By 3220104364 王晓宇

1

> *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. This means that each non-diverged SIMD instruction **can produce 32 results every 4 cycles**. Assume a kernel that has divergent branches causing, 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)**Compute the throughput, in GFLOP/s, for this kernel on this GPU.*

$$1.5GHz \times 10 \times 8 \times 80\% \times 70\% \times 0.85 = 57.12GFLOP/s$$

> ***(b)**Assume that you have the following choices:*
>
> 1. *Increasing the number of single-precision lanes to **16**.*

$$1.5GHz \times 10 \times 16 \times 80\% \times 70\% \times 0.85 = 114.24GFLOP/s$$

> 2. *Increasing the number of SIMD processors to **15** (assume this change doesn't affect any other performance metrics and that the code scales tothe additional processors).*

$$1.5GHz \times 15 \times 8 \times 80\% \times 70\% \times 0.85 = 85.68GFLOP/s$$

> 3. *Adding a cache that will effectively reduce memory latency by **40%**, which will increase the instruction issue rate to **0.95**.*

$$1.5GHz \times 10 \times 8 \times 80\% \times 70\% \times 0.95 = 63.84GFLOP/s$$

> *What is the speedup in throughput for each of these improvements?*

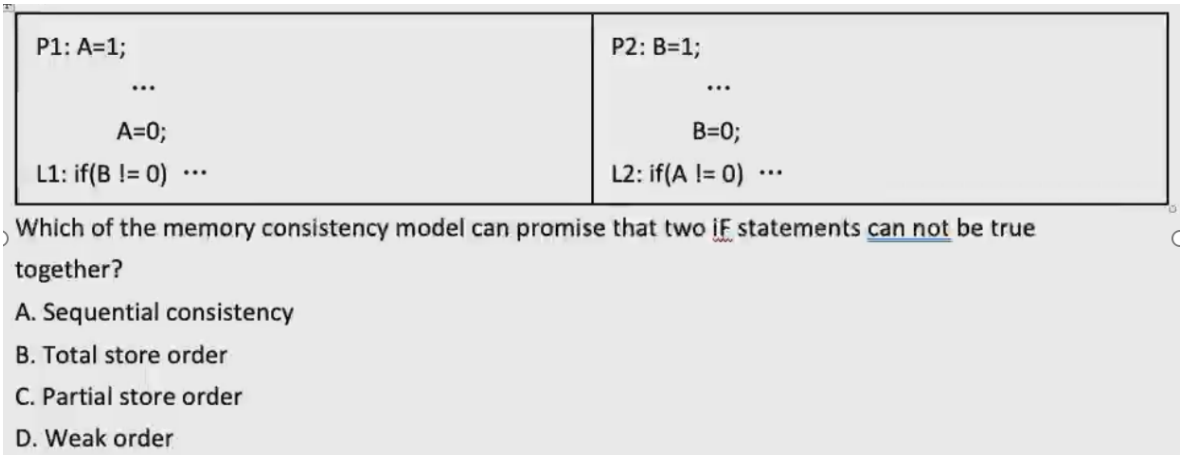
$$\text{speedup}_1 = \frac{114.24}{57.12} = 2$$
$$\text{speedup}_2 = \frac{85.68}{57.12} = 1.5$$
$$\text{speedup}_3 = \frac{63.84}{57.12} = 1.11$$

2

> ```
> for (int i = 0; i < 524; i++) {
> c[i] = a[i] + b[i];
> }
> ```
>
> *Consider the code above, arrays **a**, **b**, and **c** each have **524 8-byte-wide integer elements**.*
>
> *Assume that:*
>
> *A **64-bit-wide integer add instruction** takes **1 clock cycle**.*
>
> *A **512-bit-wide vector add instruction** takes **4 clock cycles**.*
>
> ***Question:***
>
> *How many clock cycles do we need to finish the above calculation using **512-bit** wide vector add instructions?*

$$\frac{512bits}{8 \times 8bits} = 8\text{elements}$$
$$\left\lceil \frac{524}{8} \right\rceil \times 4 = 264cycles$$

3



P1: A=1;	P2: B=1;
…	…
A=0;	B=0;
L1: if(B != 0) …	L2: if(A != 0) …

Which of the memory consistency model can promise that two IF statements can not be true together?

A. Sequential consistency

B. Total store order

C. Partial store order

D. Weak order

The answer is **A**

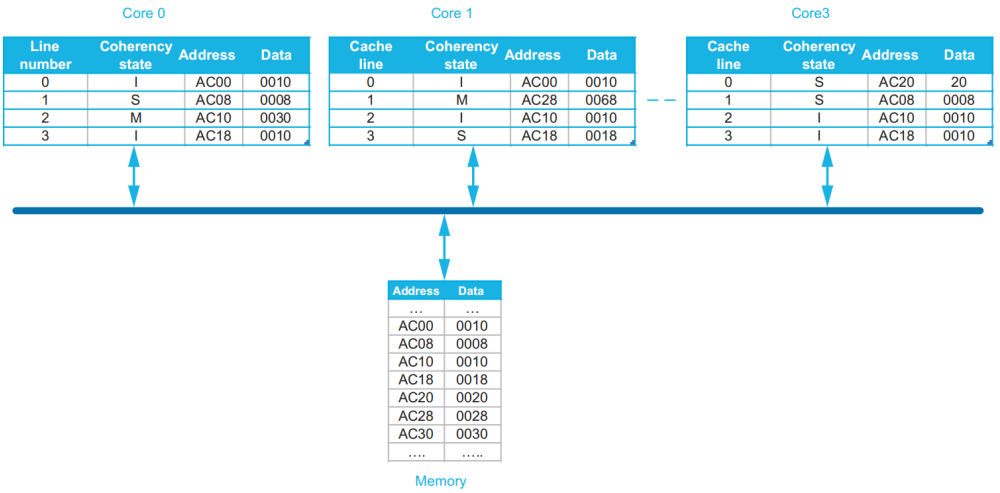
> *Which technique is not serving for data-level **parallelism** (DLP)?*
>
> *A. SIMD*
>
> *B. Dynamic Scheduling*
>
> *C. Vector process*
>
> *D. GPU*

The answer is **B**

> *Which of the following techniques may get **CPI (cycle per instruction)** smaller than 1?*
>
> *A. Superscalar*
>
> *B. Dynamic Scheduling*
>
> *C. Software pipelining*
>
> *D. Speculation based on Tomasulo*

The answer is **A**

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`

INDEX	ANSWER
a	C0.0: (S, AC20, 0020), we can get 0020
b	C0.0: (M, AC20, 0080),C3.0: (I, AC20, 0020)
c	C3.0: (M, AC20, 0080)
d	C1.2: (S, AC10, 0010), we can get 0010
e	C0.1 (M, AC08, 0048),C3.1: (I, AC08, 0008)
f	C0.2: (M, AC30, 0078),Memory: AC10<-0030 (write-back to memory)

5

> *Consider the following code, which multiplies two vectors that contain single precision complex values:*
>
> ```
> for (i = 0; i < 300; i++) {
> c_re[i] = a_re[i] * b_re[i] - a_im[i] * b_im[i];
> c_im[i] = a_re[i] * b_im[i] + a_im[i] * b_re[i];
> }
> ```
>
> *Assume that the processor runs at 700 MHz and has a maximum vector length of 64. The load/store unit has a start-up overhead of 15 cycles; the multiply unit, 8 cycles; and the add/subtract unit, 5 cycles.*

> *a. Assuming chaining and a single memory pipeline, how many chimes are required?*
>
> *How many clock cycles are required per complex result value, including start-up overhead?*

CONVOY_INDEX	INST	COMMENT
1	mulvv.s + lv	a_re * b_re & load a_im
2	lv + mulvv.s	load b_im & a_im*b_im
3	subvv.s + sv	subtract & store c _re
4	mulvv.s + lv	a_re * b_im & load next a_re iter
5	mulvv.s + lv	a_im*b_re & load next b_re iter
6	addvv.s + sv	add & store c_im

So we have 6 chimes.

$$15\text{cycles} \times 6 + 8\text{cycles} \times 4 + 5\text{cycles} \times 2 = 132\text{cycles}$$

$$\frac{132 + 6\text{chimes} \times 64\text{elements}}{64} = 8.0625$$

> *b. Now assume that the processor has three memory pipelines and chaining.*
>
> *If there are no bank conflicts in the loop's accesses, how many clock cycles are required per result?*

CONVOY_INDEX	INST	COMMENT
1	mulvv.s	a_re * b_re

CONVOY_INDEX	INST	COMMENT
2	mulvv.s	a_im*b_im
3	subvv.s + sv	subtract & store c _re
4	mulvv.s	a_re * b_im
5	mulvv.s + lv	a_im*b_re & load next a_re
6	addvv.s + sv + lv + lv + lv	add & store c_im & load next b_re,a_im,b_im

$$15\text{cycles} \times 6 + 8\text{cycles} \times 4 + 5\text{cycles} \times 2 = 132\text{cycles}$$

$$\frac{132 + 6\text{chimes} \times 64\text{elements}}{64} = 8.0625$$