Design of Parallel and High Performance Computing
HS 2016
Torsten Hoefler, Markus Püschel **Homework 2**
Department of Computer Science
ETH Zurich

Cache addressing

In the following we will assume we are working with a bus-based multiprocessor machine with four processors. Each processor has its own direct-mapped cache. Each cache is 32 Bytes in size, organized in four sets. The memory in the machine is byte-addressable and the address width of the machine is eight bit.

Show how cache-addressing works on this machine. Which part of an address is used as tag, set and offset?

MESI State Transitions

Describe what happens in the MESI protocol (bus traffic, state changes) if a processor experiences

1. a local read miss, while another cache holds a copy in exclusive state
2. a local read miss, while another cache has a copy in modified state
3. a local write hit, while the cacheline is in modified state
4. a local write hit, while the cacheline is in exclusive state
5. a local write hit, while the cacheline is in shared state in several caches

Cache coherence

The machine described above uses the MESI protocol to maintain cache coherence. In the following $R_p(a)$ means that processor p reads one byte from the address a. $W_p(a) = v$ means processor p writes v to the memory location a. Addresses are represented in binary. Initially all cache lines are invalid and all memory locations contain 0.

Show the transitions of cache lines in the table below.

Action	P_0	P_1	P_2	P_3	Latency
$R_0(00000000)$					
$R_1(00000010)$					
$R_2(00000011)$					
$W_3(00000100) = 1$					
$W_3(00000101) = 2$					
$R_0(00000100)$					
$W_0(00000101) = 3$					

Assume a cache hit takes one cycle (both for read and write operations), transfering a cache line (either fetch or write back) takes 8 cycles, and a request to use a cache line exclusively (transition I - M) resp. a request to uprade to exclusive use (transition S - M) takes 2 cycles. What is the penalty of this code sequence over the best alternative execution sequence that you can think of?

False sharing

Write a program which demonstrates *false sharing*. Provide two versions of the same algorithm, one version where false sharing is likely to occur, one where it does not. Meassure the execution times of both versions to check if they differ significantly.