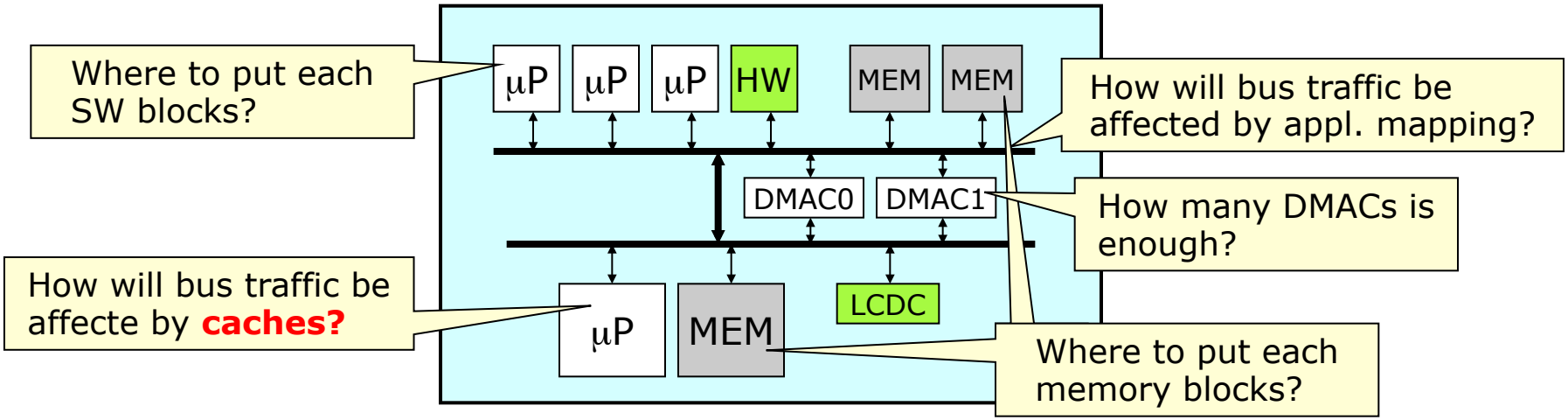

# Trace-Driven MPSoC Simulation with Cache Modeling

**Tsuyoshi Isshiki**

*Dept. of Communications and Integrated Systems*
*Tokyo Institute of Technology*

**MPSoC'13**
**July. 18th, 2013**

# MPSoC Design Exploration Requirements



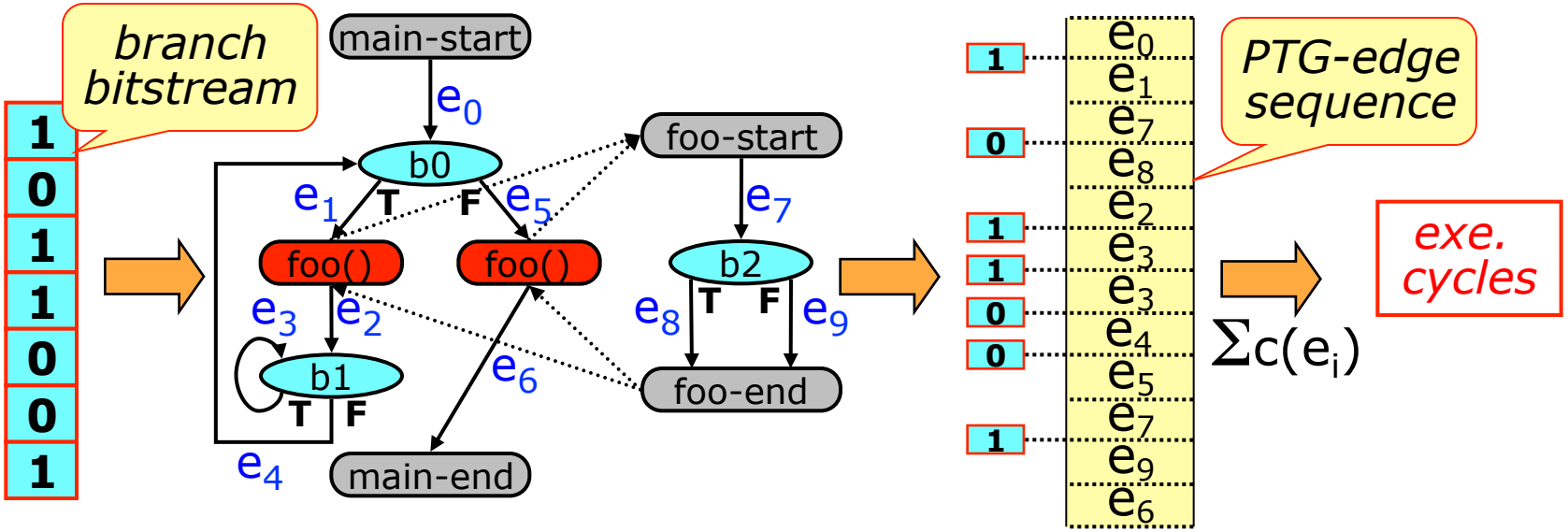

- HW/SW MPSoC design space exploration
  - SW: algorithm design, SW partitioning
  - HW: architecture (CPUs, HW blocks, busses, memories, DMACs)
  - Mapping: SWs→CPUs, memory mapping
- MPSoC architecture evaluation
  - System performance: cycle counts, power
  - Locating system bottlenecks: bus/memory bandwidth?, CPUs? caches?

→ Framework for efficient evaluation of SW/HW design choices

# Trace-Driven Workload Model

## [@DAC'09, MPSoC'09]

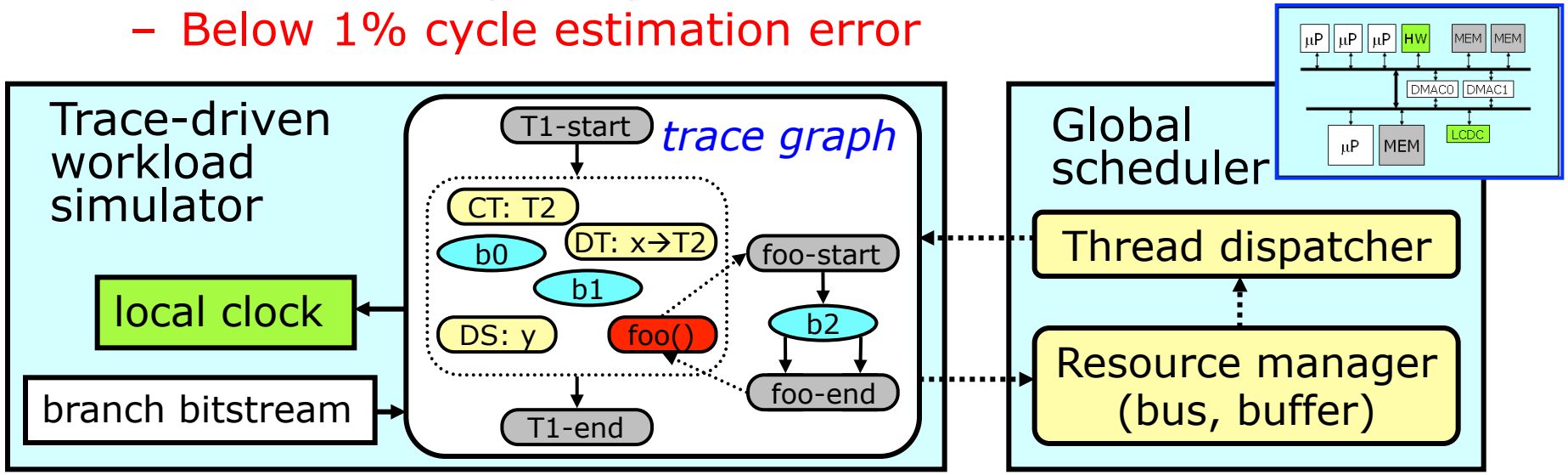
- Branch bitstream: Branch condition bit sequence recorded in program execution order (via source-level instrumentation and native code execution)
- Program Trace Graph: degenerate ICFG at basic-blocks
- PTG-edge [$e_i$] : code segment without conditional jumps
- PTG-edge cycle count [$c(e_i)$] : extracted by compiler backend




*Program Trace Graph*

# MPSoC Trace-Driven Workload Simulator

- MPSoC Trace-driven workload simulator
  - Coordinate parallel workload models with system synchronization events
    - CT(thread activation), DT(data transfer), DS(data synch.)
  - Parameterized MPSoC architecture model
    - # of processors, interconnect topology
  - 70x ~ 200x speedup over ISS-based simulation
  - Below 1% cycle estimation error

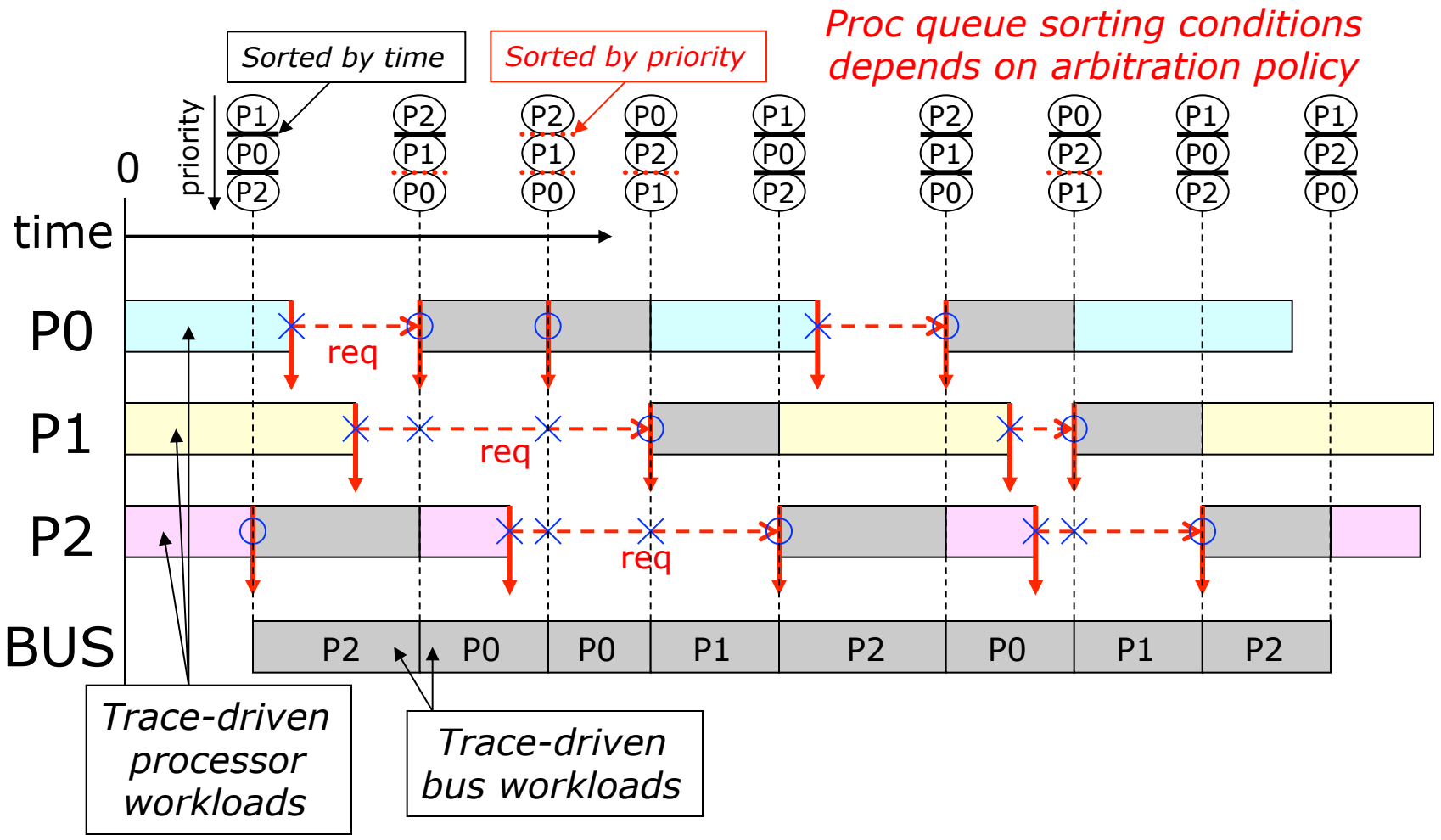


# Trace-Driven Bus Traffic Simulation
[@MPSoC'11]

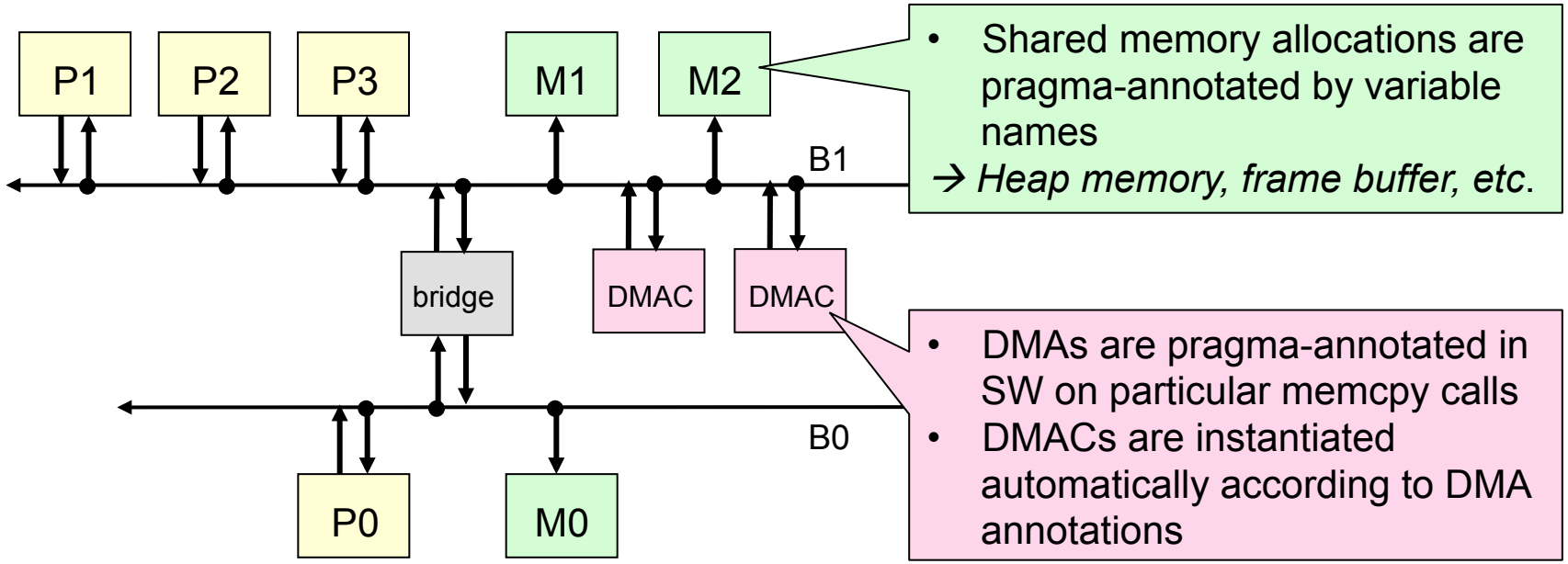
- Trace-driven bus traffic generation
  - Triggered by *processor communications, accesses to memories and IPs*
  - **PTG-sync-nodes** inside thread-PTGs generate the bus traffic sequence with accurate (dynamically changing) intervals and payloads according to a particular set of thread-BBs
- Bus arbitration simulation
  - Schedules multiple outstanding bus requests to resolve bus conflicts
  - Several standard arbitration schemes supported (fixed priority, round-robin, QoS guaranteed)

# Trace-Driven Bus Traffic Simulation

- Bus traffic scheduling done at cycle-accurate level



# MPSoC Architecture Description with Shared-Memories and Bus Bridges




```
BusMapping{
  {P0,M0},              // bus_0 : proc{P0}, mem{M0}
  {P1,P2,P3,M1,M2 }     // bus_1 : proc{P1,P2,P3}, mem{M1,M2}
}
BridgeMapping{
  {B0, B1} // bus_0 and bus_1 are connected through bridge
}
```

# Bus Traffic Workload Simulation


- TinyGL (Open-GL subset) "WHEEL" 3D animation

→ *Uses MPSoC model in previous page*

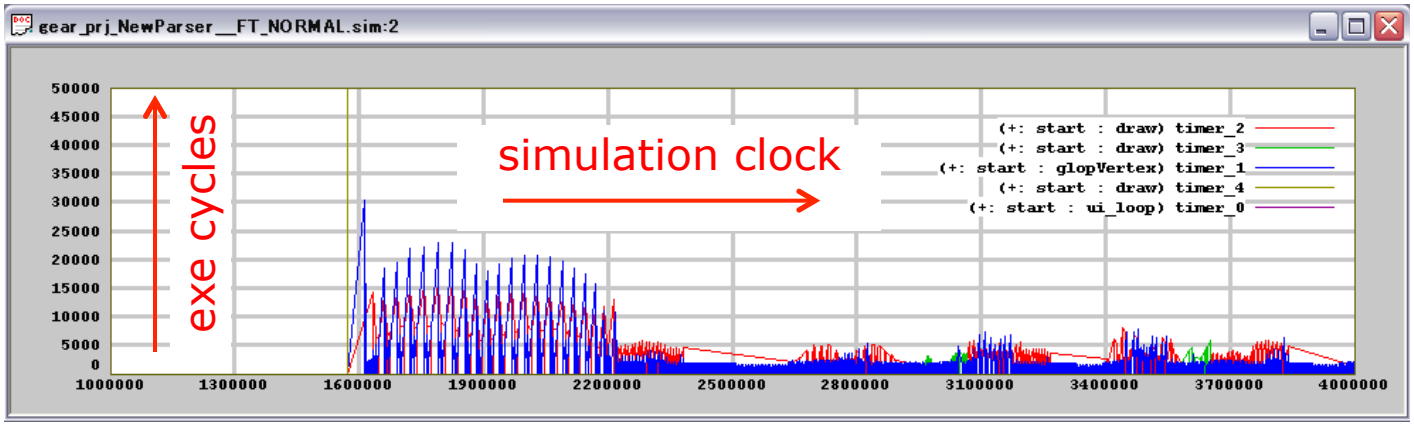
| Execution platform | 100 frames | | 1000 frames | |
|---|---|---|---|---|
| SW exe (Xeon 3.4GHz) | 0.165 sec | 1.00 | 1.158 sec | 1.00 |
| Trace-sim (bus:ON) | 6.222 sec | 37.71 | 62.871 sec | 54.29 |
| Trace-sim (bus:OFF) | 1.203 sec | 7.29 | 12.169 sec | 10.51 |
| Trace-sim (bus:OFF+reduction) | 0.684 sec | 4.15 | 6.878 sec | 5.94 |

| Bus model | PTG reduction | Simulation time | Estimated cycles | error |
|---|---|---|---|---|
| ON | OFF | 6.222 sec | 339,828,586 | --- |
| OFF | OFF | 1.203 sec | 328,726,603 | 3.27% |
| OFF | ON | 0.684 sec | 321,540,634 | 5.38% |

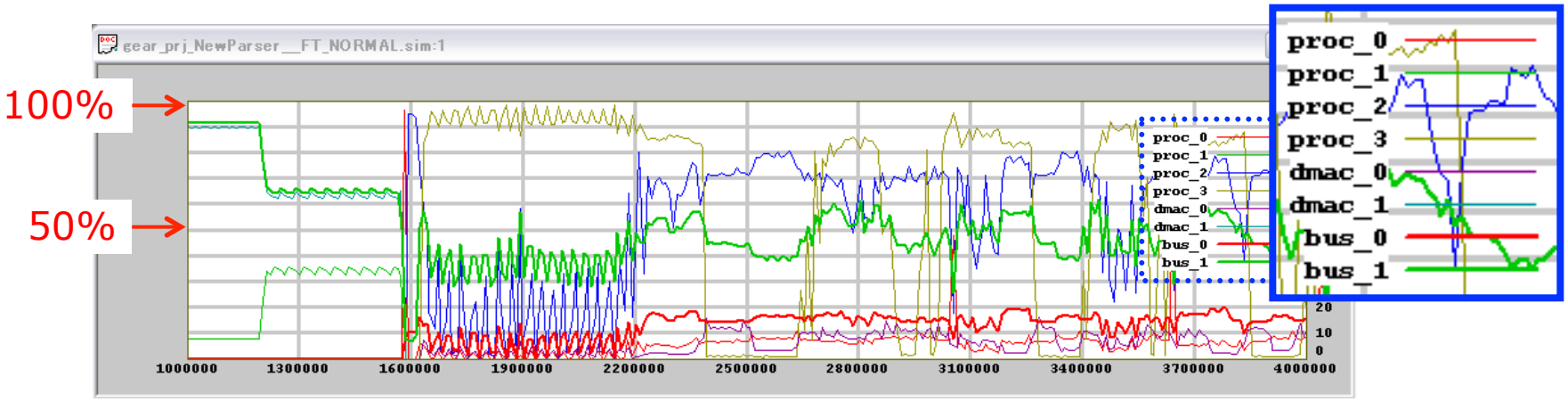
- *Cycle estimation errors are against "bus model = ON"*
- *Bus traffic simulation requires 5x simulation time → very expensive*
- *PTG-edges become more fine-grain when shared memory accesses exit (PTG-reduction merges all memory accesses)*

# Processor/Bus Workload Profiling

- Subroutine exe. cycles showing *data dependent workloads*

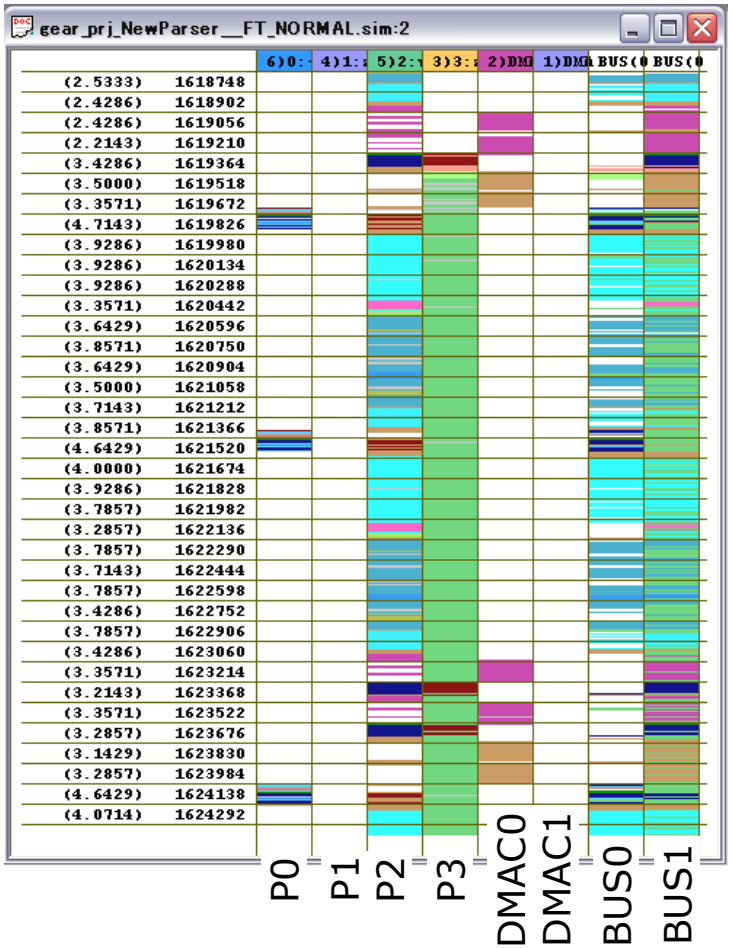



- Dynamic workload profiles for processors and busses

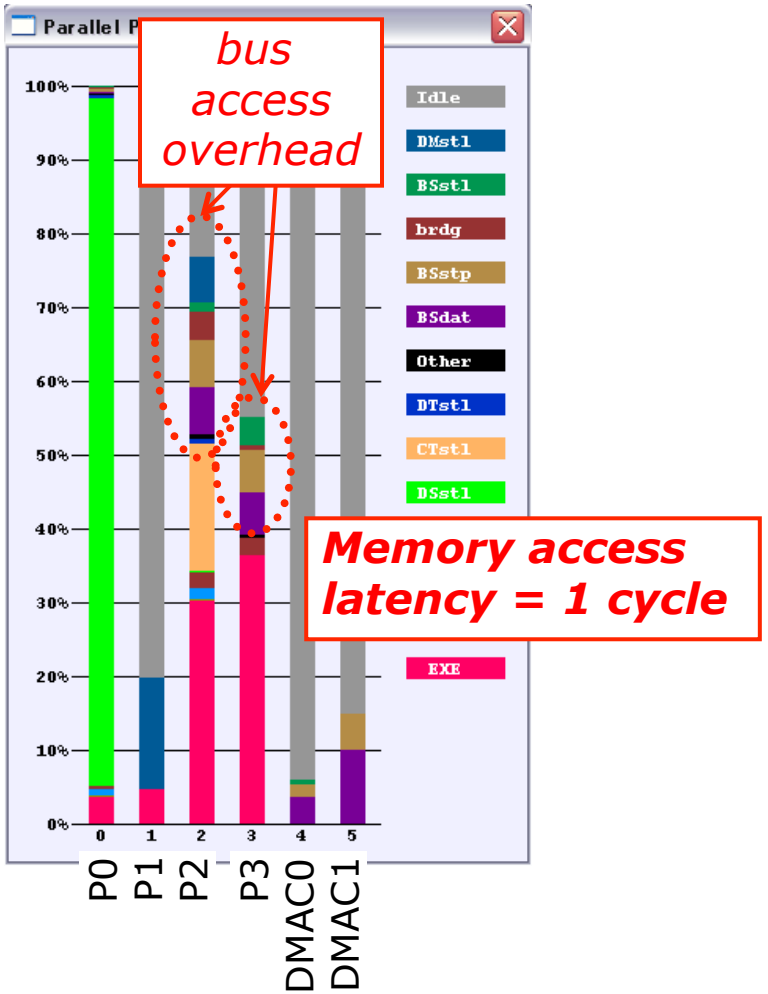


# Processor/Bus Workload Profiling

- Workload scheduling details

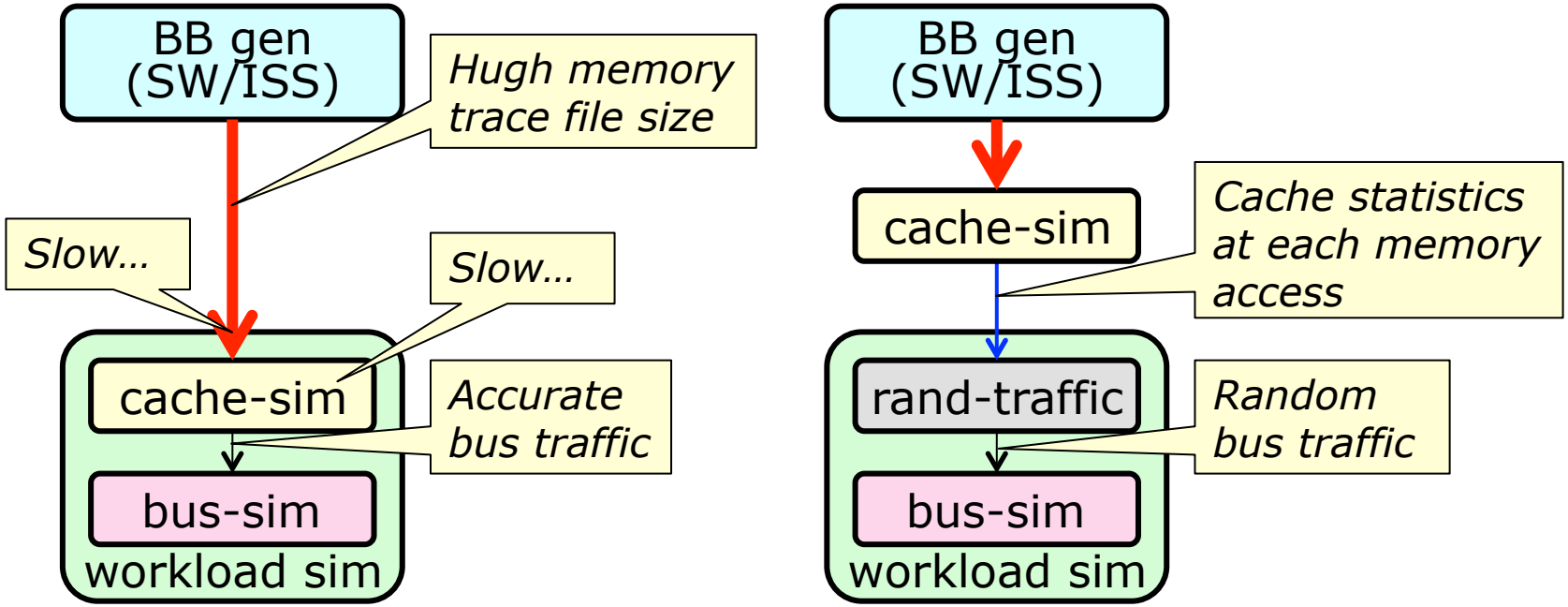



- Workload percentages



# Cache Workload Modeling

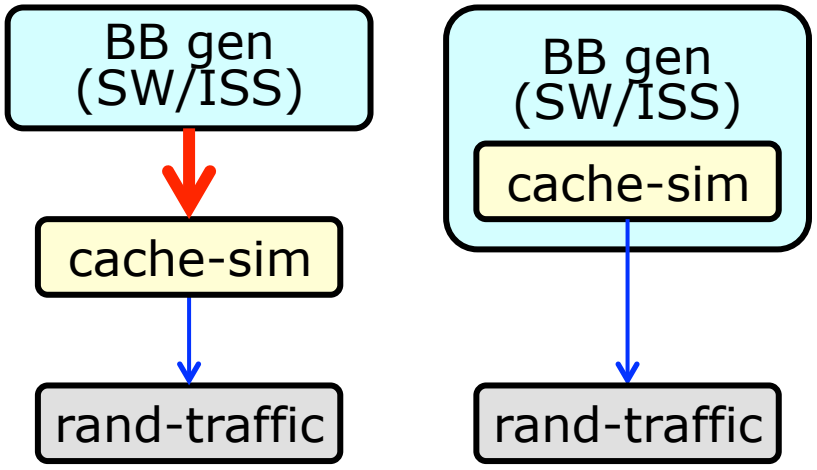
- Cache miss event → *dynamic bus traffic generation*
- Put cache simulator inside or outside the workload simulator?
  - Inside: accurate, but very slow
  - Outside: fast, but accuracy may become an issue…



# Cache Workload Modeling

- Assuming we put the cache simulator outside the workload simulator, should we put it inside or outside the BB generator?

→ *If BB generator with cache simulator runs faster than the standalone one, it makes sense to put it inside…*

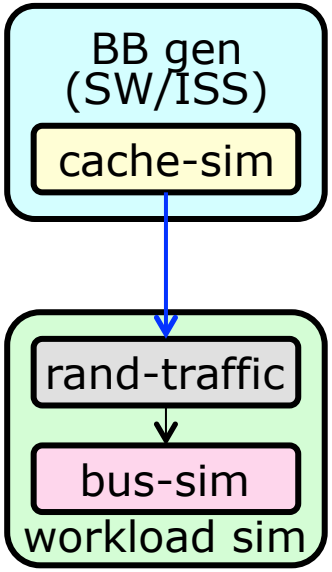



| 123M memory traces (~1GB) | |
|---|---|
| SW instrumentation | time |
| BB-gen | 0.153 sec |
| BB-gen + cache-sim | 2.037 sec |
| BB-gen + trace output | 3.035 sec |

*Trace output file access overhead is larger than cache-sim time*

# Cache Workload Modeling

- Memory tracing during BB generation
  - → *Required only for D-cache*
  - → *Instruction trace obtained from PTG-edge seq.*
- Cache simulator → cache statistics
  - D-cache: separate statistics at each memory access operation
  - I-cache: separate statistics at each PTG-edge




- MPSoC cache simulation during sequential BB generation
  - N processors → N cache models
  - *Switch the cache model when crossing the thread boundaries during BB gen.*

# SW Instrumentation for I/D-Cache Tracing

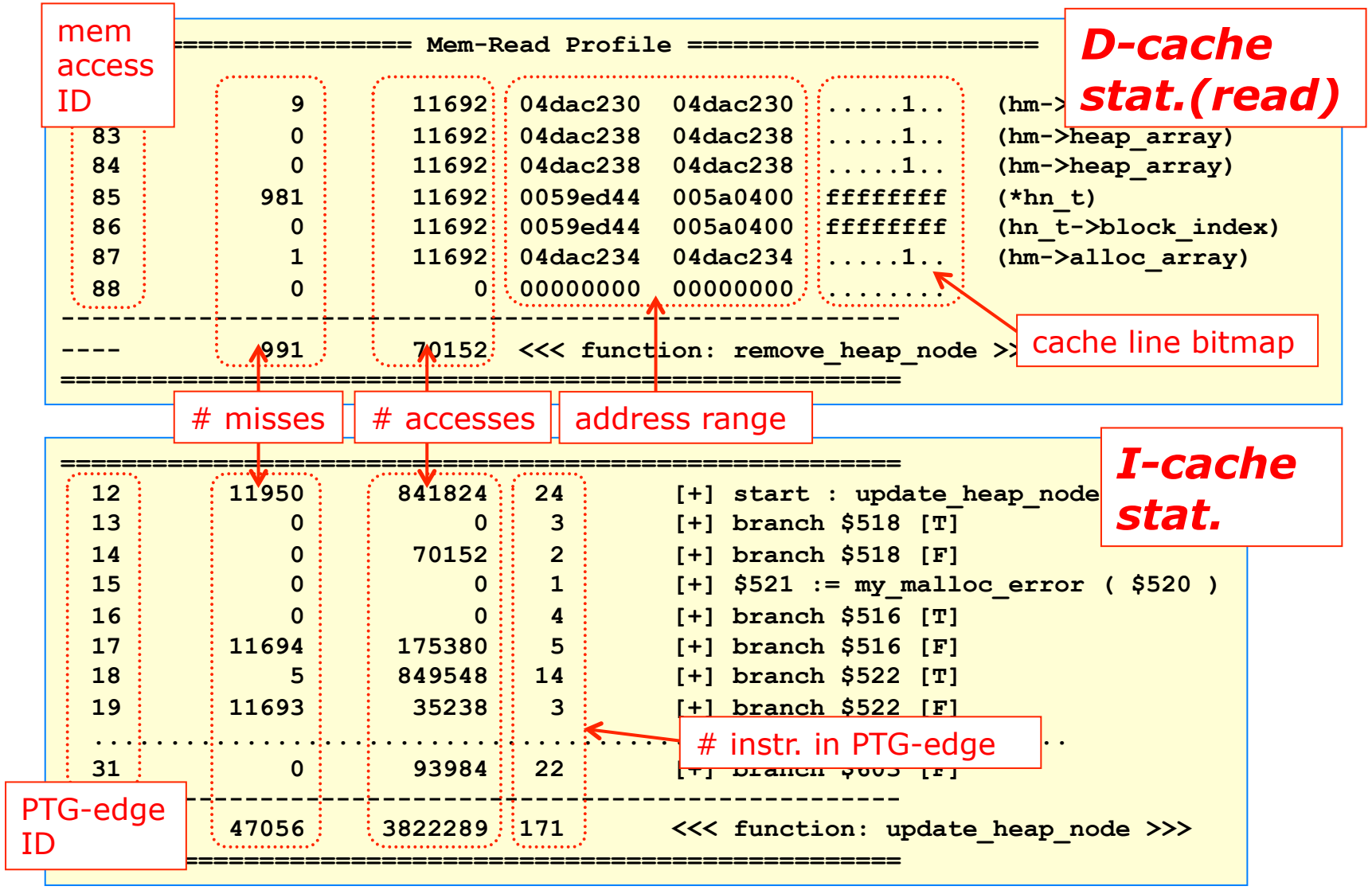
```
static void v4dwt_decode_step1(v4 * w, int count, const dwt_real_t c)
{
  dwt_real_t * fw = (dwt_real_t *) w;
  int i;
  for(i = 0; __BC_L_(i < count, 121); ++ i){
    __Mem_R_PT__(fw[0], 276);
    fw[0] = ((fw[0] * c) + ((fw[0] * c) & 4096)) >> 13;
    __Mem_W_PT__(fw[0], 178);
    __Mem_R_PT__(fw[1], 277);
    fw[1] = ((fw[1] * c) + ((fw[1] * c) & 4096)) >> 13;
    __Mem_W_PT__(fw[1], 179);
    __Mem_R_PT__(fw[2], 278);
    fw[2] = ((fw[2] * c) + ((fw[2] * c) & 4
    __Mem_W_PT__(fw[2], 180);
    __Mem_R_PT__(fw[3], 279);
    fw[3] = ((fw[3] * c) + ((fw[3] * c) & 4
    __Mem_W_PT__(fw[3], 181);
    fw += 8;
  }
}
```

**BB generation macro:**
also used for I-cache trace

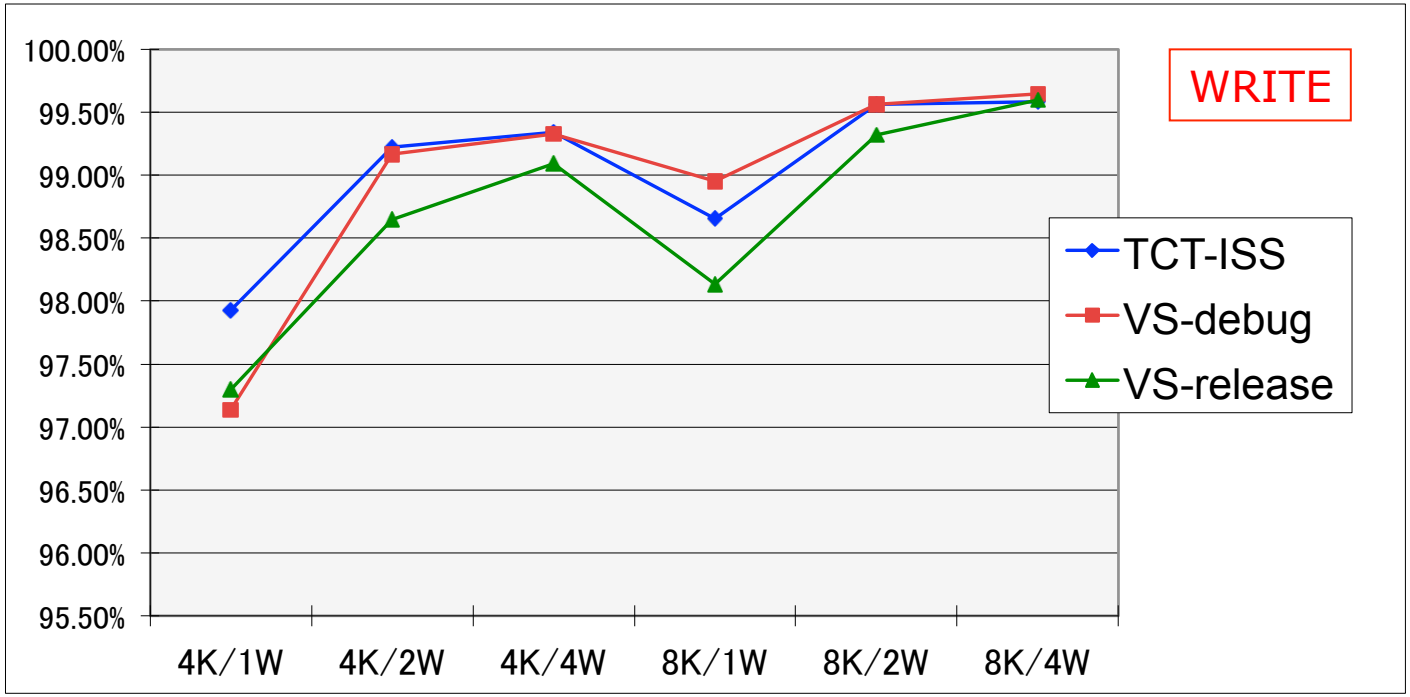
*Data traces need to match with the generated target executable code*
*→ Requires the target compiler framework to do this*

**Data trace macro:**
WRITE data trace
at (&fw[3]) with ID = 181

# SW-Instrumented Cache Statistics

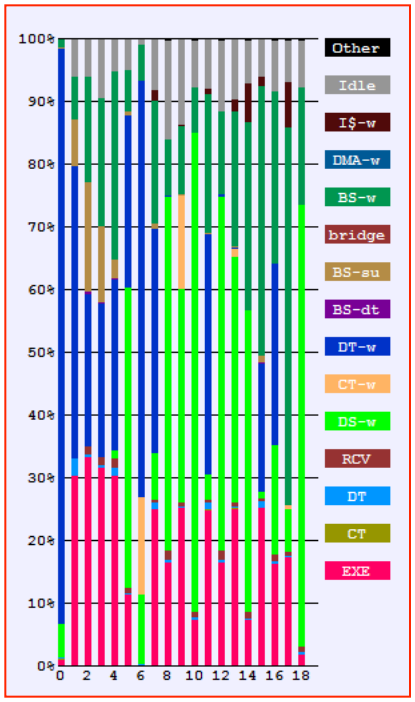


# Accuracy Issues in SW-Instrumented Data Cache Statistics

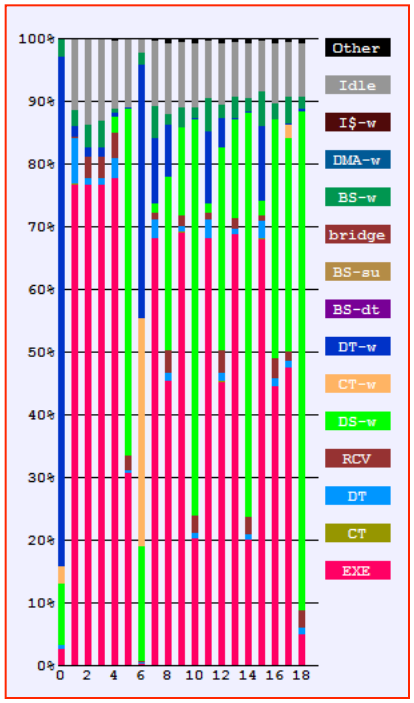



- Differences in the data trace from the native code and target code → slight deviation in statistics
- Not an issue for instruction trace

# Cache Workload Simulation
## (JPEG: 19 threads)




I/D-Cache: ON
7,838,891 cycles
0.127 sec




I/D-Cache: OFF
2,980,005 cycles
0.032 sec

I/D-Cache(OFF)

- Program memory and data memory are local
- No shared memory access

I/D-Cache(ON)

- Program memory and data memory are allocated on the same shared memory

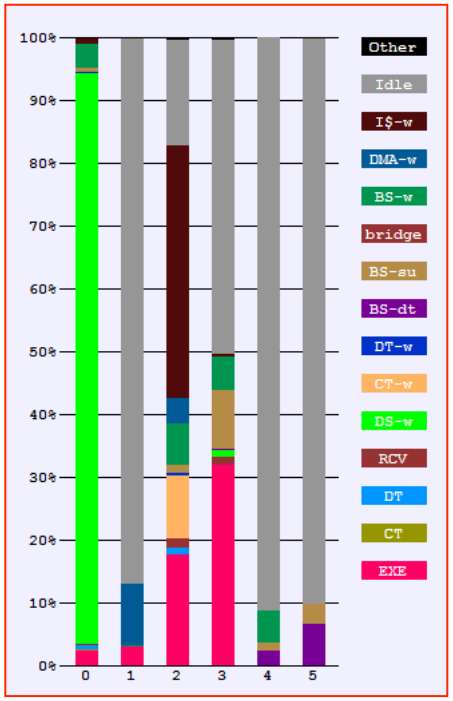
- I-cache: 4KB, 1-way
- D-cache: 4KB, 2-way

→ *cache line size: 64 bytes*

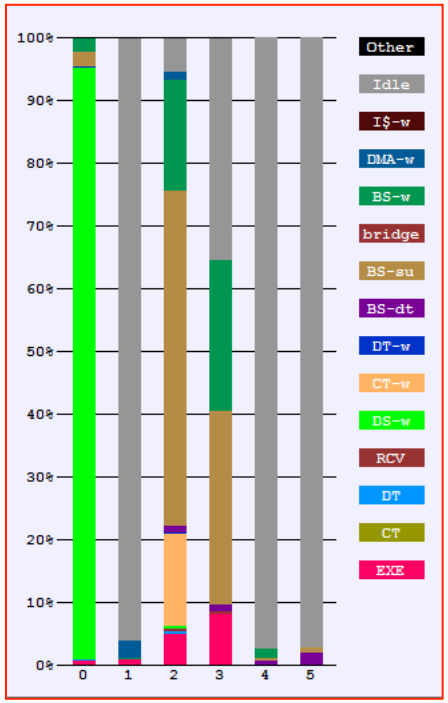
- Shared memory latencies
  - **Read: 50 cycles**
  - **Write: 10 cycles**

# Cache Workload Simulation

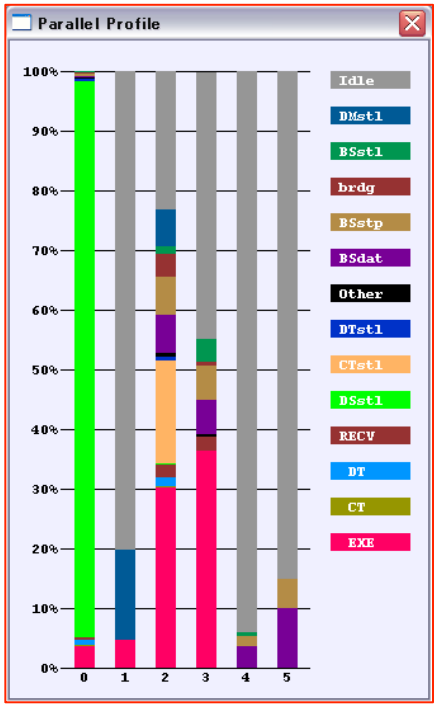
(TinyGL: 4 threads, 2 DMACs, 2 shared memories)




I/D-Cache: ON
494,542,152 cycles
2.144 sec




I/D-Cache: OFF
1,681,570,743 cycles
5.034 sec




I/D-Cache: OFF
339,828,586 cycles
6.222 sec

**Read: 50 cycle**
**Write: 10 cycle**

*memory access latencies*

**Read: 1 cycle**
**Write: 1 cycle**

# Summary

- Trace-driven bus traffic modeling
  - SW workload modeled as “Program Trace Graph”
    - SW workloads steered according to branch bitstreams (program execution trace)
  - Bus workload triggered by trace-driven SW workloads
    - Reflect detail bus traffics of “real” applications
- Cache modeling
  - Cache simulation during BB gen. → *supports multicore*
  - Random bus traffic generation (based on cache statistics at each memory access) inside workload simulation


# *Thank You for Your Attention!*

**Tsuyoshi Isshiki**

*isshiki@vlsi.ce.titech.ac.jp*

*Dept. Communications and Computer Engineering*

***Tokyo Institute of Technology***