# Visualizing the out-of-order CPU model

Ryota Shioya Nagoya University

### Introduction

### This presentation introduces

### $\bigcirc$ the visualization of the out-of-order CPU model in gem5

🐵 Konata	- 0 ×
File Window View Help	
trace (2).out 🛪	
1390: 1390 (T0: R252): 0x0042	650: sub x1, x1, #3 F 1 2 Dc Rn 1 Is Cm 1
1391: 1391 (T0: R253): 0x0042	6554: subs x1, #30 F 1 2 DC Rn 1 DS IS Cm 1
1392: 1392 (T0: R254): 0x0042	658: b.hi 0x425678 F 1 2 Dc Rn 1 Ds 1 Is Cm 1
1393: 1393 (T0: R255): 0x0042	i65c: subs w1, #30 F 1 2 Dc Rn 1 Ds Is Cm 1 2
1394: 1394 (T0: R256): 0x0042	6660: b.hi 0x425678 F 1 2 Dc Rn 1 Ds 1 Is Cm 1
1395: 1395 (T0: R257): 0x0042	664: ldrb w1, [w3, w1, UXTW] F 1 2 Dc Rn 1 Ds Is Cm 1 2 3 4 5 6
1396: 1396 (T0: R258): 0x0042	668: adr x26, #12 F 1 2 Dc Rn 1 Is Cm 1 2 3 4 5 6 7
1397: 1397 (T0: R259): 0x0042	i66c: add x1, x26, x1, LSL #2 F 1 2 Dc Rn 1 Ds 1 2 3 Is Cm 1 2 3
1398: 1398 (T0: R260): 0x0042	
1399: 1399 (T0: R261): 0x0042	678: ldr x1, [x0, #16]! F 1 2 Dc Rn 1 Is Cm 1 2 3 4 5 6 7 8 9 10 11 12 13 14
1400: 1400 (T0: R262): 0x0042	i678: addxi_uop x0, x0, #16 F 1 2 Dc Rn 1 Is Cm 1 2 3 4 5 6 7 8 9 10 11 12 13 14
1401: 1401 (T0: R263): 0x0042	i67c: cbnz x1, 0x425650 F 1 2 Dc Rn 1 Ds 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
1402: 1402 (T0: R-1): 0x00425	
1403: 1403 (T0: R-1): 0x00425	584: str x7, [x15, #4048] F Dc Rn 1 Is
1404: 1404 (T0: R-1): 0x00425	588: str x25, [x14, #2560] F Dc Rn 1 Is
1405: 1405 (T0: R-1): 0x00425	38c: subs w2, #15 F Dc Rn 1 Is
1406: 1406 (T0: R-1): 0x00425	i90: str x24, [x13, #2536] F 1 Dc Rn 1 Is
1407: 1407 (T0: R-1): 0x00425	i94: str x23, [x12, #2592] F 1 Dc Rn 1 Is
1408: 1408 (T0: R-1): 0x00425	598: str x22, [x5, #3536] F 1 Dc Rn 1 Is
1409: 1409 (T0: R-1): 0x00425	i9c: str x21, [x0, #8] F 1 Dc Rn 1 Is
1410: 1410 (T0: R-1): 0x00425	ia0: str x20, [x0, #16] F Dc Rn 1 Ds Is
1411: 1411 (T0: R-1): 0x00425	ia4: str x19, [x11, #2568] F DC Rn 1 Ds Is
1412: 1412 (T0: R-1): 0x00425	ia8: str x18, [x10, #768] F DC Rn 1 Ds Is
1413: 1413 (T0: R-1): 0x00425	ac: str x17, [x9, #2640] F Dc Rn 1 Ds Is
1414: 1414 (T0: R-1): 0x00425	5b0: str x16, [x8, #624] F 1 Dc Rn 1 Ds Is
1415: 1415 (T0: R-1): 0x00425	

### Introduction

### Let's suppose

- $\diamondsuit$  you come up with an excellent idea and
- $\diamond$  try to extend the CPU model in gem5 for adding your new method.
- You will probably tackle the following issues:
  - ◇ difficult bugs, especially performance related ones
  - a situation where your method cannot improve the performance as expected

- You probably investigate your modified gem5 as follows:
  - Check counters outputted by gem5
    - □ e.g. the number of LLC misses / branch mispredictions
    - □ These counters sometimes give us clues.
  - $\diamond$  Check the behavior by using a debugger and step execution
  - But, it is difficult to fix issues in the following situations:
    - $\diamond$  You have no idea what causes it
    - You recognize some counters show that something is wrong, but you have no idea what happened

## Visualizing the pipeline behavior

- In such situations, pipeline visualization is very useful.
- In general, visualization is a powerful tool for investigating bugs or behavior.
  - If you have developed hardware with HDL such as Verilog, you may have used a waveform viewer.
    - In a waveform view, you can easily see signal transitions and relations between signals.
    - □ Such viewers may have helped you a lot.
  - This is also true for gem5!

## A text-based pipeline viewer is provided for gem5

This viewer is very useful to investigate the pipeline behavior.

- $\diamond$  But, you can see only a limited range of instruction sequences at once
- $\diamond$  This is the "less" command itself, it is not very user-friendly.

Thanks for flying Vim	— less — 162×44
[ <mark>dn.]</mark> c <sup>c</sup> .	( 480000) 0x120007bf4.0 ldg r2,0(r15) [ 328]
[dn.p., ]cn.	480000)0x120007bf8.0 cmpeg r2,5,r1 [ 329]
[ <mark>dn.p]e.r</mark>	( 480000)0x120007bfc.0 bne r1.0x120007c4c [ 330]
[ <b>T</b> r <b>dn.] cn</b>	520000)0x120007c00.0 cmple r2,5,r1 [ 331]
[rdn.plic.n	520000)0x120007c04.0 beg r1,0x120007c14 [ 332]
[ <mark>rdn.]zn</mark>	520000)0x120007c08.0 cmpeq r2,3,r1 [ 333]
[ <mark>fdn.p][e.r</mark>	(520000)0x120007c0c.0 bne r1,0x120007c40 [334]
[	(520000)0x120007c40.0 ldqr1,8(r16) [349]
[f <mark>dn.p]c.n</mark>	(520000) 0x120007c44.0 stq r1,0(r4) [ 350]
	(520000)0x120007c48.0 br 0x120007c54 [351]
	( 520000)0x120007c54.0 lda r16,16(r16) [ 358]
fdn.plep.	(520000)0x120007c58.0 ldqr1,0(r16) [359]
	(520000)0x120007c5c.0 bne r1,0x120007bf4 [360]
f <mark>dn.j</mark> cn	(520000)0x120007bf4.0 ldq r2,0(r16) [ 377]
f <mark>dn.p]</mark>	(520000)0x120007bfc.0 bne r1,0x120007c4c [379]
[dn.]cn	
f <mark>fdn.p]c.n</mark>	(520000)0x120007c50.0 stq r1,0(r3) [ 400]
<mark>r</mark> d <b>n.lc<mark>r</mark></b>	( 520000) 0x120007c54.0 lda r16,16(r16) [ 401]
<mark>fdn.p∥c</mark> p	(520000)0x120007c58.0 ldqr1,0(r16) [ 402]
	( 520000)0x120007c5c.0 bne r1,0x120007bf4 [ 403]
<mark>fdn. </mark> c <mark>r</mark> .	
fdn.pje.n	(520000)0x120007bf8.0 cmpeq r2,5,r1 [ 405]
	(520000)0x120007bfc.0 bne r1,0x120007c4c [ 406]
	520000)0x120007c00.0 cmple r2,5,r1 [ 407]
	( 520000)0x120007c04.0 beq r1,0x120007c14 [ 408]
	(520000)0x120007c14.0 cmpeq r2,6,r1 [ 409]
	(520000)0x120007c18.0 bne r1,0x120007c28 [ 410]
	(520000)0x120007c1c.0 cmpeq r2,17,r1 [ 426]
r <mark>dn.plic.n</mark>	( 520000)0x120007c20.0 bne r1,0x120007c34 [ 427]
	(520000)0x120007c24.0 br 0x120007c54 [443]
-lc	( 520000) 0x120007c54.0 lda r16,16(r16) [ 444]
.plcp	( 520000) 0x120007c58.0 ldg r1,0(r16) [ 445]
-pBen	520000)0x120007c5c.0 bne r1,0x120007bf4 [ 446]
	560000)0x120007bf4.0 ldg r2,0(r16) [ 463]
	(560000)0x120007bf8.0 cmpeq r2,5,r1 [ 464]
[dn.]c.n.	560000)0x120007c00.0 cmple r2.5.r1 [ 478]
rdn. pile. P	560000) 0x120007c04.0 beg r1,0x120007c14 [ 479]
Idn. Je.e.	
fdn.pjc.c	

### Konata: a new GUI based viewer

You can see the pipeline behavior as a map app.

 $\diamond$  This presentation introduces Konata and best practice in gem5.

le Konata	- 0 X
File Window View Help	
trace (2).out 🕱	
1390: 1390 (T0: R252): 0x00425650:	sub x1, x1, #3 F 1 2 Dc Rn 1 Is Cm 1
1391: 1391 (T0: R253): 0x00425654:	subs x1, #30 F 1 2 Dc Rn 1 Ds Is Cm 1
1392: 1392 (T0: R254): 0x00425658:	b.hi 0x425678 F 1 2 Dc Rn 1 Ds 1 Is Cm 1
1393: 1393 (T0: R255): 0x0042565c:	subs w1, #30 F 1 2 Dc Rn 1 Ds Is Cm 1 2
1394: 1394 (T0: R256): 0x00425660:	b.hi 0x425678 F 1 2 Dc Rn 1 Ds 1 Is Cm 1
1395: 1395 (T0: R257): 0x00425664:	ldrb w1, [w3, w1, UXTW] F 1 2 Dc Rn 1 Ds Is Cm 1 2 3 4 5 6
1396: 1396 (T0: R258): 0x00425668:	adr x26, #12 F 1 2 Dc Rn 1 Is Cm 1 2 3 4 5 6 7
1397: 1397 (T0: R259): 0x0042566c:	add x1, x26, x1, LSL #2 F 1 2 Dc Rn 1 Ds 1 2 3 Is Cm 1 2 3
1398: 1398 (T0: R260): 0x00425670:	br x1 F 1 2 3 Dc Rn 1 Ds 1 2 Is Cm 1 2
1399: 1399 (T0: R261): 0x00425678:	ldr x1, [x0, #16]! F 1 2 Dc Rn 1 Is Cm 1 2 3 4 5 6 7 8 9 10 11 12 13 14
1400: 1400 (T0: R262): 0x00425678:	addxi_uop x0, x0, #16 F 1 2 Dc Rn 1 Is Cm 1 2 3 4 5 6 7 8 9 10 11 12 13 14
1401: 1401 (T0: R263): 0x0042567c:	cbnz x1, 0 0x00425678: addxi_uop x0, x0, #16 Dc Rn 1 Ds 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
1402: 1402 (T0: R-1): 0x00425680:	add x0, x5, Fetched Tick: 1610000 Dc Rn 1 Is
1403: 1403 (T0: R-1): 0x00425684:	str x7, [x1 Line: 9787 Global Serial ID: 1400 DC Rn 1 IS
1404: 1404 (T0: R-1): 0x00425688:	str x25, [x Thread ID: 0 DC Rn 1 IS
1405: 1405 (T0: R-1): 0x0042568c:	subs w2, #1 Retire ID: 262 DC Rn 1 Is
1406: 1406 (T0: R-1): 0x00425690:	str x24, [x13, #2536] F 1 Dc Rn 1 Is
1407: 1407 (T0: R-1): 0x00425694:	str x23, [x12, #2592] F 1 Dc Rn 1 Is
1408: 1408 (T0: R-1): 0x00425698:	str x22, [x5, #3536] F 1 Dc Rn 1 Is
1409: 1409 (T0: R-1): 0x0042569c:	str x21, [x0, #8] F 1 Dc Rn 1 Is
1410: 1410 (T0: R-1): 0x004256a0:	str x20, [x0, #16] F Dc Rn 1 Ds Is
1411: 1411 (T0: R-1): 0x004256a4:	str x19, [x11, #2568] F DC Rn 1 DS IS
1412: 1412 (T0: R-1): 0x004256a8:	str x18, [x10, #768] F DC Rn 1 Ds Is
1413: 1413 (T0: R-1): 0x004256ac:	str x17, [x9, #2640] F Dc Rn 1 Ds Is
1414: 1414 (T0: R-1): 0x004256b0:	str x16, [x8, #624] F 1 DC Rn 1 DS IS
1415: 1415 (T0: R-1): 0x004256b4:	b.ne 0x4256e0 F 1 Dc Rn 1 Is

- 1. A brief explanation of how to use
- 2. Typical visualization examples
- 3. Use cases

- 1. Install: All you have to do is to download the package and unpack it.
  - https://github.com/shioyadan/Konata/releases
  - ◇ Windows/Linux/Mac packages are provided.
  - One of the second se
- 2. Start the executable file such as Konata.exe

- 1. Generate a trace log from gem5 with the O3 CPU model
  - $\bigcirc$  Execute gem5 with the following flags
  - ./build/ARM/gem5.opt --debug-flags=03PipeView --debug-start=<first
    tick of interest> --debug-file=trace.out configs/example/se.py
    --cpu-type=detailed --caches -c <path to binary> -m <last cycle of
    interest>
  - This example is from http://www.m5sim.org/Visualization
- 2. Load the generated "trace.out" to Konata
  - $\diamond$  from the menu in the window or using drag&drop

### How to use

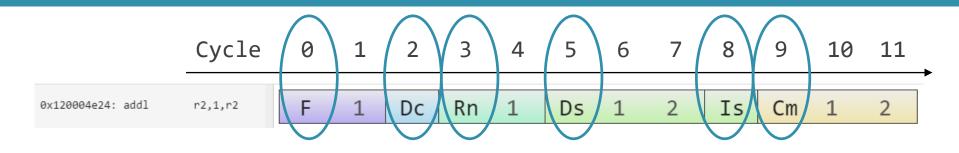
• After loading the file, contents like the following are shown.

 $\diamond$  Left side: instruction information such as a PC and mnemonic

◇ Right side: the image of visualized pipeline behavior

🎯 Konata													-	_		×
File Window View Help																
trace.out 🗙																
16579: 16579 (T0: R14097): 0x1	120004740: addl	r31,r1,r1	F	Dc	Rn	1	Is	Cm	1							
16580: 16580 (T0: R14098): 0x1	120004744: cmpeq	r2,r5,r3	F	Dc	Rn	1	Is	Cm	1							
16581: 16581 (T0: R14099): 0x1	120004748: s4addq	r4,r6,r4	F	Dc	Rn	1	Ds	Is	Cm	1						
16582: 16582 (T0: R14100): 0x1	12000474c: ldl	r4,0(r4)	F	Dc	Rn	1	Ds	1	Is	Cm	1	-2	3			
16583: 16583 (T0: R14101): 0x1	120004750: xor	r4,r1,r1		F	Dc	Rn	1	Ds	1	2	3	[649	95, 10	6581]		
16584: 16584 (T0: R14102): 0x1	120004754: stl	r1,648(r16)		F	Dc	Rn	1	Ds	1	2	3	4	Is	Cm	1	
16585: 16585 (T0: R14103): 0x1	120004758: beq	r3,0x120004		F	Dc	Rn	1	Is	Cm	1	2	3	4	5	6	
16586: 16586 (T0: R14104): 0x1	120004730: extbl	r1,3,r4				F	Dc	Rn	1	Ds	1	2	Is	Cm	1	
16587: 16587 (T0: R14105): 0x1	120004734: sll	r1,8,r1				F	Dc	Rn	1	Ds	1	2	Is	Cm	1	
16588: 16588 (T0: R14106): 0x1	120004738: addl	r2,1,r2				F	Dc	Rn	1	Is	Cm	1	2	3	4	
16589: 16589 (T0: R14107): 0x1	12000473c: xor	r4,r7,r4				F	Dc	Rn	1	Ds	1	2	3	Is	Cm	1

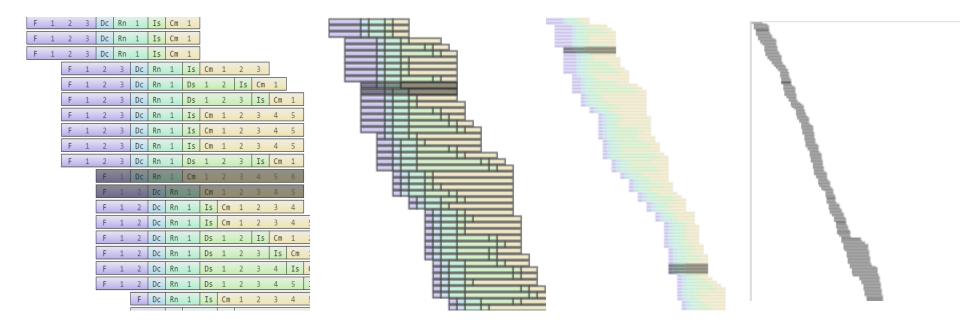
### How to see the visualized image



- The clock cycle proceeds from left to right
  - $\bigcirc$  F : Instruction fetch
    - □ In this insn., the fetch latency is 2 cycles
  - Oc : Instruction Decode
  - 🔷 Rn : Rename
  - ◇ Ds : Dispatch
  - $\diamond$  IS : Issue
  - $\diamond$  Cm : Completion of execution
    - □ The execution stage is not explicitly shown
  - $\bigcirc$  (The end of Cm stages) : Retire

### Zoom in/out

#### You can zoom in/out as follows:



zoom-in

zoom-out

### **Compare Two Pipelines**

- Konata can show two pipelines overlapping as follows:
  - 1. Load two files
  - 2. Right click -> "Transparent mode" & "Synchronize scroll"



## **Typical Visualization Examples**

- Introduce how the following things are shown:
  - 1. Out-of-order execution
  - 2. Branch misprediction
  - 3. Cache miss
  - 4. Execution speed

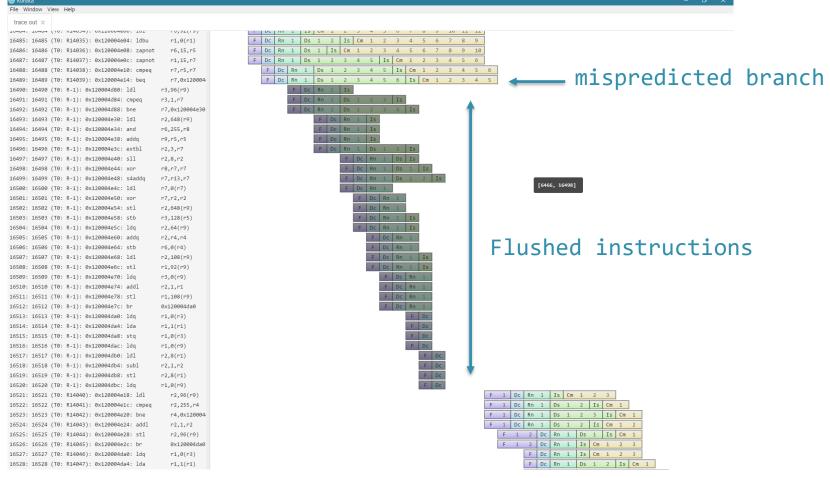
### **Example: Out-of-order Execution**

- Fetch and retirement, marked with the blue circles, are performed in-order
  - $\diamond$  Instruction issue, marked with the red circles, is performed out-of-order

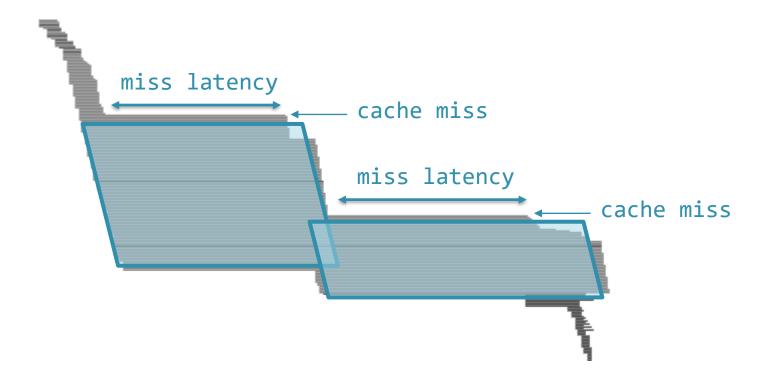
🛞 Konata	– 🗆 X
File Window View Help	
trace.out ×	$\frown$
6461: 6461 (T0: R2421): 0x120016d44: cmpeq	r10,r8,r7 F D Rn 1 Ds 1 2 3 IS Cm 1
6462: 6462 (T0: R2422): 0x120016d48: cmpeq	r7,0,r7 F Dc Rn 1 Ds 1 2 3 4 5 Is Ch 1
6463: 6463 (T0: R2423): 0x120016d4c: and	r3,r7,r7 F Dc Ro 1 Ds 1 2 3 4 5 6 Is
6464: 6464 (T0: R2424): 0x120016d50: beq	r7,0x12091 F 1 Dc Rn 1 Ds 1 2 3 4 5 6 Is (000100)
6465: 6465 (T0: R2425): 0x120016d54: ldq	r3,568(rL5 F 1 DC Rn I IS Cm 1 2 3 4 5 7 [4674, 6163] Cm[2]
6466: 6466 (T0: R2426): 0x120016d58: bne	r3,0x12001 F 1 Dc R Ds 1 2 Is cm 2 3 4 5
6467: 6467 (T0: R2427): 0x120016d5c: subl	r10,48,r: F 1 Dc Rn 1 D5 1 2 3 I5 Cm 2 3 4
6468: 6468 (T0: R2428): 0x120016d60: zapnot	r3,15,r3 F Dc Rn 1 Ds 1 2 3 Is Cm 2 3
6469: 6469 (T0: R2429): 0x120016d64: cmpule	r3,9,r3 F Dc Rn Ds 1 2 3 4 T Is Cm 1 2
6470: 6470 (T0: R2430): 0x120016d68: beq	r3,0x1200 F Dc Rn 1 D5 1 2 3 4 5 0 Is Cm L
6471: 6471 (T0: R2431): 0x120016d6c: ldq	r0,616(r19 F Dc Rn 1 2 - 4 5 6 7 8
6472: 6472 (T0: R2432): 0x120016d70: subl	r10,47,r3 F 1 DC R 1 DS 1 2 IS Cm 1 2 3 4 5
6473: 6473 (T0: R2433): 0x120016d74: cmple	r3,r0,r3 F 1 DC Rn DS 1 2 1 IS Cm 1 2 3 4
6474: 6474 (T0: R2434): 0x120016d78: bne	r3,0x12001 F 1 DC Rn 1 Ds 1 2 3 4 Is C 1 2 3
6475: 6475 (T0: R2435): 0x120017440: cmpeq	r12,r2,r7 F DC Rt Is m 1 2 4 5 6 7
6476: 6476 (T0: R2436): 0x120017444: sextb	r10,r3 F DC Rn 1 Ds 1 Is Cm 1 2 3 4 3
6477: 6477 (T0: R2437): 0x120017448: bne	r7,0x12001 F DC Rn 1 Ds Is Cm 1 2 3 4 5 6
6478: 6478 (T0: R2438): 0x12001744c: addq	r4,r12,r22 F DC Rn 1 Is Cm 1 2 3 4 5 6 7
6479: 6479 (T0: R2439): 0x120017450: subl	r6,1,r8 F DC RN IS Cm 1 2 3 4 5 6 7
6480: 6480 (T0: R2440): 0x120017454: lda	r7,1(r10) F DC Rn 1 DS 1 IS Cm 1 2 3 4 5 5
6481: 6481 (T0: R2441): 0x120017458: lda	r12,1(r12) F DC Rn 1 Is Cm 1 2 3 4 5 6 7 8
6482: 6482 (T0: R2442): 0x12001745c: stb	r3,0(r22) F DC Rn 1 DS 1 2 IS Cm 1 2 3 4 5
6483: 6483 (T0: R2443): 0x120017460: cmovgt	r6,r8,r6 F Dc Rn 1 Is Cm 1 2 3 4 5 6 7
6484: 6484 (T0: R2444): 0x120017464: beq	r7,0x12001 F Dc Rn 1 Ds 1 Is Cm 1 2 3 4 5
6485: 6485 (T0: R2445): 0x120017468: ldq	r7,8(r11) F Dc Rn 1 Is Cm 1 2 3 4 5 6 7
6486: 6486 (T0: R2446): 0x12001746c: ldq	r3,16(r11) F Dc Rn 1 Is Cm 1 2 3 4 5 6 7
6487: 6487 (T0: R2447): 0x120017470: cmpult	r7,r3,r3 F Dc Rn 1 Ds 1 2 Is Cm 1 2 3 4
6488: 6488 (T0: R2448): 0x120017474: beq	r3,0x12001 F Dc Rn 1 Ds 1 2 3 Is Cm 1 2 3 4 5 6 7
6489: 6489 (T0: R2449): 0x120017478: lda	r3,1(r7) F Dc Rn 1 Ds 1 Is Cm 1 2 3 4 5 6 7 8
6490: 6490 (T0: R2450): 0x12001747c: ldbu	r10,0(r7) F Dc Rn 1 Ds 1 Is Cm 1 2 3 4 5 6 7 8 9
6491: 6491 (T0: R2451): 0x120017480: stq	r3,8(r11) F Dc Rn 1 Ds Is Cm 1 2 3 4 5 6 7 8

### **Example: Branch Misprediction**

#### $\diamond$ Flushed instructions are shown as dark ones

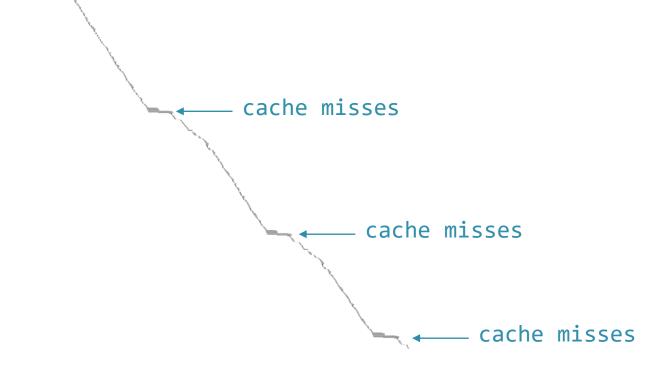


A cache miss is typically shown as a diamond-like shape when the image is zoomed out as follows



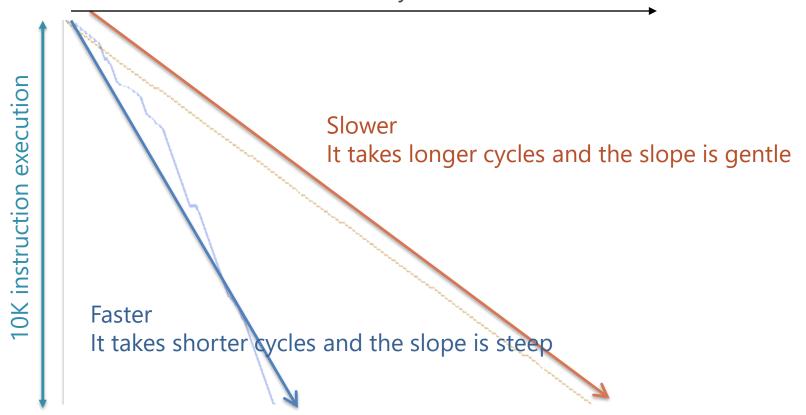
### **Example: Cache Misses**

- As it is zoomed out more, the pipeline is typically shown as follows
  - $\diamond$  This is the pipeline behavior of MCF in SPECCPU 2006
  - This figure shows the performance is degraded by the cache misses



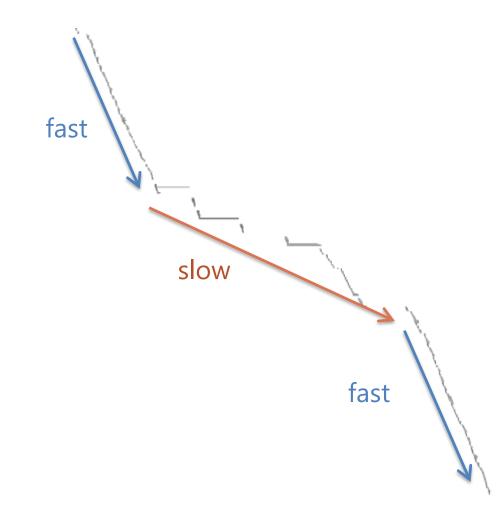
### **Example: Execution Speed**

- The slope of a pipeline shape roughly represents the execution speed (IPC).
- The following two pipelines show the execution of the same 10K instructions
  clock cycle



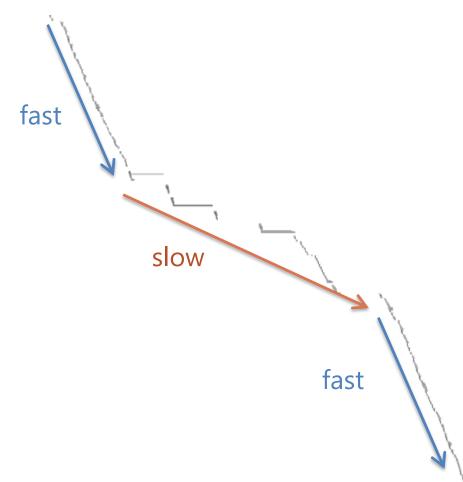
## The slope of a pipeline shape roughly represents the execution speed (IPC)

You can see the transition in the execution speed for each part of the program as follows



# The slope of a pipeline shape roughly represents the execution speed (IPC)

- $\diamond$  It is not accurate because flushed instructions are also shown.
- If you want to compare accurately, use "Hide flushed ops" option from the right click menu



- 1. A brief explanation of how to use
- 2. Typical visualization examples
- 3. Use cases
  - 1. Grasping the pipeline behavior
  - 2. Comparing pipelines

- The pipeline visualization makes it easy to grasp the pipeline behavior
  - $\diamondsuit$  Explain this by some use cases
- Let's suppose you newly add speculative execution with branch prediction
  - $\diamond$  (Of course, gem5 already has this feature
  - ◇ Something wrong happens in recovery from mispredictions

# Investigating with a log

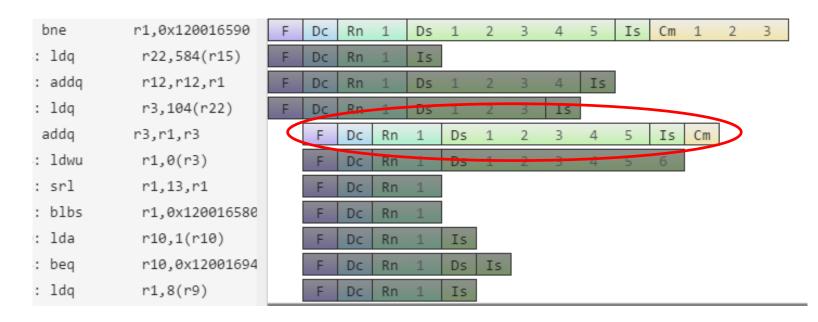
- 15 O3PipeView:fetch:25069250:0x120016358:0:177:addq r12,r12,r1 16 O3PipeView:decode:25069500 O3PipeView:rename:25069750 17 18 O3PipeView:dispatch:25070250 O3PipeView:issue:25071500 19 O3PipeView:complete:25071750 20 O3PipeView:retire:0:store:0 21 O3PipeView:fetch:25069250:0x12001635c:0:178:ldq r3,104(r22) 22 23 O3PipeView:decode:25069500 24 O3PipeView:rename:25069750 O3PipeView:dispatch:25070250 25 O3PipeView:issue:25071250 26 27 O3PipeView:complete:25071500 O3PipeView:retire:0:store:0 28 O3PipeView:fetch:25069500:0x120016360:0:179:addq r3,r1,r3 29 O3PipeView:decode:25069750 30 31 O3PipeView:rename:25070000 32 O3PipeView:dispatch:25070500 O3PipeView:issue:25072000 33 34 O3PipeView:complete:25072250 O3PipeView:retire:25072500:store:0 35 36 O3PipeView:fetch:25069500:0x120016364:0:180:ldwu r1,0(r3) O3PipeView:decode:25069750 37 38 O3PipeView:rename:25070000 O3PipeView:dispatch:25070500 39 O3PipeView:issue:25072250 40 41 O3PipeView:complete:0 42 O3PipeView:retire:0:store:0 43 O3PipeView:fetch:25069500:0x120016368:0:181:srl r1,13,r1 O3PipeView:decode:25069750 44 45 O3PipeView:rename:25070000 46 O3PipeView:dispatch:25070500 O3PipeView:issue:0 47 O3PipeView:complete:0 48
- 49 03PineView:retire:0:store:0

For investigating your implementation, you probably:

- Check custom logs or your "printf" outputs such as the left example
- It records when/what instructions are flushed.
- It's very difficult to detect which point is incorrect from such text logs.

25

# Investigating with visualization

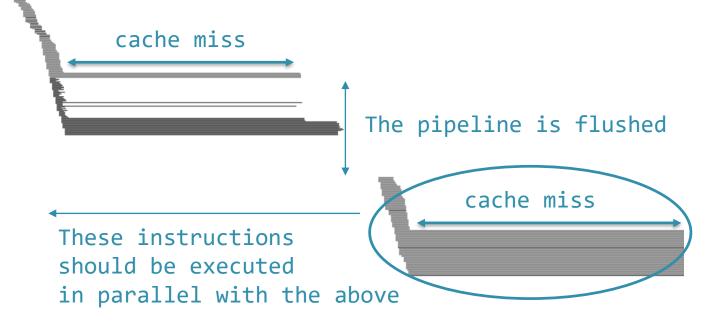


- By visualizing it, you can easily notice the incorrect point.
  - There is the light instruction (not flushed) between the dark flushed instructions.
- Although this is an artificial example,
  - $\diamond$  visualization gives us a lot of hints intuitively

- One of my friends researched a theme related to memory level parallelism
  - In short, his method improves the performance by performing multiple memory accesses in parallel
- He enlarged the size of the OoO scheduling window so that more memory accesses are performed in parallel
  - $\diamond$  But, the performance is not improved

## Another example: memory level parallelism

- He realized that something wrong happened from the following zoomed out image,
  - $\diamond$  because the shape is unnatural
- He realized that the pipeline was flushed on a cache miss
  - In this sequence, memory accesses should be performed in parallel



He examined the flushed instruction in detail and found the cause.

- This was because he used Alpha ISA
  - In Alpha architecture, TLB miss causes a trap and the pipeline is flushed
  - $\diamond$  On a cache miss, a TLB miss often occurs
  - ◇ So memory accesses cannot be performed in parallel
- It is not easily noticed simply by observing the counters in gem5.
  - $\diamond$  The shape or pattern of visualized pipelines often tell us hints.

- 1. A brief explanation of how to use
- 2. Typical visualization examples
- 3. Use cases
  - 1. Grasping the pipeline behavior
  - 2. Comparing pipelines

### Let's suppose

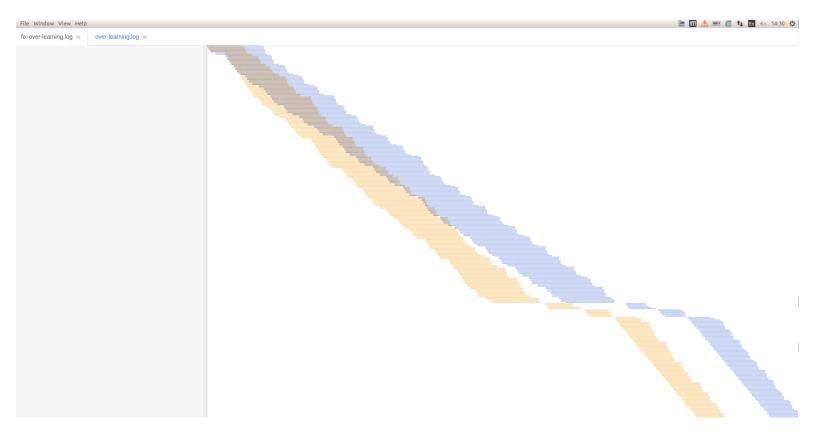
- $\diamond$  your new method seems to work correctly,
- $\diamond$  but it does not improve the performance as you expected.

### Konata can compare two pipelines.

 $\diamond$  It is useful when investigating the above situation.

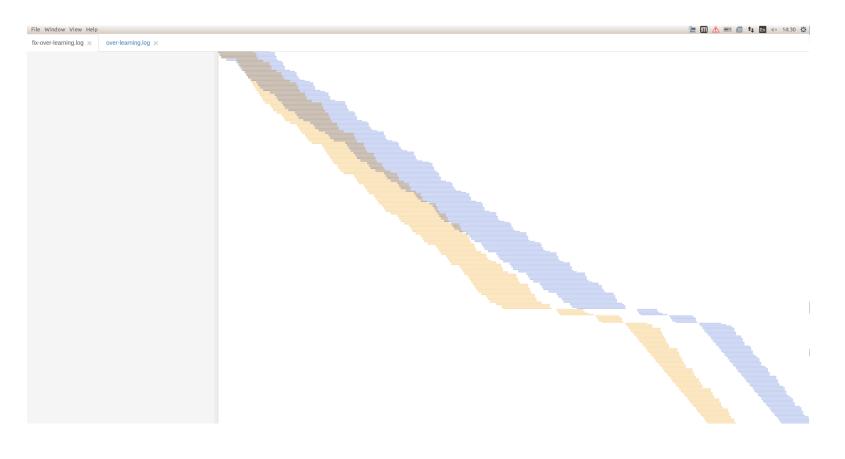
## **Example of comparing**

- $\diamond$  My friend implemented the new method to the baseline CPU.
- $\diamond$  Konata can show two pipelines overlapping.
  - Blue shows a baseline pipeline
  - □ Orange shows a pipeline with a new method



## **Example of comparing**

### ◇ The orange one (new) is basically faster than the blue one (baseline)

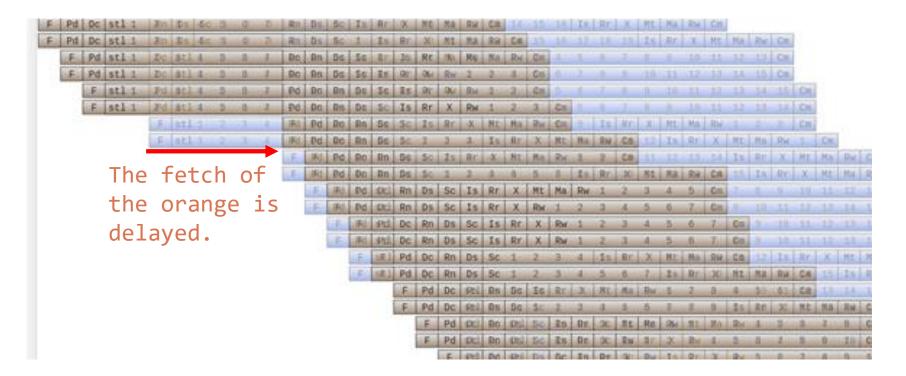


### **Example of comparing**

In the zoomed-in image,

 $\diamond$  in some places, the fetching of the orange is unreasonably delayed.

 $\diamond$  This was caused by a bug in fact.



- Visual comparison is very effective for analysis when adding new features to gem5.
  - If the performance is not improved as expected, something is delayed.
  - $\diamond$  You can detect such parts by visual comparison.
    - □ It is easy to see which part is different.

### Conclusion

- It is generally difficult to investigate the cause of a bug related to the performance.
  - $\bigcirc$  Especially, when you have no idea what happened.
- In such cases, visualization is very useful.
  - ◇ This presentation introduced the pipeline visualization in gem5
- Please try it!
  - $\bigcirc$  It is simply fun to see how the processor works.
  - https://github.com/shioyadan/Konata/releases