HW5 b05202068 劉耿宇

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 26 27 28
li   x12, 0           IF ID EX ME WB
jal  ENT              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
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
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
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
slli x5,x12,3                                                        IF .. ID .. EX ME WB
```

②. 1-issue machine:

```
     li   x12, 0
     jal  ENT
TOP: slli x5, x12, 3
     add  x6, x10, x5
     ld   x7, 0(x6)
     ld   x29, 8(x6)
     nop
     sub  x30, x7, x29
      ⋮
(same)
```

} 10 cycles/loop.

2-issue machine

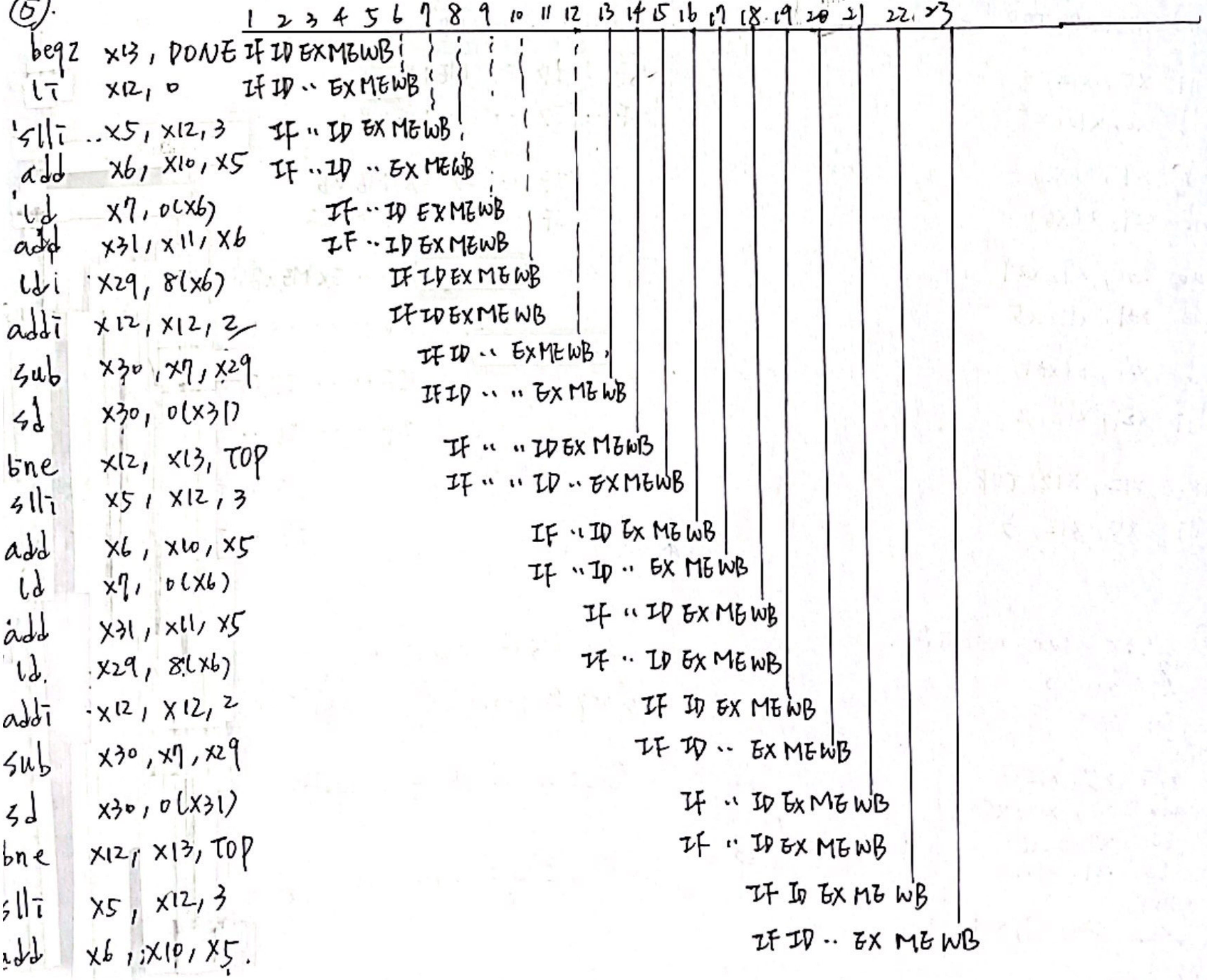
loop 1 starts from slli at cycle 2.

loop 3 start in cycle 22.

$\therefore \frac{22-2}{2} = 10$ cycles/loop #

③
```
    beqz  x13, DONE
    li    x12, 0
    jal   ENT
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(x31)
ENT:
    bne   x12, x13, TOP
DONE:
```
→ 9 cycles/loop.

④
```
    beqz  x13, DONE
    li    x12, 0
TOP:
    slli  x5, x12, 3      } i=[i<<3]
    add   x6, x10, x5     } a+i
    ld    x7, 0(x6)       }
    add   x31, x11, x5    } b+i
    ld    x29, 8(x6)      }
    addi  x12, x12, 2     }
    sub   x30, x7, x29    }
    sd    x30, 0(x31)     }
    bne   x12, x13, TOP
DONE:
```

⑤

```
                       1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
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, x6             IF .. ID EX ME WB
ldi   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
slli  x5, x12, 3                                                       IF ID EX ME WB
add   x6, x10, x5.                                                     IF ID .. EX ME WB
```

⑥. from ③ → 9 cycles/loop ; from ⑤ → 7.5 cycles/loop.

∴ $\frac{9}{7.5} = \boxed{1.2}$ speedup.

⑦.
```
    beqz  x13, DONE
    li    x12, 0
TOP:
    slli  x5, x12, 3
    add   x6, x10, x5
    add   x31, x11, x5
    ld    x7, 0(x6)
    ld    x29, 8(x6)
    ld    x5, 16(x6)
    ld    x15, 24(x6)        } → 13 cycles.
    addi  x12, x12, 4
    sub   x30, x7, x29
    sub   x14, x5, x15
    sd    x30, 0(x31)
    sd    x14, 16(x31)
    bne   x12, x13, TOP
DONE:
```

⑧
```
    beqz  x13, DONE
    li    x12, 0
    addi  x6, x10, 0
TOP
    ld    x7, 0(x6)      }
    add   x31, x11, x5
    ld    x29, 8(x6)     }
    slli  x5, x12, 3
    ld    x15, 24(x6)    }
    sub   x30, x7, x29
    sd    x30, 0(x31)    }
    sub   x14, x16, x15
    sd    x14, 16(x31)   }
    add   x6, x10, x5
    bne   x12, x13, TOP
DONE:
```

⑨ 1-issue processor : 13 cycles/unrolled iteration = 6.5 cycles/loop.

2-issue processor : 7.5 cycles/unrolled iteration = 3.75 cycles/loop.

∴ $\frac{6.5}{3.75}$ = 1.73 speedup

⑩ Using code in ⑧, no further improvement (∵ no stalls due to structural hazards)

5.5

①. 4 (8-byte) words / cache block.

∵ There are 5 bits in offset : $\underbrace{\boxed{1\,|\,1}}_{\substack{\uparrow \\ \text{block} \\ \text{offset}}}\underbrace{\boxed{1\,|\,1\,|\,\;}}_{\substack{\uparrow \\ \text{the offset into an 8-byte word}}}$ ∴ 2^2 words/block

② $\overset{9 \sim 5}{\boxed{\qquad\qquad}}$ → 5 index bits → $2^5 = 32$ blocks (cache lines) ※

③.

The cache stores : 32 lines × 4 words/block 8 bytes/word × 8 bits/bytes

= 8192 bits

With data , 54 tag bits + 1 valid bits → $8192 + (54+1) \times \underset{\text{lines.}}{\underset{\sim}{32}} = 9952$ bits

∴ The ratio = $\frac{9952}{8192} \approx 1.2148$ ※

④

byte address	binary addr.	Tag	index	offset	Hit/Miss	bytes replaced
0X00	0000 0000 0000	0X0	0X00	0X00	M.	
0X04	0000 0000 0100	0X0	0X00	0X04	H.	
0X10	0000 0001 0000	0X0	0X00	0X10	H	
0X84	0000 1000 0100	0X0	0X04	0X04	M	
0Xe8	0000 1110 1000	0X0	0X07	0X08	M	
0Xa0	0000 1010 0000	0X0	0X05	0X00	M	
0X400	0100 0000 0000	0X1	0X00	0X00	M .	0X00-0X1F
0X1e	0000 0001 1110	0X0	0X00	0X1e	M	0X400-0X41F
0X8c	0000 1000 1100	0X0	0X04	0X0c	H	
0Xc1c	1100 0001 1100	0X3.	0X00	0X1c	M	0X00-0X1F
0Xb4	0000 1011 0100	0X0	0X05	0X14	H	
0X884	1000 1000 0100	0X2	0X04	0X04	M	0X80 - 0X9f.

⑤ Hit ratio : $\frac{4}{12} = 33.3\%$

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 Cycle time = L1 hit time
clock rate = 1/(clock time)

∴ P1 : 1/0.66ns = 1.51 GHz
P2 : 1/0.90ns = 1.11 GHz

② ∵ AMAT = (time for a hit + miss rate × miss penalty) $\leftarrow \frac{\text{memory access time}}{\text{L1 Hit time}}$
and each instruction requires at least 1 cycle.

P1:
→ AMAT : $1 + 0.08 \times \left\lceil \frac{70}{0.66} \right\rceil = 9.56$ cycles. (or 6.36 ns)

P2:
AMAT : $1 + 0.06 \times \underbrace{\left\lceil \frac{70}{0.90} \right\rceil}_{\uparrow \text{ can't divide cycles}} = 5.68$ cycles (5.1 ns) #

③
P1 : 12.64 CPI ; 8.34 ns/inst
P2 : 7.36 CPI ; 6.63 ns/inst # → P2 is faster.

∵ every inst. takes at least one cycle.

P1 - $1 + 0.08 \times 107 + 0.36 \times 0.08 \times 107 = 12.64$ CPI $\xrightarrow{0.66\text{ns/cycle}}$ 8.34 ns/inst

P2 : $1 + 0.08 \times 78 + 0.36 \times 0.08 \times 78 = 7.36$ CPI $\xrightarrow{0.90\text{ns/cycle}}$ 6.63 ns/inst

④ L2 access takes 9 cycles ($\left\lceil \frac{5.62}{0.66} \right\rceil$).

∴ $1 + \underbrace{0.08}_{\text{miss rate}} (9 + \underbrace{0.95}_{\text{L2 miss}} \times \underbrace{107}_{\text{memory lookup}}) = 9.85$ cycles, worse #

⑤

The CPI for P1 with an L2 cache = $9.85 + 0.36 \times 8.85 = 13.04$ #

(AMAT + memory percentage×(AMAT−1))

⑥ AMAT with L2 < AMAT with only L1

$1 + 0.08 \times (9 + x \times 107) < 9.56 \Rightarrow x < 0.916$ ※

⑦ P1 AMAT with L2 < P2 AMAT (6.63ns)

$\Rightarrow$

$CPI_{P1} \times 0.66 < 6.63 \rightarrow CPI_{P_1} < 10.05$

but $CPI_{P_1} = AMAT_{P1} + 0.36(AMAT_{P1} - 1) < 10.05$

$\Rightarrow AMAT_{P1} < 7.65$

$\quad\quad \parallel$

$1 + 0.08(9 + x \times 107)$

$\therefore x < 0.693$.

The miss rate is at most 69.3%.

5.16.

①. Addr.	Virtual Page	TLB H/M.	TLB. valid	tag	physical page
0x123d	1	TLB miss PT hit/PF	1	b	12
			1	7	4
			1	3	6
			1 (last access 0)	1	13
0x08b3	0	TLB miss PT hit	1 (last 1)	0	5
			1	7	4
			1	3	6
			1 (last 0)	1	13
0x365c	3	TLB miss PT hit	1 (last 1)	0	5
			1	7	4
			1 (last 2)	3	6
			1 (last 0)	1	13.
0x871b	8	TLB miss PT hit/PF	1 (last 1)	0	5
			1 (last 3)	8	14
			1 (last 2)	3	6
			1 (last 0)	1	13
0xbee6	b	TLB miss PT hit	1 (last 1)	0	5
			1 (last 3)	8	14
			1 (last 2)	3	6
			1 (last 4)	b	12.
0x3140	3	TLB miss PT hit	1 (last 1)	0	5
			1 (last 3)	8	14
			1 (last 5)	3	6
			1 (last 4)	b	12.

0xc040	c	TLB miss PT hit PF	1 (last 6)	c	15
			1 (last 3)	8	14
			1 (last 5)	3	6
			1 (last 4)	b	12.

② Address	Virtual Page	TLB H/M	Valid	Tag	Physical Page
0x123d	1	TLB miss PT hit	1	11	12
			1	7	4
			1	3	6
			1 (last 0)	0	5
0x08b3	0	TLB hit	1	11	12
			1	7	4
			1	3	6
			1 (last 1)	0	5
0x365c	0	TLB hit/PT hit	1	11	12
			1	7	4
			1	3	6
			1 (last 2)	0	5
0x871b	2	TLB miss/PT hit PF	1 (last 3)	2	13
			1	7	4
			1	3	6
			2	0	5
0xbeeb	2	TLB hit/PT hit	1 (last 4)	2	13
			1	7	4
			1	3	6
			1 (last 2)	0	5
0x3140	0	TLB hit/PT hit	1 (last 4)	2	13
			1	7	4
			1	3	6
			5	0	5
0xc040	3	TLB hit/ PT hit.	1 (last 4)	2	13
			1	7	4
			1 (last 6)	3	6
			1 (last 5)	0	5

Scanned with CamScanner

A larger page size → TLB miss ↓

but fragmentation can be serious and the usage of physical memory is not spatially ^efficient .

③ 2-way associative

Address	Virtual Page	Tag	Index	TLB H/M	Valid	Tag	Phy. Page	Index
0x123d	1	0	1	TLB miss PT hit/PF	1	6	12	1
					1	7	4	0
					1	3	6	1
					1 (last 0)	0	13	0
0x08b3	0	0	0	TLB miss/ PT hit	1 (last 1)	0	5	0
					1	7	4	1
					1	3	6	0
					1 (last 0)	0	13	1
0x365c	3	1	1	TLB miss/PT hit	1 (last 1)	0	5	0
					1 (last 2)	1	6	1
					1	3	6	0
					1 (last 0)	1	13	1
0x871b	8	4	0	TLB miss/PT hit PF	1 (last 1)	0	5	0
					1 (last 2)	1	6	1
					1 (last 3)	4	14	0
					1 (last 0)	1	13	1
0xbeeb	b	5	1	TLB miss/PT hit	1 (last 1)	0	5	0
					1 (last 2)	1	6	1
					1 (last 3)	4	14	0
					1 (last 4)	5	12	1
0x3140	3	1	1	TLB hit / PT hit	1 (last 1)	0	5	0
					1 (last 5)	1	6	1
					1 (last 3)	4	14	0
					1 (last 4)	5	12	1

0X c049	c	6	0	TLB miss	1 (last 6) 6	15	0
				PT miss	1 (last 5) 1	6	1
				PF	1 (last 3) 4	14	0
					1 (last 4) 5	12	1.

④ Direct mapping.

					TLB			
Address	Virtual Page	Tag	Index	TLB H/M	Valid	Tag	Physical page	Index
0x123d	1	0	1	TLB miss PT hit/PF	1	6	12	0
					1	0	13	1
					1	3	6	2
					0	4	9	3
0x08b3	0	0	0	TLB miss PT hit	1	0	5	0
					1	0	13	1
					1	3	6	2
					0	4	9	3
0x365c	3	0	3	TLB miss PT hit	1	0	5	0
					1	0	13	1
					1	3	6	2
					1	0	6	3
0x871b	8	2	0	TLB miss PT hit/PF	1	2	14	0
					1	0	13	1
					1	3	6.	2
					1	0	6	3.
0xbeeb	b	2	3	TLB miss PT hit	1	2	14	0
					1	0	13	1
					1	3	6	2
					1	2	12	3
0x3140	3	0	3	TLB hit PT hit	1	2	14	0
					1	0	13	1
					1	3	6	2
					1	0	6	3.

0xc049	c	7	0	TLB miss	1	7	15	0
				PT miss/PF	1	0	13	1
					1	3	6	2
					1	0	6	3.

⑤ If there were no TLB, almost every memory access will takes two accesses to get the data: one to page table, followed by an access to the requested data in RAM ※

6.7 ① For four processors in SMP.

x	y	w	z	
2	2	1	0	(core 3 : init w to 1 ; core 4 : do nothing yet)
2	2	3	0	(core 3 : add x to w)
2	2	5	0	(core 3 : add y to w)
2	2	1	2	core 4 : add x to z
2	2	3	2	
2	2	5	2	
2	2	1	4	core 4 : add y to z.
2	2	3	4	
2	2	5	4.	

② Use synchronization on instructions (mutex) after each operation s.t. all cores see the same values on all variables. ※

6.9

① cycle	Core 1 on Thread X	Core 2 on Thread Y.
1	A3, ⬚	B1, B4
2	A1, A2	B1, B4
3	A1, A4	B2, ⬚
4	A1, ⬚	B3, ⬚

4 cycles to execute, 4 issue slots are wasted ※

② Same as ①

③

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

9 cycles, 6 issue slots are wasted ※

④

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

6 cycles, no slots wasted. ※

Computer Architecture – Homework 5

Tony G. Liu b05202068

January 18, 2021

1 Handwritten

Shown above.

2 Programming

2.1 Observing cache behavior

The cycle counts for different confiurations over different workloads.

	dhrystone	median	multiply	qsort	rsort	towers	vvadd
config 1	557936(4)	8863	44964	269251	900737(3)	7497	11830(1)
config 2	539075	8817	44947	257841	902477	7497	5053(1)
config 3	542214	8881	45032	257034	911861(2),(3)	7577	4808
config 4	545513	8864	45111	254099	884849(2)	7577	4653
config 5	527386	8864	45112	254384	885937	7577	4653
config 6	574790(4)	8789	44900	269251	901048	7457	11830
config 7	582962(4)	8789	44892	269342	900876	7476	11808
config 8	551369	9337	45091	274111	1025081	7485	12795
config 9	551704	9315	45096	274363	1026321	7485	12872
config 10	552352	9292	45101	274172	1026003	7499	13006
config 11	546999	9390	45127	275235	1031835	7501	12648
config 12	549202(5)	9330	45112	263335	1051311	7606	5476
config 13	547674(5)	9361	45244	263814	1051300	7599	5541

(1): Config 1 uses direct mapping in Dcache, while config 2 is 2-way associative. The latter has higher hit rate compared to the former. And therefore the cycle counts reduce to half of that of the former.

(2): Config 3 uses random replacement policy, while config 4 uses `LRU`. Both of them use the same 4-way mapping, and since random replacement is an approximate for `LRU`, their cycle counts are closed.(But random policy could cause higher miss rate)

(3): Config 1 is direct mapping in Dcache, while config 3 is 4-way associative. The cycle count is slightly higher in config 3 because 4-way mapping requires more searching time of blocks within a set. The loop in the workload `rsort` is not too long, and hence the miss rate could be low compared to `vvadd`. Therefore, the cycle count won't decrease to 1/4 of that in config 1.

(4): The difference betwenn config 1, 6, 7 is their L1 Icache mapping function, which is 1-, 2-, 4-way mapping. Icache stores instuctions for PC to fetch, normally if there are not too many branches, Icache won't give too much speedup. On the other hand, the cycle count is higher in config 7 since the searching time within a set is longer.

(5): Config 12 and 13 are different in L2 cache bank. The L2 cache supports both L1 Dcach and Icache miss. Hence the one with more banks means the bandwidth of the cache and the number of parallel access increase. Hence, the cycle count in config 13 is lower.

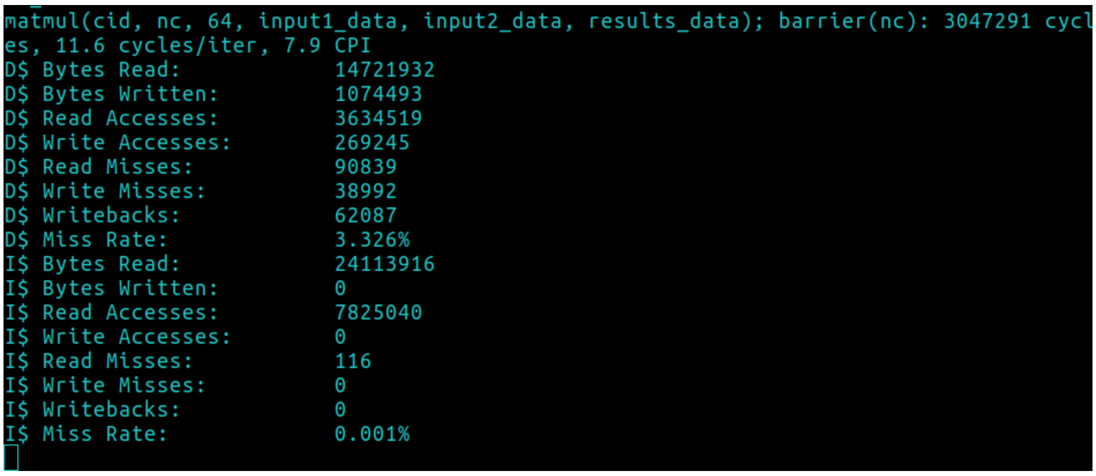
The `pmp.c` deals with the paging in virtual memory.

The `mt-matmul`: The cycle counts decrease linearly, since as the cores number increases, the speedup will also increase linearly.

	cycle counts	cycles/iter	CPI
1 core	180192	43.9	6.5
2 cores	92287	22.5	6.2
4 cores	48239	11.7	6.5

2.2 Cache and matrix multiplication

I use blocking to reduce cache miss rate. By blocking, I could reuse the data in cache to compute the matrix elements that appear in the reuse pattern. On top of that, loop unrolling is also used. The miss rate did decrease from 52.258% to 3.326%. And the cycle count drops from the original 3544288 to 3047291.



```
matmul(cid, nc, 64, input1_data, input2_data, results_data); barrier(nc): 3047291 cycl
es, 11.6 cycles/iter, 7.9 CPI
D$ Bytes Read:              14721932
D$ Bytes Written:           1074493
D$ Read Accesses:           3634519
D$ Write Accesses:          269245
D$ Read Misses:             90839
D$ Write Misses:            38992
D$ Writebacks:              62087
D$ Miss Rate:               3.326%
I$ Bytes Read:              24113916
I$ Bytes Written:           0
I$ Read Accesses:           7825040
I$ Write Accesses:          0
I$ Read Misses:             116
I$ Write Misses:            0
I$ Writebacks:              0
I$ Miss Rate:               0.001%
```

2.3 Architecture and Security

1. Exploiting conditional branch misprediction: let attacker to read arbitrary memory from another process.

Consider the following code:

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

```

The bound check may leads to incorrect prediction. An attacker might exploit this by choosing an out-of-bounds `x` s.t. `array1[x]` is a secret byte k in victim's memory, with `array1_size` and `array2` uncached. Since `x` in the previous operations were valid, the branch predictor will take the branch. So speculative execution continues and uses k to get the addres `array2[`k`*4096]`, and sends a request to read the address. Attacker can then know the value of `array2[`k `*4096]` by measuring the response time in cache for the address.

2. Poisoning indirect branches: The adversary could mistrains the branch predictor with malicious destination branch s.t. speculative execution continues at a location chosen by the adversary.

3. Mitigate Spectre Attacks:

- Serializing(*Speculation blocking* instruction): Encure instructions within the block are not executed speculatively

- Replacing array bounds checking with index masking: Apply a bit mask to the index, which can limit the distance of the bounds violation, preventing access to secret data.
- *Retpolines* : A code sequence that replaces indirect branches with return instructions. This can prevent branch poisoning.