計算機結構 HW5

B07902048 資工三 李宥霆

Handwritten

4.31

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	MEM	WB																				
jal ENT		IF	ID	EX	MEM	WB																			
bne x12, x13, TOP			IF	ID	EX	MEM	WB																		
slli x5, x12, 3				IF	ID	EX	MEM	WB																	
add x6, x10, x5					IF	ID	EX	MEM	WB																
ld x7, 0(x6)						IF	ID	EX	MEM	WB															
ld x29, 8(x6)							IF	ID	EX	MEM	WB														
sub x30, x7, x29									IF	ID	EX	MEM	WB												
add x31, x11, x5										IF	ID	EX	MEM	WB											
sd x30, 0(x31)											IF	ID	EX	MEM	WB										
addi x12,x12,2											IF	ID	EX	MEM	WB										
bne x12, x13, TOP												IF	ID	EX	MEM	WB									
slli x5, x12, 3													IF	ID	EX	MEM	WB								
add x6, x10, x5														IF	ID	EX	MEM	WB							
ld x7, 0(x6)															IF	ID	EX	MEM	WB						
ld x29, 8(x6)																IF	ID	EX	MEM	WB					
sub x30, x7, x29																		IF	ID	EX	MEM	WB			
add x31, x11, x5																			IF	ID	EX	MEM	WB		
sd x30, 0(x31)																				IF	ID	EX	MEM	WB	
addi x12,x12,2																				IF	ID	EX	MEM	WB	
bne x12, x13, TOP																					IF	ID	EX	MEM	WB

4.31.2

one issue: 10 cycles in loop
two issue: 9 cycles in loop
speedup: 10/9 = 1.11

4.31.3

```
    beqz x13, DONE
    li x12, 0
TOP:
    ld x7, 0(x10)
    ld x29, 8(x10)
    addi x12, x12, 2
    sub x30, x7, x29
    sd x30, 0(x11)
    addi x10, x10, 16
    addi x11, x11, 16
    bne x12, x13, TOP
DONE:
```

如果x13 == 0則直接跳過整個迴圈，另外改用pointer的方式access array

4.31.4

```
    beqz x13, DONE
    li x12, 0
TOP:
    ld x7, 0(x10)
    addi x12, x12, 2

    ld x29, 8(x10)

    sub x30, x7, x29

    sd x30, 0(x11)
    addi x10, x10, 16

    addi x11, x11, 16

    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
beqz x13, DONE	IF	ID	EX	MEM	WB													
li x12, 0		IF	ID	EX	MEM	WB												
ld x7, 0(x10)			IF	ID	EX	MEM	WB											
addi x12, x12, 2			IF	ID	EX	MEM	WB											
ld x29, 8(x10)				IF	ID	EX	MEM	WB										
sub x30, x7, x29					IF	ID	EX	MEM	WB									
sd x30, 0(x11)						IF	ID	EX	MEM	WB								
addi x10, x10, 16						IF	ID	EX	MEM	WB								
addi x11, x11, 16							IF	ID	EX	MEM	WB							
bne x12, x13, TOP								IF	ID	EX	MEM	WB						
ld x7, 0(x10)									IF	ID	EX	MEM	WB					
addi x12, x12, 2									IF	ID	EX	MEM	WB					
ld x29, 8(x10)										IF	ID	EX	MEM	WB				
sub x30, x7, x29											IF	ID	EX	MEM	WB			
sd x30, 0(x11)												IF	ID	EX	MEM	WB		
addi x10, x10, 16												IF	ID	EX	MEM	WB		
addi x11, x11, 16													IF	ID	EX	MEM	WB	
bne x12, x13, TOP														IF	ID	EX	MEM	WB

4.31.6

4.31.3: 8 cycles in loop
4.31.4: 6 cycles in loop
speedup: 8/6 = 1.33

4.31.7

```
    beqz x13, DONE
    li x12, 0
TOP:
    ld x5, 0(x10)
    ld x6, 8(x10)
    ld x7, 16(x10)
    ld x29, 24(x10)
    addi x12, x12, 4
    sub x30, x5, x6
    sub x31, x7, x29
    sd x30, 0(x11)
    sd x31, 16(x11)
    addi x10, x10, 32
    addi x11, x11, 32
    bne x12, x13, TOP
DONE:
```

4.31.8

```
    beqz x13, DONE
    li x12, 0
TOP:
    ld x5, 0(x10)
    addi x12, x12, 4

    ld x6, 8(x10)
    nop

    ld x7, 16(x10)
    nop

    ld x29, 24(x10)
    sub x30, x5, x6

    sd x30, 0(x11)
    sub x31, x7, x29

    sd x31, 16(x11)
    addi x11, x11, 32

    bne x12, x13, TOP
DONE:
```

4.31.9

4.31.7: 12 cycles in loop
4.31.8: 7 cycles in loop
speedup: 12/7 = 1.71

4.31.10

```
    beqz x13, DONE
    li x12, 0
TOP:
    ld x5, 0(x10)
    addi x12, x12, 4
    ld x6, 8(x10)
    nop
    ld x7, 16(x10)
    nop
    ld x29, 24(x10)
    sub x30, x5, x6
    sd x30, 0(x11)
    sub x31, x7, x29
    sd x31, 16(x11)
    addi x11, x11, 32
    bne x12, x13, TOP
DONE:
```

因為可以放的位置都已經放滿，剩下也無法rearrange，因此與4.31.8結果相同

5.5

5.5.1

offset有5個bit，1個bit指向一個byte，因此總共可以指向32個byte = 8個word

5.5.2

index有5個bit，所以cache總共有2^5=32個block

5.5.3

data storage bit: 32個block×32個byte = 1024byte = 8192 bit
valid bit: 32bit
tag bit: 54×32個block=1728 bit

$$\frac{8192+32+1728}{8192} = 1.21$$

5.5.4

Addr	binary addr	tag	index	offset	hit/miss	replace
0x00	00 00000 00000	00	00000	00000	miss	
0x04	00 00000 00100	00	00000	00100	hit	
0x10	00 00000 10000	00	00000	10000	hit	
0x84	00 00100 00100	00	00100	00100	miss	
0xE8	00 00111 01000	00	00111	01000	miss	
0xA0	00 00101 00000	00	00101	00000	miss	
0x400	01 00000 00000	01	00000	00000	miss	0x00-0x1F
0x1E	00 00000 11110	00	00000	11110	miss	0x400-0x41F
0x8c	00 00100 01100	00	00100	01100	hit	
0xC1C	11 00000 11100	11	00000	11100	miss	0x00-0x1F
0xB4	00 00101 10100	00	00101	10100	hit	
0x884	10 00100 00100	10	00100	00100	miss	0x80-0x9F

5.5.5

4/12 = **33%**

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

5.10.1

P1: 1/0.66ns = **1.515G Hz**
P2: 1/0.9ns = **1.11G Hz**

5.10.2

P1: 70/0.66 = 107個cycle, 1 + 107×0.08 = **9.56**個cycle
P2: 70/0.9 = 78個cycle, 1 + 78×0.06 = **5.68**個cycle

5.10.3

P1: CPI = 1 + 0.08×107(inst miss) + 0.08×0.36×107(data miss) = 12.64
12.64×0.66 = 8.34ns
P2: CPI = 1 + 0.06×78(inst miss) + 0.06×0.36×78(data miss) = 7.36
7.36×0.9=6.62ns
Hence, **P2 is faster**

5.10.4

5.62/0.66 = 9個cycle
1 + 0.08×9 + 0.08×0.95×107 = **9.85**個cycle
which is worse than 9.56個cycle

5.10.5

1 + 0.08×9(inst miss L1) + 0.08×0.95×107(inst miss L2) + 0.08×0.36×9(data miss L1) + 0.08×0.36×0.95×107(data miss L2) = 13.04

5.10.6

1 + 0.08×9 + 0.08×X×107 < 9.56
X < **0.916**

5.10.7

AMAT : 1 + 0.08×9 + 0.08X×107 = 1.72 + 8.56X
total CPI = AMAT+0.36×(AMAT-1)=1.9792 + 11.641X
每個instruction需要0.66×(1.9792 + 11.641X) = 1.3 + 7.68X
1.3 + 7.68X < 6.624
X < **0.693**

5.16

5.16.1

Address	virtual Page	TLB hit/miss	page hit/miss	page fault?	TLB state			
					valid	tag	physical page	last access
0x123D	1	miss	miss	yes	1	B	12	5
					1	7	4	2
					1	3	6	4
					1	1	13	0
0x08B3	0	miss	hit	No	1	0	5	0
					1	7	4	3
					1	3	6	5
					1	1	13	1
0x365C	3	hit	hit	No	1	0	5	1
					1	7	4	4
					1	3	6	0
					1	1	13	2
0x871B	8	miss	miss	Yes	1	0	5	2
					1	8	14	0
					1	3	6	1
					1	1	13	3
0xBEE6	B	miss	hit	No	1	0	5	3
					1	8	14	1
					1	3	6	2
					1	B	12	0
0x3140	3	hit	hit	No	1	0	5	4
					1	8	14	2
					1	3	6	0
					1	B	12	1
0xC049	C	miss	miss	Yes	1	C	15	0
					1	8	14	3
					1	3	6	1
					1	B	12	2

5.16.2

Address	virtual Page	TLB hit/miss	page hit/miss	page fault?	TLB state			
					valid	tag	physical page	last access
0x123D	0	miss	hit	No	1	B	12	5
					1	7	4	2
					1	3	6	4
					1	0	5	0
0x08B3	0	hit	hit	No	1	B	12	6
					1	7	4	3
					1	3	6	5
					1	0	5	0
0x365C	0	hit	hit	No	1	B	12	7
					1	7	4	4
					1	3	6	6
					1	0	5	0
0x871B	2	miss	miss	Yes	1	2	14	0
					1	7	4	5
					1	3	6	7
					1	0	5	1
0xBEE6	2	hit	hit	No	1	2	14	0
					1	7	4	6
					1	3	6	8
					1	0	5	2
0x3140	0	hit	hit	No	1	2	14	1
					1	7	4	7
					1	3	6	9
					1	0	5	0
0xC049	3	hit	hit	No	1	2	14	2
					1	7	4	8
					1	3	6	0
					1	0	5	1

用比較大的page size可以把附近的資料一起搬進來，連續資料page fault的次數降低，但每次搬資料的overhead變大，需要花更久時間把一個page搬進來。

5.16.3

Address	virtual Page	tag	index	TLB hit/miss	page hit/miss	page fault?	TLB state				
							valid	tag	physical page	last access	set index
0x123D	1	0	1	miss	miss	Yes	1	B	12	5	0
							1	7	4	2	
							1	3	6	4	1
							1	0	13	0	
0x08B3	0	0	0	miss	hit	No	1	0	5	0	0
							1	7	4	3	
							1	3	6	5	1
							1	0	13	1	
0x365C	3	1	1	miss	hit	No	1	0	5	1	0
							1	7	4	4	
							1	1	6	0	1
							1	0	13	2	
0x871B	8	4	0	miss	miss	Yes	1	0	5	2	0
							1	4	14	0	
							1	1	6	1	1
							1	0	13	3	
0xBEE6	B	5	1	miss	hit	No	1	0	5	3	0
							1	4	14	1	
							1	1	6	2	1
							1	5	12	0	
0x3140	3	1	1	hit	hit	No	1	0	5	4	0
							1	4	14	2	
							1	1	6	0	1
							1	5	12	1	
0xC049	C	6	0	miss	miss	Yes	1	6	15	0	0
							1	4	14	3	
							1	1	6	1	1
							1	5	12	2	

5.16.4

Address	virtual Page	tag	index	TLB hit/miss	page hit/miss	page fault?	TLB state			
							valid	tag	physical page	set index
0x123D	1	0	1	miss	miss	Yes	1	B	12	0
							1	0	13	1
							1	3	6	2
							0	4	9	3
0x08B3	0	0	0	miss	hit	No	1	0	5	0
							1	0	13	1
							1	3	6	2
							0	4	9	3
0x365C	3	0	3	miss	hit	No	1	0	5	0
							1	0	13	1
							1	3	6	2
							1	0	6	3
0x871B	8	2	0	miss	miss	Yes	1	2	14	0
							1	0	13	1
							1	3	6	2
							1	0	6	3
0xBEE6	B	2	3	miss	hit	No	1	2	14	0
							1	0	13	1
							1	3	6	2
							1	2	12	3
0x3140	3	0	3	miss	hit	No	1	2	14	0
							1	0	13	1
							1	3	6	2
							1	0	6	3
0xC049	C	3	0	miss	miss	yes	1	3	15	0
							1	0	13	1
							1	3	6	2
							1	0	6	3

5.16.5

如果沒有TLB的話，則每次做virtual-physical address的mapping都需要到page table去找資料，而page table又放在RAM裡面，會導致translate的速度變得很慢，需要花很多個cycle來處理

6.7

6.7.1

x	y	w	z
2	2	1	0
2	2	3	0
2	2	5	0
2	2	1	2
2	2	3	2
2	2	5	2
2	2	1	4
2	2	3	4
3	2	5	4

6.7.2

Set synchronization instructions after each operation so that all cores see the same value on all variable

6.9

6.9.1

core 1	core 2
A3	B2, B4
A1, A4	B1, B4
A1, A2	B1, B3
A1	

4 cycles, 4 slot waste

6.9.2

core 1	core 2
A3	B2, B4
A1, A4	B1, B4
A1, A2	B1, B3
A1	

4 cycles, 4 slot waste

6.9.3

thread	unit 1	unit 2
X	A1	
X	A1	
X	A1	A2
X	A3	
X	A4	
Y	B1	
Y	B1	
Y	B2	
Y	B3	B4
Y	B4	

10 cycles, 8 slot waste

6.9.4

unit 1	unit 2
A1	B1
A1	B1
A1	B2
A2	B3
A3	B4
A4	B4

6 cycles, no slot waste

Programming

Part 1

Run test .sh and fill in cycle counts for each benchmark and each setting in the following form

	dhrystone	median	multiply	qsort	rsort	towers	vvadd
Configuration1	557936	8863	44964	269251	900737	7497	11830
Configuration2	539075	8817	44947	257841	902477	7497	5053
Configuration3	542214	8881	45032	257034	911861	7577	4808
Configuration4	545513	8864	45111	254099	884849	7577	4653
Configuration5	527386	8864	45112	254384	885937	7577	4653
Configuration6	574790	8789	44900	269251	901048	7457	11830
Configuration7	582962	8789	44892	269342	900876	7476	11808
Configuration8	551369	9337	45091	274111	1025081	7485	12795
Configuration9	551704	9315	45096	274363	1026321	7485	12872
Configuration10	552352	9292	45101	274172	1026003	7499	13006
Configuration11	546999	9390	45127	275235	1031835	7501	12648
Configuration12	549202	9330	45112	263335	1051311	7606	5476
Configuration13	547675	9361	45244	263814	1051300	7599	5541

- Why are (1) the same or different?
  - Config2 have 2-way Dcache, which means vector data has less chances to be moved out of cache, cache miss is less.
- Why are (2) the same or different?
  - Replacing algorithm are different. Lru is better in performance since it can choose better place to replace than random. However, random and lru does not have too much different on performance since there is only 4 way on L1 Dcache, influence of using LRU is limited.
- Why are (3) the same or different?
  - Using 4 way means we have less offset in cache, which is easier to have conflict and replacement in L1 Dcache. But using random replacement will lead to more wrong replacement in small number of ways case.

- Why are (4) the same or different?
    - Icache way are different. The cycle will increase as more ways in I-cache because of using random replacement. As number of sets goes down, more collision happens, but number of wrong replacement increase due to random replacement.
- Why are (5) the same or different?
    - The increasing of number of bank in L2 cache can decrease a little cycle count. More bank means more data we can get in a period of cycle.
- pmp
    - trying to set physical memory protection by reading and writing on control and status register.
- Change the number of cores available in crt.S file
    - Report the cycle count of configuration
        - 1-core: 180192 cycles
        - 2-core: 92287 cycles
        - 4-core: 48239 cycles
    - The cycle count decreases non-linearly; however, core num is inversely proportional to cycle counts because with more thread computing matrix, it can divide its job to several cores in order to get faster speed. Nontheless, some operations still can’t be divide equally, thus the multiplication of core num and cycle counts may not be equal.

Part 2

- Report on how you make your matrix multiplication and maybe some cache miss rate statistics using spike
    - 我用block size為4的大小來做blocking，總共會需要6個迴圈，每個block之中用cache friendly的access方法，8-way的L1_Dcache, 2-way的L1_Icache, lru replacement，所計算出來的cycle數為2693409個cycle

Bonus

How to perform “exploiting conditional branch misprediction” attack?

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

以paper上的例子來說，**假設：**

- array1_size跟array2不在cache裡
- array1[x]在cache裡
- x >= array1_size(想要存取非法記憶體位置)

- branch prediction是true
  則以下事情會依序發生

1. array1_size要stall很久，所以branch pridiction先假設是true，執行if裡面的statement
2. 因為array1[x]在cache裡面，所以可以馬上算出array1[x]*4096的結果
3. 因為array2[array1[x]*4096]不在cache裡，所以也讓cache去跟RAM要資料
4. array1_size從Ram拿到了，發現猜錯，於是要rewind，但這時array2[array1[x]*4096]這筆資料可能已經從RAM搬進cache了
5. 一段一段的要求array2的資料，發現某段資料要的特別快(已經被搬進cache裡面，這段資料就是之前mispredict搬進來的那段)，就可以藉此反推出array1[x]的數值，成功獲取非法位置的資訊。

How to perform “poisoning indirect branches” attack?

1. 在context A訓練branch predictor在predict的時候每次都跳到某個固定的case裡面
2. 讓該branch predictor在context B做出錯誤的prediction，進入spectre gadget的function，spectre gadget簡單來說就是利用兩個register去access非法的位置，利用類似上題的方法來逆取得非法位置的值

How to mitigate Spectre Attacks?

1. use serializing or speculation blocking instructions that ensure that instructions following them are not executed speculatively
2. use static analysis to reduce the number of speculation blocking instructions required
3. replaces array bounds checking with index masking