4th Homework for Computer Architecture

Submission deadline: Dec 29 , 11：55pm

Read Chapter 4 and 5 then do the following problems.

(Total 110 points)

In 6th Edition

4.13 5.1 5.23

4.13 [10/15] <4.4> Assume a GPU architecture that contains 10 SIMD processors. Each SIMD instruction has a width of 32 and each SIMD processor contains 8 lanes for single-precision arithmetic and load/store instructions, meaning that each nondiverged SIMD instruction can produce 32 results every 4 cycles. Assume a kernel that has divergent branches that causes, on average, 80% of threads to be active. Assume that 70% of all SIMD instructions executed are single-precision arithmetic and 20% are load/store. Because not all memory latencies are covered, assume an average SIMD instruction issue rate of 0.85. Assume that the GPU has a clock speed of 1.5 GHz.

a. [10] <4.4> Compute the throughput, in GFLOP/s, for this kernel on this GPU.

b. [15] <4.4> Assume that you have the following choices:
   (1) Increasing the number of single-precision lanes to 16
   (2) Increasing the number of SIMD processors to 15 (assume this change doesn't affect any other performance metrics and that the code scales to the additional processors)
   (3) Adding a cache that will effectively reduce memory latency by 40%, which will increase instruction issue rate to 0.95

What is speedup in throughput for each of these improvements?

a. $1.5\ \text{GHz} \times 0.80 \times 0.85 \times 0.70 \times 10\ \text{cores} \times 32/4 = 57.12\ \text{GFLOPs/s}$

b. **Option 1:**
$1.5\ \text{GHz} \times 0.80 \times 0.85 \times 0.70 \times 10\ \text{cores} \times 32/2 = 114.24\ \text{GFLOPs/s}$
$(\text{speedup} = 114.24/57.12 = 2)$
Option 2:
$1.5\ \text{GHz} \times 0.80 \times 0.85 \times 0.70 \times 15\ \text{cores} \times 32/4 = 85.68\ \text{GFLOPs/s}$
$(\text{speedup} = 85.68/57.12 = 1.5)$
Option 3:
$1.5\ \text{GHz} \times 0.80 \times 0.95 \times 0.70 \times 10\ \text{cores} \times 32/4 = 63.84\ \text{GFLOPs/s}$
$(\text{speedup} = 63.84/57.12 = 1.11)$
Option 3 is best.

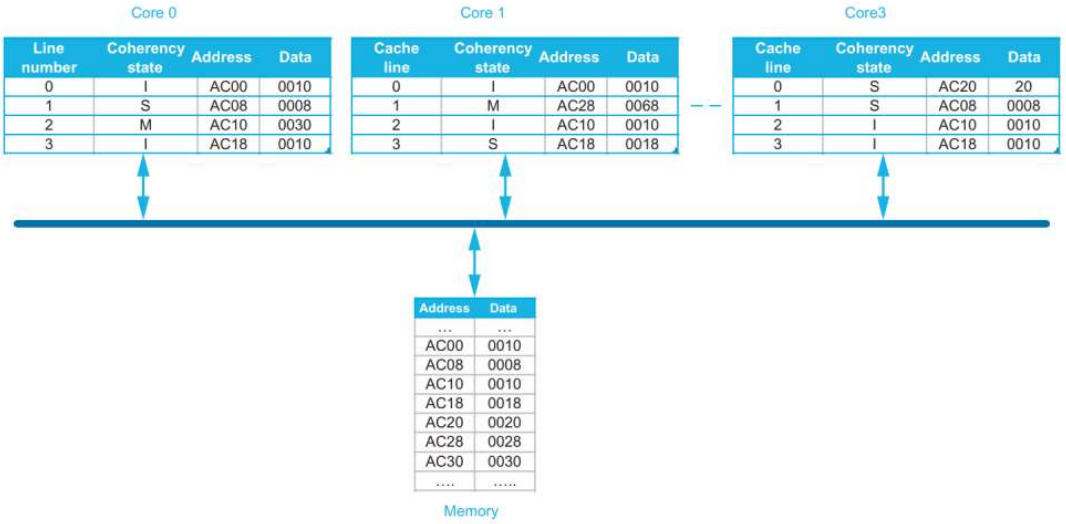

Figure 5.37 Multicore (point-to-point) multiprocessor.

5.1. [10/10/10/10/10/10/10] <5.2> For each part of this exercise, the initial cache and memory state are assumed to initially have the contents shown in Figure 5.37. Each part of this exercise specifies a sequence of one or more CPU operations of the form

`Ccore#: R, <address>` for reads

and

`Ccore#: W, <address> <-- <value written>` for writes.

For example,

`C3: R, AC10 & C0: W, AC18 <-- 0018`

Read and write operations are for 1 byte at a time. Show the resulting state (i.e., coherence state, tags, and data) of the caches and memory after the actions given below. Show only the cache lines that experience some state change; for example:

`C0.L0: (I, AC20, 0001)` indicates that line 0 in core 0 assumes an "invalid" coherence state (`I`), stores `AC20` from the memory, and has data contents `0001`. Furthermore, represent any changes to the memory state as `M: <address> <- value`.

Different parts (a) through (g) do not depend on one another: assume the actions in all parts are applied to the initial cache and memory states.

a. [10] <5.2> `C0: R, AC20`

b. [10] <5.2> `C0: W, AC20 <-- 80`

c. [10] <5.2> `C3: W, AC20 <-- 80`

d. [10] <5.2> `C1: R, AC10`

e. [10] <5.2> `C0: W, AC08 <-- 48`

f. [10] <5.2> `C0: W, AC30 <-- 78`

g. [10] <5.2> `C3: W, AC30 <-- 78`

Answer	
a	C0.L0:(S, AC20, 0020) return 0020
b	C3.L0:(I, AC20, 0020) C0.L0:(M, AC20, 0080)
c	C3.L0:(M, AC20, 0080)
d	C0.L2:(S, AC10, 0030) M: AC10 <- 0030 C1.L2:(S, AC10, 0030) return 0030
e	C3.L1:(I, AC08, 0008) C0.L1:(M, AC08, 0048)
f	M: AC10 <- 0030 C0.L2:(M, AC30, 0078)
g	C3.L2:(M, AC30, 0078)

5.23. [15] <5.2> Show how the basic snooping protocol of Figure 5.6 can be changed for a write-through cache. What is the major hardware functionality that is not needed with a write-through cache compared with a write-back cache?

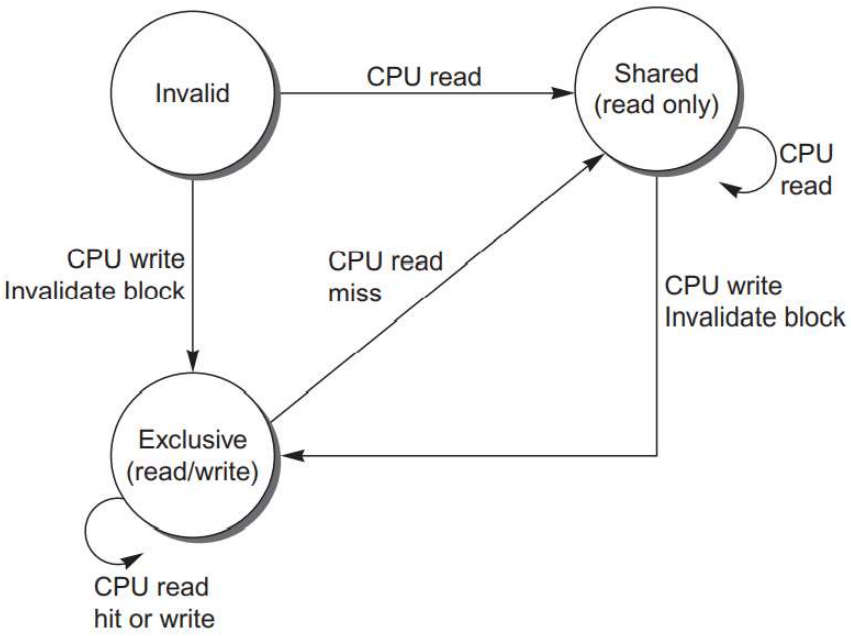

Figure S.3 CPU portion of the simple cache coherency protocol for write-through caches.

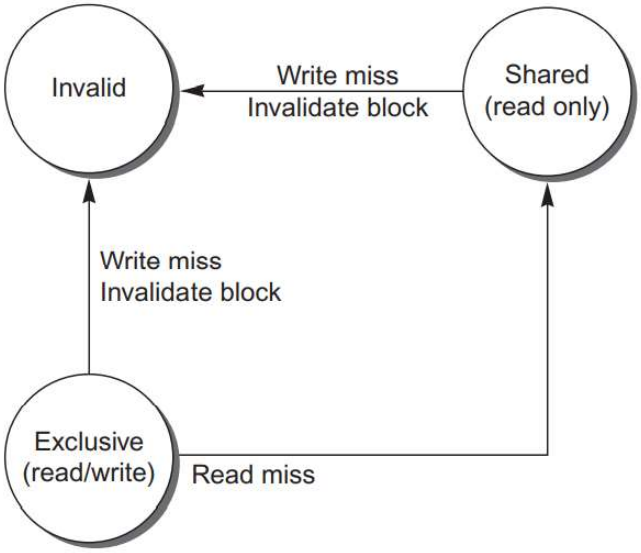

Figure S.4 Bus portion of the simple cache coherency protocol for write-through caches.