Advanced Computer Architecture  
21-09-2016

A) A server with 128GB RAM is built using 4 processors that embed each four cores that have private L1 caches, while two L2 caches are shared by each couple of cores; the single L3 cache is shared by the four cores. The caches obey the MESI protocol and have the following structure: 32KB, 4-way L1 I-cache and D-cache, each 32-byte block; 512KB, 4-way associative, 64-byte block L2 cache; 4MB, 8-way associative, 128-byte block shared L3 cache. The latencies (disregarding virtual memory TLB) expressed in clock cycles are: 4 in L1, 8 in L2, 24 in L3. Addresses are 48-bit long.

A Ram controller manages the access to a shared bus from each processor. The MESI protocol is enforced throughout the memory hierarchy.

a1) assuming initially empty and invalidated cache lines throughout the hierarchy, consider the following memory operations

core 1) ST F1, 0000FFFFA000hex;

core 2) LD F2, 0000FFFFA010hex;

core 3) LD F3, 0000FFFFA010hex

core 2) ST F2, 0000FFFFA010hex;

core 4) ST F4, 0000FFFFA000hex;

show the cache blocks involved throughout the memory hierarchy, the sequence of miss/hit in the hierarchy (assume writes are managed with allocate - write-back policy), and the blocks MESI states.

a2) considering an array of 1024 64-bit floating point elements X allocated in memory starting in such a way that X[1023] is located at address 0000FFFFA000hex and the following code fragment run on core 1)

*for (i=0;i<1023;i++) {X[1022-i)=X[1022-i]+X[1023-i]};*

determine the number of hits and misses in each level of the cache hierarchy.

B) The server RAM is built using 4x4GB DDR3 1600 chips attached to the 64-bit external bus of the processors (that run at 2.6GHz). The external bus allows a maximum transfer rate of 12 GB/sec.

b1) how many chips are necessary for the whole RAM?

b2) is the memory subsystem (bus+memory chips) well balanced?

b3) the DDR chips allow reading out a 32-byte memory word in a single activation; assuming that addressing takes 2 clock cycles and that memory activation takes 2 clocks, choose the memory organization that minimizes a cache miss, and compute the cost of the miss in processor clock cycles.

C) Each core of the processor is organized as a superscalar, 2-way pipeline, that fetches, decodes issues and retires (commits) bundles containing each 2 instructions. The front-end in-order section (fetch, decode\_A, decode\_B) consists of 3 stages. The issue logic takes 1 clock cycle, if the instructions in the bundle are independent, otherwise it takes 2 clock cycles. The architecture supports dynamic speculative execution, and control dependencies from branches are solved when the branch evaluates the condition, even if it is not at commit. The execution model obeys the attached state transition diagram. There is a functional units (FUs) Int1 for integer arithmetics (arithmetic and local instructions, branches and jumps, no multiplication, 2 FUS FAdd1-Fadd2 for floating point addition/subtraction, a FU FMolt1 for floating point multiplication, and a FU for division, FDiv1.

There are 12 integer (R0-R11) and 12 floating point (F0-F11) registers. Speculation is handled through a 8-entry ROB, a pool of 4 Reservation Stations (RS) Rs1-4 shared among all FUs, 1 load buffers Load1, 1 store buffer Store1 (see the attached execution model): an instruction bundle is first placed in the ROB (if two entries are available), then up to 2 instructions are dispatched to the shared RS (if available) when they are ready for execution and then executed in the proper FU. FUs are *blocking* and have the latencies quoted in the following table:

|  |  |
| --- | --- |
| Int - 2 | Fadd – 3 |
| Fmolt – 5 | Fdiv – 6 |

Further assumption

* The code is assumed to be already in the I-cache; data caches are described in point A) and are assumed empty and invalidated; the cost of a miss is conventionally set to 40.

c1) assuming a write-back protocol for cache management, show state transitions for the instructions of the first iteration of the following code fragment, that scans the 1024 floating elements, each 8-byte, of array W[] (assumed allocated at increasing addresses from 0000FFFFA000hex ) and replaces W[i] with B(W[i]+A), with A and B float constants (R0 is always 0).

PC01 OR R1,R0,0000FFFFA000hex -- set base address of W[0]

PC02 OR R5,R0,1023dec –- set loop terminating condition

PC03 OR R3,R0,R0 -- initialize loop controlling variable

PC04 LD F0,0(R1) -- load W[i]

PC05 ADDF F2,F1,F0 -- intermediate value: W[i]+A (A is preloaded in F1)

PC06 MULTF F3,F2,F4 -- new value B\*(W[i]+A) (B is preloaded in F4)

PC07 ST F3,0(R1) -- new value written back

PC08 ADD R3,R3,1 -- increase loop controlling variable

PC09 ADD R1,R1,8 -- advance pointer into array W

PC10 BL R5,R3,PC04 -- testing for loop exit condition

c2) show ROB, RS and buffer status at the issue of PC04 in the second iteration.

c3) assuming the functional units for float multiplications are “boosted” through an enhanced clock at 3.2 Ghz; what is the speed-up to be expected?

Dynamic speculative execution

Decoupled ROB RS execution mode

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ISTRUCTION** |  |  | **INSTRUCTION STATE** | | | | | | |
|  | **n. ite** | **ROB pos** | **WO** | **RE** | **DI** | **EX** | **WB** | **RR** | **CO** |
| PC01 OR R1,R0,00000FFFFA000hex |  |  |  |  |  |  |  |  |  |
| PC02 OR R5,R0,1023dec |  |  |  |  |  |  |  |  |  |
| PC03 OR R3,R0,R0 |  |  |  |  |  |  |  |  |  |
| PC04 LD F0,0(R1) |  |  |  |  |  |  |  |  |  |
| PC05 ADDF F2,F1,F0 |  |  |  |  |  |  |  |  |  |
| PC06 MULTF F3,F2,F4 |  |  |  |  |  |  |  |  |  |
| PC07 ST F3,0(R1) |  |  |  |  |  |  |  |  |  |
| PC08 ADD R3,R3,1 |  |  |  |  |  |  |  |  |  |
| PC09 ADD R1,R1,8 |  |  |  |  |  |  |  |  |  |
| PC10 BL R5,R3,PC04 |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Reservation station and load/store buffers | | | | | | | |
| Busy | Op | Vj | Vk | ROBj | ROBk | ROB pos | Address |
| Rs1 |  |  |  |  |  |  |  |  |
| Rs2 |  |  |  |  |  |  |  |  |
| Rs3 |  |  |  |  |  |  |  |  |
| Rs4 |  |  |  |  |  |  |  |  |
| Load1 |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| Store1 |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

ROBj ROBk: sources not yet available

ROB pos: ROB entry number where instruction is located

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Result Register status | | | | | | | | | | | | | |
| Integer | R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 | R8 | R9 | R10 | R11 |  |  |
| ROB pos |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| state |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Float. | F0 | F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F9 | F10 | F11 |  |  |
| ROB pos |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| state |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Reorder Buffer (ROB) | | | | | |
| ROB Entry# | Busy | Op | Status | Destination | Value |
| 0 |  |  |  |  |  |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |
| 4 |  |  |  |  |  |
| 5 |  |  |  |  |  |
| 6 |  |  |  |  |  |
| 7 |  |  |  |  |  |

DI

EX

WB

RR

RE

CO

wo

QUEUE

**Decoupled execution model for bundled (paired) instructions**

The state diagram depicts the model for a dynamically scheduled, speculative execution microarchitecture equipped with a Reorder Buffer (ROB) and a set of Reservation Stations (RS). The ROB and RSs are allocated during the ISSUE phase, denoted as RAT (Register Alias Allocation Table) in INTEL microarchitectures, as follows: a bundle (2 instructions) if fetched from the QUEUE of decoded instructions and ISSUED if there is a free couple of consecutive entries in the ROB ( head and tail of the ROB queue do not match); a maximum of two instructions are moved into the RS (if available) when all of their operands are available. Access memory instructions are allocated in the ROB and then moved to a load/store buffer (if available) when operands (address and data, if proper) are available .

**States** are labelled as follows:

WO: Waiting for Operands (at least one of the operands is not available)

RE: Ready for Execution (all operands are available)

DI: Dispatched (posted to a free RS or load/store buffer)

EX: Execution (moved to a load/store buffer or to a matching and free UF)

WB: Write Back (result is ready and is returned to the Rob by using in exclusive mode the Common Data Bus CDB)

RR: Ready to Retire (result available or STORE has completed)

CO: Commit (result is copied to the final ISA register)

**State transitions** happen at the following events:

*from* QUEUE *to* WO: ROB entry available, operand missing

*from* QUEUE *to* RE: ROB entry available, all operands available

*loop at* WO: waiting for operand(s)

*from* WO *to* RE: all operands available

*loop at* RE: waiting for a free RS or load/store buffer

*from* RE *to* DI: RS or load/store buffer available

*loop on* DI: waiting for a free UF

*from* DI *to* EX: UF available

*loop at* EX: multi-cycle execution in a UF, or waiting for CDB

*from* EX *to* WB: result written to the ROB with exclusive use of CDB

*from* EX *to* RR: STORE completed, branch evaluted

*loop at* RR: instruction completed, not at the head of the ROB, or bundled with a not RR instruction

*from* RR *to* CO: bundle of RR instructions at the head of the ROB, no exception raised

**Resources***Register-to-Register* instructions hold resources as follows:

ROB: from state WO (or RE) up to CO, inclusive;

RS: state DI

UF: EX and WB

*Load/Store* instructions hold resources as follows:

ROB: from state WO (or RE) up to CO, inclusive;

Load buffer: from state WO (or RE) up to WB

Store buffer: from state WO (or RE) up to EX (do not use WB)

**Forwarding**: a write on the CDB (WB) makes the operand available to the consumer in the same clock cycle. If the consumer is doing a state transition from QUEUE to WO or RE, that operand is made available; if the consumer is in WO, it goes to RE in the same clock cycle of WB for the producer.

**Branches**: they compute Next-PC and the branch condition in EX and optionally forward Next-PC to the “in-order” section of the pipeline (Fetch states) in the next clock cycle. They do not enter WB and go to RR instead.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Standard name** | **Memory clock**  **(MHz)** | **Cycle time**  **(ns)** | **I/O bus clock**  **(MHz)** | **Data rate**  **(**[**MT/s**](http://en.wikipedia.org/wiki/Transfer_(computing))**)** | **Module name** | **Peak transfer rate**  **(MB/s)** |
| DDR3-800D DDR3-800E | 100 | 10 | 400 | 800 | PC3-6400 | 6400 |
| DDR3-1066E DDR3-1066F DDR3-1066G | 133⅓ | 7 1⁄2 | 533⅓ | 1066⅔ | PC3-8500 | 8533⅓ |
| DDR3-1333F\* DDR3-1333G DDR3-1333H DDR3-1333J\* | 166⅔ | 6 | 666⅔ | 1333⅓ | PC3-10600 | 10666⅔ |
| DDR3-1600G\* DDR3-1600H DDR3-1600J DDR3-1600K | 200 | 5 | 800 | 1600 | PC3-12800 | 12800 |
| DDR3-1866J\* DDR3-1866K DDR3-1866L DDR3-1866M\* | 233⅓ | 4 2⁄7 | 933⅓ | 1866⅔ | PC3-14900 | 14933⅓ |
| DDR3-2133K\* DDR3-2133L DDR3-2133M DDR3-2133N\* | 266⅔ | 3 3⁄4 | 1066⅔ | 2133⅓ | PC3-17000 | 17066⅔ |

DDR3 standard JEDEC specification (source Wikipedia)