CA Homework#5

b07902076 資工三 許世儒

Handwritten

4.31.1

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
li x12,0	IF	ID	EX	ME	WB																				
jal ENT		IF	ID	EX	ME	WB																			
bne x12,13,TOP			IF	ID	EX	ME	WB																		
sli x5,x12,3				IF	ID	EX	ME	WB																	
add x6, x10, x5					IF	ID	EX	ME	WB																
ld x7, 0(x6)						IF	ID	EX	ME	WB															
ld x29, 8(x6)							IF	ID	EX	ME	WB														
sub x30, x7, x29									IF	ID	EX	ME	WB												
add x31, x11, x5										IF	ID	EX	ME	WB											
sd x30, 0(x31)											IF	ID	EX	ME	WB										
addi x12, x12, 2											IF	ID	EX	ME	WB										
bne x12, x13, TOP												IF	ID	EX	ME	WB									
sli x5, x12, 3													IF	ID	EX	ME	WB								
add x6, x10, x5														IF	ID	EX	ME	WB							
ld x7, 0(x6)															IF	ID	EX	ME	WB						
ld x29, 8(x6)																IF	ID	EX	ME	WB					
sub x30, x7, x29																		IF	ID	EX	ME	WB			
add x31, x11, x5																			IF	ID	EX	ME	WB		
sd x30, 0(x31)																				IF	ID	EX	ME	WB	
addi x12, x12, 2																				IF	ID	EX	ME	WB	
bne x12, x13, TOP																					IF	ID	EX	ME	WB

4.31.2

For one iteration,

one-issue processor: 10 cycles

two-issue processor: 9 cycles

Therefore, speedup $= \frac{10}{9} \approx 1.11$

4.31.3

put addi x12, x12, 2 after ld x29, 8(x6) to avoid stalling and reduce one cycle

```
  beqz x13, DONE
  li  x12, 0
TOP:
  slli x5, x12, 3
  add x6, x10, x5
  ld x7, 0(x6)
  ld x29, 8(x6)
  addi x12, x12, 2
  sub x30, x7, x29
  add x31, x11, x5
  sd x30, 0(x11)
  bne x12, x13, TOP
DONE:
```

4.31.4

```
  beqz x13, DONE
  li x12, 0
TOP:
  slli x5, x12, 3
  add x6, x10, x5
  ld x7, 0(x6)
  add x31, x11, x5
  ld x29, 8(x6)
  addi x12, x12, 2
  sub x30, x7, x29
  sd x30, 0(x31)
  bne x12, x13, TOP
DONE:
```

4.31.5

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
beqz x13, DONE	IF	ID	EX	ME	WB														
li x12, 0	IF	ID	EX	ME	WB														
slli x5, x12, 3		IF	ID	EX	ME	WB													
add x6, x10, x5			IF	ID	EX	ME	WB												
ld x7, 0(x6)				IF	ID	EX	ME	WB											
add x31, x11, x5				IF	ID	EX	ME	WB											
ld x29, 8(x6)					IF	ID	EX	ME	WB										
addi x12, x12, 2					IF	ID	EX	ME	WB										
sub x30, x7, x29							IF	ID	EX	ME	WB								
sd x30, 0(x31)								IF	ID	EX	ME	WB							
bne x12, x13, TOP								IF	ID	EX	ME	WB							
slli x5, x12, 3									IF	ID	EX	ME	WB						
add x6, x10, x5										IF	ID	EX	ME	WB					
ld x7, 0(x6)											IF	ID	EX	ME	WB				
add x31, x11, x5											IF	ID	EX	ME	WB				
ld x29, 8(x6)												IF	ID	EX	ME	WB			
addi x12, x12, 2												IF	ID	EX	ME	WB			
sub x30, x7, x29														IF	ID	EX	ME	WB	
sd x30, 0(x31)															IF	ID	EX	ME	WB
bne x12, x13, TOP															IF	ID	EX	ME	WB

4.31.6

For one instruction,

one-issue processor: 9 cycles

two-issue processor: 7 cycles

speedup: $\frac{9}{7} \approx 1.29$

4.31.7

Assume that register x14 and x15 can be used safely.

```
  beqz x13, DONE
  li x12, 0
TOP:
  slli x5, x12, 3
  add x6, x10, x5
  ld x7, 0(x6)
  ld x29, 8(x6)
  ld x14, 16(x6)
  ld x15, 24(x6)
  addi x12, x12, 4
  sub x30, x7, x29
  add x31, x11, x5
  sd x30, 0(x31)
  sub x30, x15, x14
  sd x30, 16(x31)
  bne x12, x13, TOP
DONE:
```

4.31.8

Assume that register x14 and x15 can be used safely.

```
  beqz x13, DONE
  li x12, 0
  addi x6, x10, 0
TOP:
  ld x7, 0(x6)
  add x31, x11, x5
  ld x29, 8(x6)
  addi x12, x12, 4
  ld x14, 16(x6)
  slli x5, x12, 3
  ld x15, 24(x6)
  sub x30, x7, x29
  sd x30, 0(x31)
  sub x11, x15, x14
  sd x11, 8(x31)
  add x6, x15, x5
  bne x12, x13, TOP
```

4.31.9

For one iteration,
one-issue processor: 13 cycles
two-issue processor: 8 cycles
speed up $= \frac{13}{8} = 1.625$

4.31.10

We can no longer improve the code from 4.31.8 because every two instructions are processed in one cycle except for bne instruction in each iteration.

5.5.1

For each word is 8-byte. There are 5 bits of total offsets. Hence, there are 2^5 bytes, so it's **four 8-byte words**.

5.5.2

There are five bits for index. Therefore, there are $2^5 = 32$ blocks.

5.5.3

Total bits $\Rightarrow$ 32(blocks) $\times$ (1 valid bit + 54 tag bits + 32 $\times$ 8 data bits) = 32 $\times$ 311 bits
Data bits $\Rightarrow$ 32(blocks) $\times$ 32 $\times$ 8 data bits = 32 $\times$ 256 bits
Ratio $= \frac{311}{256} \approx 1.215$

5.5.4

Address	Tag	Index	Offset	Hit/Miss	Bytes replaced
0x00	0x0	0x00	0x00	Miss	
0x04	0x0	0x00	0x04	Hit	
0x10	0x0	0x00	0x10	Hit	
0x84	0x0	0x04	0x04	Miss	
0xE8	0x0	0x07	0x08	Miss	
0xA0	0x0	0x05	0x00	Miss	
0x400	0x1	0x00	0x00	Miss	0x00-0x1F
0x1E	0x0	0x00	0x1E	Miss	0x400-0x41F
0x8C	0x0	0x04	0x0C	Hit	
0xC1C	0x3	0x00	0x1C	Miss	0x00-0x1F
0xB4	0x0	0x05	0x14	Hit	
0x884	0x2	0x04	0x04	Miss	0x80-0x9F

5.5.5

Hit Ratio $= \frac{4}{12} \approx 0.333$

5.5.6

<0, 3, Mem[0xC00]-Mem[0xC1F]>
<4, 2, Mem[0x880]-Mem[0x89F]>
<5, 0, Mem[0x0A0]-Mem[0x0BF]>
<7, 0, Mem[0x0E0]-Mem[0x0FF]>

5.10.1

Clock rate for P1 $= \frac{1}{0.66 \times 10^{-9}} = 1.15GHz$
Clock rate for P2 $= \frac{1}{0.9 \times 10^{-9}} = 1.11GHz$

5.10.2

For P1: $1 + 8\% \times \lceil \frac{70}{0.66} \rceil = 9.56$ cycles
For P2: $1 + 6\% \times \lceil \frac{70}{0.90} \rceil = 5.68$ cycles

5.10.3

For P1: instruction cache miss + data cache miss $= 1 + 8\% \times \lceil \frac{70}{0.66} \rceil + 36\% \times 8\% \times \lceil \frac{70}{0.66} \rceil = 12.64$
For P2: instruction cache miss + data cache miss $= 1 + 6\% \times \lceil \frac{70}{0.90} \rceil + 36\% \times 6\% \times \lceil \frac{70}{0.90} \rceil = 7.36$
CPI for P1 $= 12.64$ and CPI for P2 $= 7.36$
For P1 per instruction: $0.66 \times 12.64 = 8.34$(ns)
For P2 per instruction: $0.9 \times 7.36 = 6.62$(ns)
Therefore, P2 is faster.

5.10.4

$1 + 8\% \times (\lceil \frac{5.62}{0.66} \rceil) + 95\% \times \lceil \frac{70}{0.66} \rceil) = 9.85$ cycles
Therefore, it's worse with L2 cache.

5.10.5

instruction cache miss + data cache miss
$= 1 + 8\% \times (\lceil \frac{5.62}{0.66} \rceil) + 95\% \times \lceil \frac{70}{0.66} \rceil) + 36\% \times 8\% \times (\lceil \frac{5.62}{0.66} \rceil + 95\% \times \lceil \frac{70}{0.66} \rceil)$
$= 13.04$

5.10.6

Assume that the miss rate of L2 cache is p
$1 + 8\% \times (\lceil \frac{5.62}{0.66} \rceil) + p \times \lceil \frac{70}{0.66} \rceil) + 36\% \times 8\% \times (\lceil \frac{5.62}{0.66} \rceil + p \times \lceil \frac{70}{0.66} \rceil) < 12.64$
$\Rightarrow p < 0.916$

5.10.7

Assume that the miss rate of L2 cache is p
$0.66[1 + 8\% \times (\lceil \frac{5.62}{0.66} \rceil) + p \times \lceil \frac{70}{0.66} \rceil) + 36\% \times 8\% \times (\lceil \frac{5.62}{0.66} \rceil + p \times \lceil \frac{70}{0.66} \rceil)] < 6.62$
$\Rightarrow p < 0.692$

5.16.1

Address	TLB hit/miss	Page table hit/miss	TLB			
			Valid	Last access	Tag	Physical Page
0x123d	Miss	Miss (Page fault)	1	5	0xb	12
			1	2	0x7	4
			1	4	0x3	6
			1	0	0x1	13
0x08b3	Miss	Hit	1	0	0x0	5
			1	3	0x7	4
			1	5	0x3	6
			1	1	0x1	13
0x365c	Hit	--	1	1	0x0	5
			1	4	0x7	4
			1	0	0x3	6
			1	2	0x1	13
0x871b	Miss	Miss (Page fault)	1	2	0x0	5
			1	0	0x8	14
			1	1	0x3	6
			1	3	0x1	13
0xbee6	Miss	Hit	1	3	0x0	5
			1	1	0x8	14
			1	2	0x3	6
			1	0	0xb	12
0x3140	Hit	--	1	4	0x0	5
			1	2	0x8	14
			1	0	0x3	6
			1	1	0xb	12
0xc049	Miss	Miss (Page fault)	1	0	0xc	15
			1	3	0x8	14
			1	1	0x3	6
			1	2	0xb	12

5.16.2

Address	TLB hit/miss	Page table hit/miss	TLB			
			Valid	Last access	Tag	Physical Page
0x123d	Miss	Hit	1	5	0xb	12
			1	2	0x7	4
			1	4	0x3	6
			1	0	0x0	5
0x08b3	Hit	--	1	6	0xb	12
			1	3	0x7	4
			1	5	0x3	6
			1	0	0x0	5
0x365c	Hit	--	1	7	0xb	12
			1	4	0x7	4
			1	6	0x3	6
			1	0	0x0	5
0x871b	Miss	Miss (Page fault)	1	0	0x2	13
			1	5	0x7	4
			1	7	0x3	6
			1	1	0x0	5
0xbee6	Hit	--	1	0	0x2	13
			1	6	0x7	4
			1	8	0x3	6
			1	2	0x0	5
0x3140	Hit	--	1	1	0x2	13
			1	7	0x7	4
			1	9	0x3	6
			1	0	0x0	5
0xc049	Hit	--	1	2	0x2	13
			1	8	0x7	4
			1	0	0x3	6
			1	1	0x0	5

Advantages: the miss rate of TLB becomes lower

Disadvantages: fragmentation becomes higher and utilization of the physical memory becomes lower

5.16.3

Address	TLB hit/miss	Page table hit/miss	TLB				
			Index	Last access	Valid	Tag	Physical Page
0x123d	Miss	Miss (Page fault)	0	5	1	0xb	12
				2	1	0x7	4
			1	4	1	0x3	6
				0	1	0x1	13
0x08b3	Miss	Hit	0	0	1	0x0	5
				3	1	0x7	4
			1	5	1	0x3	6
				1	1	0x1	13
0x365c	Hit	--	0	1	1	0x0	5
				4	1	0x7	4
			1	0	1	0x3	6
				2	1	0x1	13
0x871b	Miss	Miss (Page fault)	0	2	1	0x0	5
				0	1	0x8	14
			1	1	1	0x3	6
				3	1	0x1	13
0xbee6	Miss	Hit	0	3	1	0x0	5
				1	1	0x8	14
			1	2	1	0x3	6
				0	1	0xb	12
0x3140	Hit	--	0	4	1	0x0	5
				2	1	0x8	14
			1	0	1	0x3	6
				1	1	0xb	12
0xc049	Miss	Miss (Page fault)	0	0	1	0xc	15
				2	1	0x8	14
			1	0	1	0x3	6
				1	1	0xb	12

5.16.4

Address	TLB hit/miss	Page table hit/miss	TLB				
			Index	Last access	Valid	Tag	Physical Page
0x123d	Miss	Miss (Page fault)	0	5	1	0xb	12
			1	0	1	0x0	13
			2	4	1	0x3	6
			3	8	0	0x4	9
0x08b3	Miss	Hit	0	0	1	0x0	5
			1	1	1	0x0	13
			2	5	1	0x3	6
			3	9	0	0x4	9
0x365c	Miss	Hit	0	1	1	0x0	5
			1	2	1	0x0	13
			2	6	1	0x3	6
			3	0	1	0x0	6
0x871b	Miss	Miss (Page fault)	0	0	1	0x2	14
			1	3	1	0x0	13
			2	7	1	0x3	6
			3	1	1	0x0	6
0xbee6	Miss	Hit	0	1	1	0x2	14
			1	4	1	0x0	13
			2	8	1	0x3	6
			3	0	1	0x2	12
0x3140	Miss	Hit	0	2	1	0x2	14
			1	5	1	0x0	13
			2	9	1	0x3	6
			3	0	1	0x0	6
0xc049	Miss	Miss (Page fault)	0	0	1	0x3	15
			1	6	1	0x0	13
			2	10	1	0x3	6
			3	1	1	0x0	6

5.16.5

TLB can help CPU to speed up (if TLB hit) the virtual address translation without looking up the page table. Without TLB, 2 memory accesses are always required. First for the page table and the second for the data.

6.7.1

w	x	y	z	order
5	2	2	4	Core1(2)→Core2(1)→Core3(4)→Core4(3)
3	2	2	4	Core1(2)→Core3→Core2(1)→Core4
3	2	2	2	Core1(2)→Core3(4)→Core4(3)→Core2(1)
5	2	2	2	Core1(2)→Core4→Core2(1)→Core3
1	2	2	4	Core3→Core1(2)→Core2(1)→Core4
1	2	2	2	Core3→Core1(2)→Core4→Core2(1)
1	2	2	0	Core3(4)→Core4(3)→Core1(2)→Core2(1)
5	2	2	0	Core4→Core1(2)→Core2(1)→Core3
3	2	2	0	Core4→Core1(2)→Core2(1)→Core3

6.7.2

We can set synchronization right after each operation. Hence, all of four cores see the same value of each variable.

6.9.1

Core1	Core2
A2, A3	B2, B4
A1, A4	B1, B4
A1	B1, B3
A1	

It'll take 4 cycles and 4 issue slots are wasted.

6.9.2

Core1	Core2
A2, A3	B2, B4
A1, A4	B1, B4
A1	B1, B3
A1	

It'll take 4 cycles and 4 issue slots are wasted.

6.9.3

FU1	FU2
A1	A2
A1	
A1	
B1	B4
B1	B4
A3	
A4	
B2	
B3	

It'll take 9 cycles and 6 issue slots are wasted.

6.9.4

FU1	FU2
A1	B1
A1	B1
A1	B2
A2	B3
A3	B4
A4	B4

It'll take 6 cycles and 0 issue slots are wasted.

Programming

Part 1

	dhrystone	median	multiply	qsort	rsort	towers	vvadd
Configuration1	557901	8863	44964	269251	900737	7497	11839
Configuration2	539076	8864	45012	257849	902477	7497	5053
Configuration3	542215	8849	45051	257292	911861	7577	4808
Configuration4	545514	8864	45111	254121	884849	7577	4653
Configuration5	527399	8864	45112	254384	885937	7577	4653
Configuration6	575048	8841	44942	269269	901048	7457	11790
Configuration7	582524	8841	44942	269315	900876	7436	11808
Configuration8	551829	9329	45080	274437	1026332	7485	12880
Configuration9	551705	9343	45108	274363	1026321	7485	12758
Configuration10	552317	9291	45073	274179	1026003	7499	13000
Configuration11	547000	9352	45143	274401	1031835	7501	12741
Configuration12	548128	9380	45191	263376	1051311	7581	5876
Configuration13	547676	9306	45179	263742	1050557	7618	5641

- (1) are different since the data cache of Config1 is one-way while that of Config2 is two-way. Besides, this benchmark requires a lot of data loading and storing. Therefore, Config2 has higher hit rate rather than Config1, so Config2 can be faster.
- (2) are different since the replacement policy of Config3 is random while that of Config4 is lru. The policy of lru will replace the one least used recently. Therefore, it comes out a better performance than random.
- (3) are different since the data cache of Config1 is one-way while that of Config3 is 4-way. This benchmark Though the Config with higher "way" may raise hit rate, it requires extra complex control logic, especially with 4-way. Hence, it becomes slower.
- (4) are different since the instruction cache of Config1 is one-way, that of Config6 is 2-way while that of Config7 is 4-way. This benchmark focuses on string handling, so the multiple ways of instruction cache may be less effective. Therefore, the overhead of extra control logic is higher.
- (5) are different since Config12 is 1-bank while Config13 is 4-bank. With serveral banks, it'll lower the access delay, so Config13 will be faster.
-
- Config17 on 1-core: 180192 cycles

```
root@80d3f3aa9eba:~/emulator# ./Config17 benchmarks/mt-matmul.riscv
This emulator compiled with JTAG Remote Bitbang client. To enable, use +jtag_rbb_enable=1.
Listening on port 35553

matmul(cid, nc, 16, input1_data, input2_data, results_data); barrier(nc): 180192 cycles, 43.9 cycles/iter, 6.5 CPI
```

  Config19 on 2-core: 92287 cycles

```
root@80d3f3aa9eba:~/emulator# ./Config19 benchmarks/mt-matmul.riscv
This emulator compiled with JTAG Remote Bitbang client. To enable, use +jtag_rbb_enable=1.
Listening on port 44139

matmul(cid, nc, 16, input1_data, input2_data, results_data); barrier(nc): 92287 cycles, 22.5 cycles/iter, 6.2 CPI
```

  Config20 on 4-core: 48239 cycles

```
root@80d3f3aa9eba:~/emulator# ./Config20 benchmarks/mt-matmul.riscv
This emulator compiled with JTAG Remote Bitbang client. To enable, use +jtag_rbb_enable=1.
Listening on port 44203

matmul(cid, nc, 16, input1_data, input2_data, results_data); barrier(nc): 48239 cycles, 11.7 cycles/iter, 6.5 CPI
```

- It doesn't decrease linearly. It's because with more cores, it has to handle the problem of synchronization. Therefore, it'll cause some overhead cost.

Part 2

```
D$ Bytes Read:            2554289
D$ Bytes Written:         86768
D$ Read Accesses:         594802
D$ Write Accesses:        22210
D$ Read Misses:           154608
D$ Write Misses:          6161
D$ Writebacks:            6183
D$ Miss Rate:             26.056%
I$ Bytes Read:            4838422
I$ Bytes Written:         0
I$ Read Accesses:         1465040
I$ Write Accesses:        0
I$ Read Misses:           93
I$ Write Misses:          0
I$ Writebacks:            0
I$ Miss Rate:             0.006%
```

I simply transpose the matrix B and assign to another array with 2 sets and 8 ways of Dcache. By doing so, it can decrease the miss rate of data cache to roughly 26%.

Bonus

- exploiting conditional branch misprediction: Before the result of the bounds check is known, the CPU speculatively executes code following the condition by predicting the most likely outcome of the comparison. There are many reasons why the result of a bounds check may not be immediately known, for example, a cache miss preceding or during the bounds check, congestion of a required execution unit, complex arithmetic dependencies, or nested speculative execution.

  eg: applying x = (address of a secret byte to read) – (base address of array1)

```
if (x < array1_size)
    y = array2[array1[x] * 4096];
```

- poisoning indirect branches: the adversary mistrains the branch predictor with malicious destinations, such that speculative execution continues at a location chosen by the adversary. The branch predictor is (mis-)trained in one context and applies the prediction in a different context. More specifically, the adversary can misdirect speculative execution to locations that would never occur during a legitimate program execution.

- 1. Preventing speculative execution
  2. Preventing access to secret data
  3. Preventing data from entering covert channels