (A, L, M, Branch) in the multiop (instruction 5 is of type Branch) and loop iterates for more than three times.

6.1 Find the resource-constrained minimum initiation interval (**ResMII)** and the recurrence-constrained minimum initiation interval (**RecMII)** for the loop.

6.2 Show the final software pipelined code. It is sufficient to write the number 1-5 to indicate the instruction. Indicate where each of the "kernel," "prolog," and "epilog" begins and ends. Instruction 6 must go at the end of the kernel

Note, if you do not understand modulo scheduling, you may wish to read

B. Ramakrishna Rau: Iterative modulo scheduling: an algorithm for software pipelining loops. MICRO 1994:63-74