Norwegian University of Science and Technology (NTNU)
DEPT. OF COMPUTER AND INFORMATION SCIENCE (IDI)

Course responsible: Professor Lasse Natvig
Quality assurance of the exam: PhD Jon Olav Hauglid
**Contact person during exam**: Magnus Jahre

Deadline for examination results: 23rd of June 2009.

**EXAM IN COURSE TDT4260 COMPUTER ARCHITECTURE**
Tuesday 2nd of June 2009
Time: 0900 - 1300

**Supporting materials**: No written and handwritten examination support materials are permitted. A specified, simple calculator is permitted.

By answering in short sentences it is easier to cover all exercises within the duration of the exam. The numbers in parenthesis indicate the maximum score for each exercise. We recommend that you start by reading through all the sub questions before answering each exercise.

*The exam counts for 80% of the total evaluation in the course. Maximum score is therefore 80 points*.

## Exercise 1) Instruction level parallelism (Max 10 points)

a) (Max 5 points) What is the difference between (true) data dependencies and name dependencies? Which of the two presents the most serious problem? Explain why such dependencies will not always result in a data hazard.

Solution sketch:
**True data dependency**: One instruction reads what an earlier has written (data flows) (RAW).
**Name dependency**: Two instructions use the same register or memory location, but there is no flow of data between them. One instruction writes what an earlier has read (WAR) or written (WAW). (no data flow).
True data dependency is the most serious problem, as name dependencies can be prevented by register renaming. Also, many pipelines are designed so that name-dependencies will not cause a hazard.
A dependency between two instructions will only result in a data hazard if the instructions are close enough together and the processor executes them out of order.

b) (Max 5 points) Explain why loop unrolling can improve performance. Are there any potential downsides to using loop unrolling?

Solution sketch:
Loop unrolling can improve performance by reducing the loop overhead (e.g. loop overhead instructions executed every 4th element, rather than for each). It also makes it possible for scheduling techniques to further improve instruction order as instructions for different elements

(iterations) now can be interchanged. Downsides include increased code size which may lead to more cache misses and increased number of registers used.

## Exercise 2) Multithreading (Max 15 points)

a) (Max 5 points) What are the differences between fine-grained and coarse-grained multithreading?

Solution sketch:
Fine-grained: Switch between threads after each instruction. Coarse-grained: Switch on costly stalls (cache miss).

b) (Max 5 points) Can techniques for instruction level parallelism (ILP) and thread level parallelism (TLP) be used simultaneously? Why/why not?

Solution sketch:
ILP and TLP can be used simultaneously. TLP looks at parallelism between different threads, while ILP looks at parallelism inside a single instruction stream/thread.

c) (Max 5 points) Assume that you are asked to redesign a processor from single threaded to simultaneous multithreading (SMT). How would that change the requirements for the caches? (I.e., what would you look at to ensure that the caches would not degrade performance when moving to SMT)

Solution sketch:
Several threads executing at once will lead to increased cache traffic and more cache conflicts. Techniques that could help: Increased cache size, more cache ports/banks, higher associativity, non-blocking caches.

## Exercise 3) Multiprocessors (Max 15 points)

a) (Max 5 points) Give a short example illustrating the cache coherence problem for multiprocessors.

Solution sketch:
See Figure 4.3 on page 206 of the text book. (A reads X, B reads X, A stores X, B now has inconsistent value for X).

b) (Max 5 points) Why does bus snooping scale badly with number of processors? Discuss how cache block size could influence the choice between write invalidate and write update.

Solution sketch:
Bus snooping relies on a common bus where information is broadcasted. As number of devices increase, this common medium becomes a bottleneck.
Invalidates are done at cache block level, while updates are done on individual words. False sharing coherence misses only appear when using write invalidate with block sizes larger than

one word. So as cache block size increases, the number of false sharing coherence misses will increase, thereby making write update increasingly more appealing.

c) (Max 5 points) What makes the architecture of UltraSPARC T1 ("Niagara") different from most other processor architectures?

Solution sketch:
High focus on TLP, low focus on ILP. Poor single thread performance, but great multithread performance. Thread switch on any stall. Short pipeline, in-order, no branch prediction.

## Exercise 4) Memory, vector processors and networks (Max 15 points)

a) (Max 5 points) Briefly describe 5 different optimizations of cache performance.

Solution sketch:
(1 point pr. optimization) 6 techniques listed on page 291 in the text book, 11 more in 5.2 on page 293.

b) (Max 5 points) What makes vector processors fast at executing a vector operation?

Solution sketch:
A Vector operation can be executed with a single instruction, reducing code size and improving cache utilization. Further, the single instruction has no loop overhead and no control dependencies which a scalar processor would have. Hazard checks can also be done per vector, rather than per element. A vector processor also contains a deep pipeline especially designed for vector operations.

c) (Max 5 points) Discuss how the number of devices to be connected influences the choice of topology.

Solution sketch:
This is a classic example of performance vs. cost. Different topologies scale differently with respect to performance or cost as the number of devices grows. Crossbar scales performance well, but cost badly. Ring or bus scale performance badly, but cost well.

## Exercise 5) Multicore architectures and programming (Max 25 points)

a) (Max 6 points) Explain briefly the research method called design space exploration (DSE). When doing DSE, explain how a cache sensitive application can be made processor bound, and how it can be made bandwidth bound.

Solution sketch:
(Lecture 10-slide 4) DSE is to try out different points in an n-dimensional space of possible designs, where n is the number of different main design parameters, such as #cores, core-types (IO vs. OOO etc.), cache size etc. Cache sensitive applications can become processor bound by

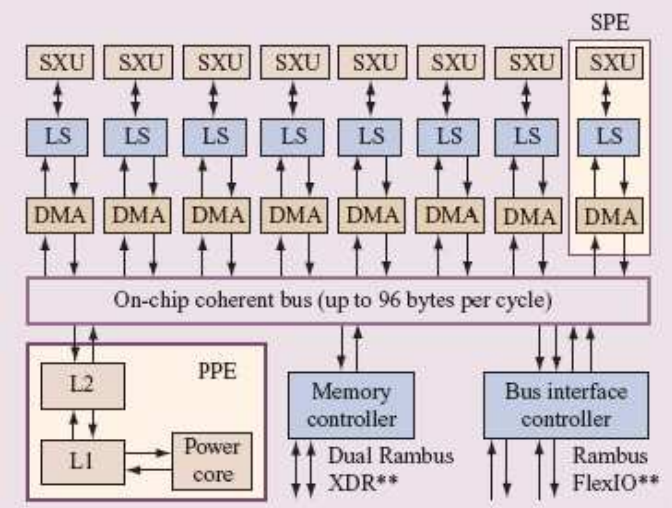
increasing the cache size, and they can be made bandwidth bound by decreasing it..

b) (Max 5 points) In connection with GPU-programming (shader programming), David Blythe uses the concept ”computational coherence”. Explain it briefly.

LF: See lecture 10, slide 36 + evt. the paper.

c) (Max 8 points) Give an overview of the architecture of the Cell processor.

Solution sketch:
All details of this figure are not expected, but the main elements.




* One main processor (Power-architecture, called PPE = Power processing element) – this acts as a host (master) processor. (Power arch., 64 bit, in-order two-issue superscalar, SMT (Simultaneous multithreading. Has a vector media extension (VMX) (Kahle figure 2))
* 8 identical SIMD processors (called SPE = Synergistic Processing element), each of these consists of SPU processing element (Synergistic processor unit) and local storage (LS, 256 KB SRAM --- not cache). On chip memory controller + bus interface. (Can operate on integers in different formats., 8, 16, 32 and floating point numbers in 32 og 64 bit. (64 bit floats in later version).
* Interconnect is a ring-bus (Element Interconnect Bus, EIB), connects PPE + 8 SPE. two unidirectional busses in each direction. Worst case latency is half distance, can support up to three simultaneous transfers
* Highly programmable DMA controller.

d) (Max 6 points) The Cell design team made several design decisions that were motivated by a wish to make it easier to develop programs with predictable (more deterministic) processing time (performance). Describe two of these.

Solution sketch:
1) They discarded the common out-of-order execution in the Power-processor, developed a simpler in-order processor

2) The local store memory (LS) in the SPE processing elements do not use HW cache-coherency snooping protocols to avoid the in-determinate nature of cache misses. The programmer handles memory in a more explicit way
3) Also the large number of registers (128) might help making the processing more deterministic wrt. execution time.
4) Extensive timers and counters (probably performance counters) (that may be used by the SW/programmer to monitor/adjust/control performance)

…---oooOOOooo---…