High Performance Computing Platforms

Step1. CA02:Simulation and emulation

http://archlab.naist.jp/Lectures/ARCH/ca02/ca02e.pdf

Download one of following templates, fill in by handwriting,
and
Send PDF (scan/photo)
To: naist.report@gmail.com
Subject: 4092-xxxxxxx (student ID)

http://archlab.naist.jp/Lectures/ARCH/ca02/ca02e.docx

These links are in http://archlab.naist.jp/Lectures

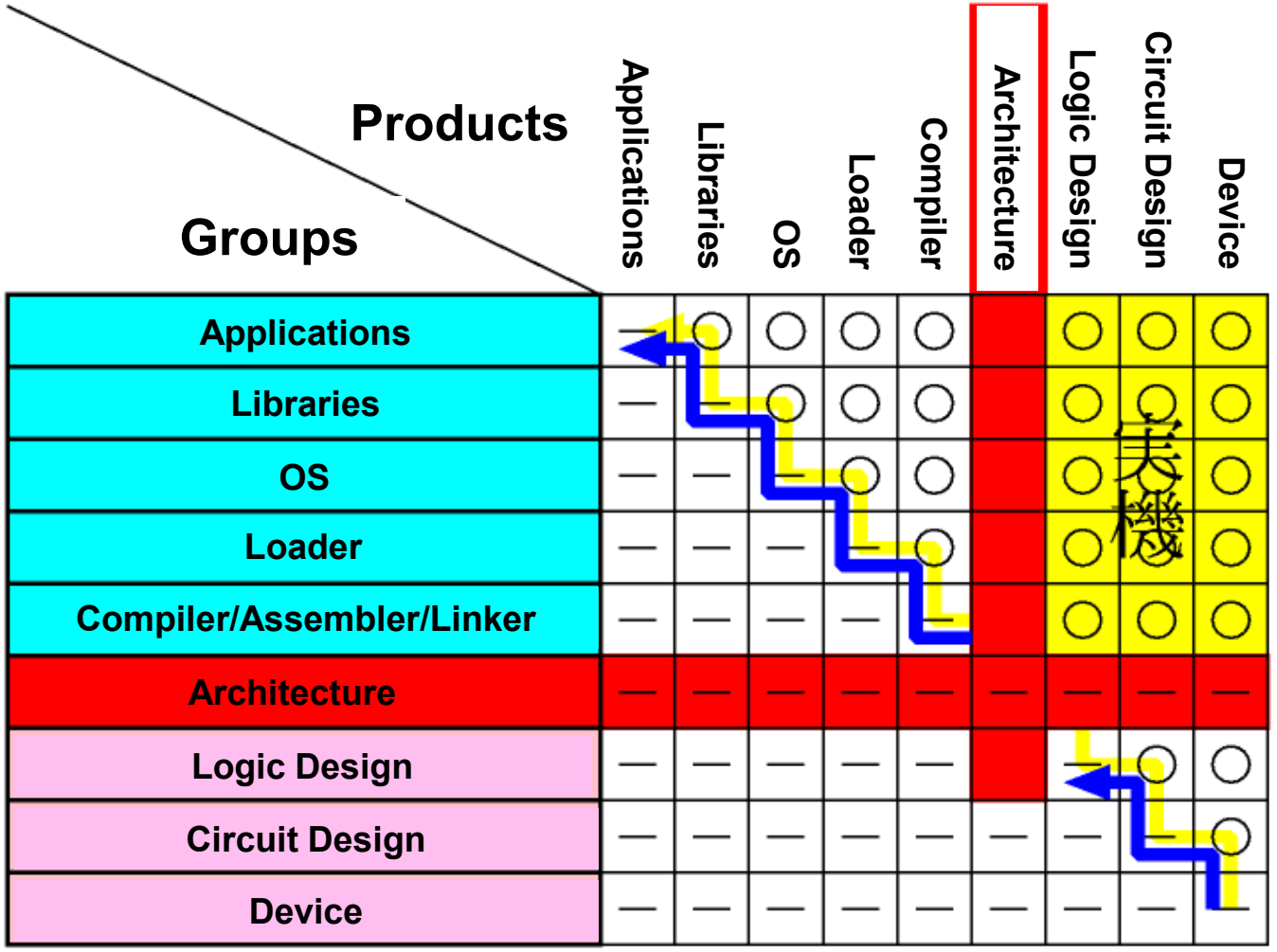
Simulation

for developing new computers

Who can start design of new computers ? (1988-)

All become bottleneck !

- **Hardware/Software designers can start their job today ?**
- **At the beginning, all designers depend on each other.**



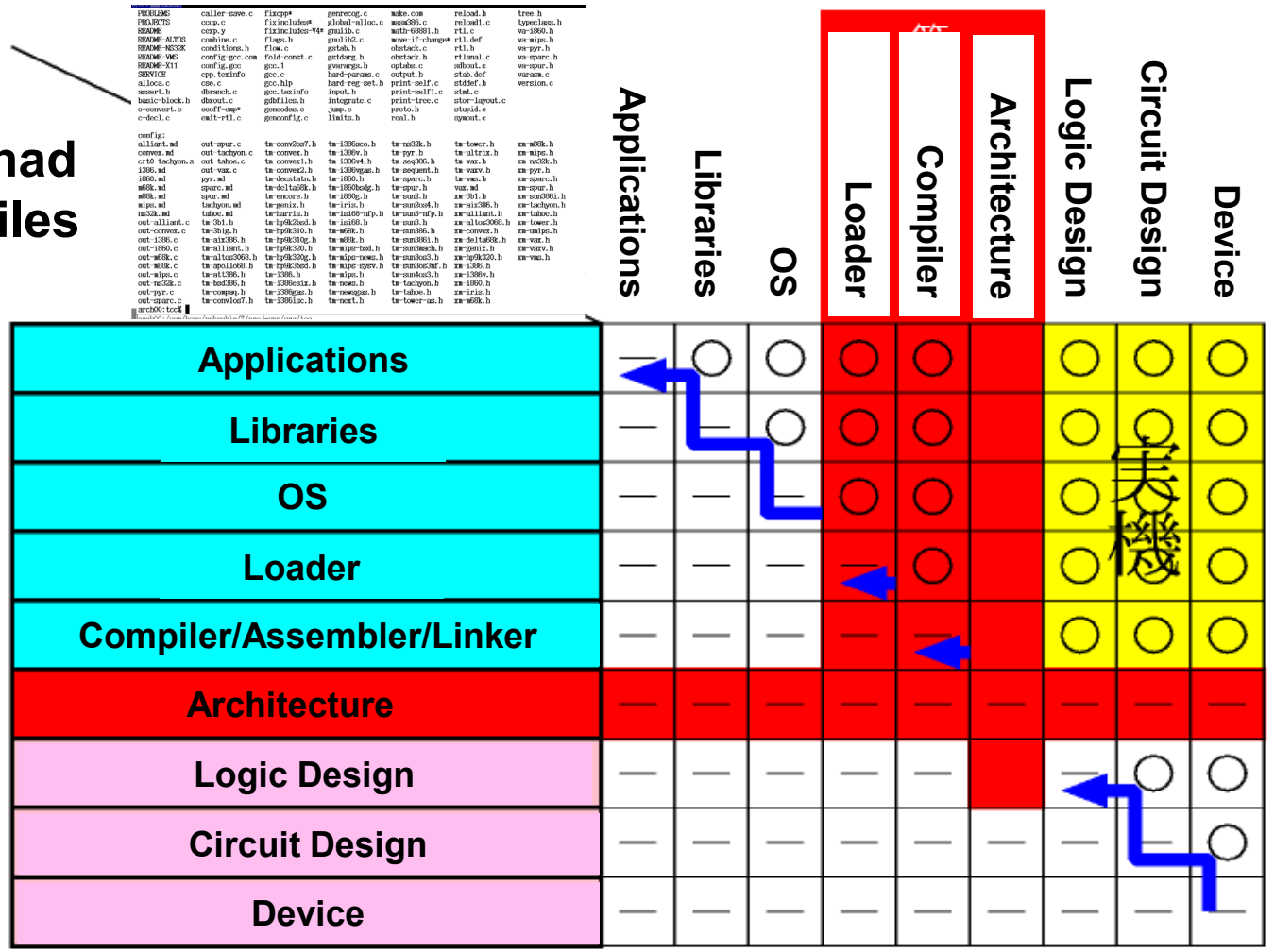
Who can start design of new computers ? (1988-)

Professional Programmers write codes at most 300 lines/month.

Architect should write basic tools at least 10000 lines/month.

All products should be ready before real machine comes.

Gcc-1.40 had only 250 files

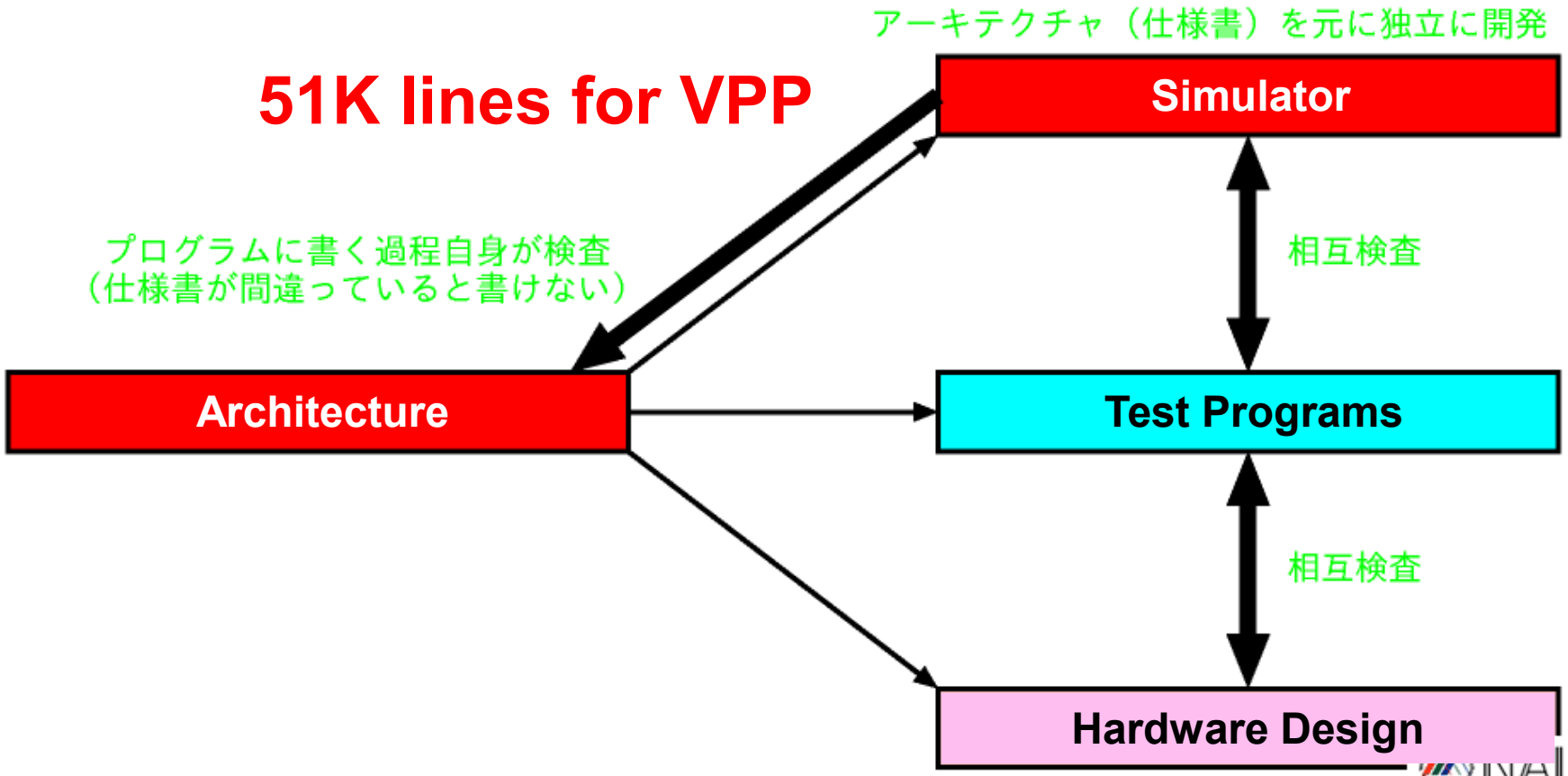


8

NAIST

Simulator is the Key for Bug-Free Development

- Writing Simulators can detect bugs of Documents
- All engineers design their products based on Documents
- Simulator and Test Programs detect bugs each other
- Hardware Design and Test Programs detect bugs each other
- High quality Simulator can detect most bugs of Software



Who needs simulators ?

- **Architects** want to verify Architecture Manuals.
  - Enough features ?
  - Consistency ?

- **Hardware Engineers** need bug-free Test Programs.

- **System Software Engineers** need to debug before new Hardware comes.
  - OS and Drivers
  - Compiler and Libraries

- **Application Engineers** need to improve Performance.
  - Cache behavior ?
  - Instruction MIX ?
  - Hidden Bottlenecks ?

Class of Simulators

- **Trace Based Simulator**
  - Can simulate cache behavior using address trace
  - Ultra-High-speed, timing is not exact

- **Instruction Simulator**
  - Can execute any software
  - High-speed, insufficient for performance estimation

- **Cycle Accurate Simulator**
  - Can estimate overheads including cache-miss
  - Middle-speed, insufficient for hardware design

- **Register Accurate Simulator**
  - Can estimate number of clocks and amount of hardware
  - Low-speed, but faster than HDL Simulators

Sub Class of Simulators

- System Simulator
  - Has full features in Architecture Manual
  - Can run OS but should boot OS before Applications
  - Too heavy for evaluating Applications

- Application Simulator
  - Has features only for User Programs
  - Can execute Applications directly
  - System-Calls are executed by Host-System-Calls
  - Can not evaluate OS overheads
  - Can not handle Dynamic Linking
  - Can not use Communication Libraries (ex. OpenMP)

Getting Basic Tools for Cross Compiling

- Binutils-2.18
  - as,ld,objdump
- Gcc-4.1.2
  - gcc,g++
- Newlib-1.18.0
  - libc,libm,libgcc
- _start.o ⇒
  - Init %sp

```
                .section ".text"
                .align      4
                .global _start
_start:
                clr         %fp
                clr         %g4
                sethi       %hi(0x00001008),%o0
                ld          [%o0+%lo(0x00001008)],%sp
                andcc       %sp, 1, %g5
                bnz,a       .LHaveBias
                mov         2047, %g5
.LHaveBias:
                add         %sp, %g5, %sp
                sub         %sp, %g5, %sp
                sethi       %hi(0x0000100c),%o0
                ld          [%o0+%lo(0x0000100c)],%o0
                sethi       %hi(0x00001010),%o1
                or          %o1,%lo(0x00001010),%o1
                call        main,0
                nop
abort: exit: _exit:
```

Making Executable Files

- Be sure how to start main() !
- _map: SECTIONS {.text 0x00010000 : { *(.text) }}

```
GCC     =       sparc64-elf-gcc -O3 -floop-optimize2 -funroll-loops
CFLAGS =        -I.
                -I/home/nakashim/proj-sap/sparc64-elf/include
                -I/usr/X11R6/include
LD      =       sparc64-elf-ld –static
                -M /home/nakashim/proj-sap/lib/ssim64-lib/_map
                -noinhibit-exec
                /home/nakashim/proj-sap/lib/ssim64-lib/_start.o
                -L/home/nakashim/proj-sap/lib/gcc/sparc64-elf/4.1.2/
                -L/home/nakashim/proj-sap/sparc64-elf/lib

test000:        test000.c
                $(GCC) -c test000.c
                $(LD) test000.o –lgcc –lm -lc -o test000
```

How to do with System-Calls

If you don't run OS, no need to implement priviledged instructions. Instead, Simulator should execute System-Calls.

```
000000000001168c <printf>:
 1168c: 9de3bf30  save  %sp, -208, %sp
 11690: f277a887  stx  %i1, [ %fp + 0x887 ]
 11694: 03000463  sethi  %hi(0x118c00), %g1
 11698: 82004004  add  %g1, %g4, %g1
 1169c: f477a88f  stx  %i2, [ %fp + 0x88f ]
          :
          :
000000000001394c <_write_r>:
 1394c: 82102004  mov  4, %g1
 13950: 9a100008  mov  %o0, %o5
 13954: 90100009  mov  %o1, %o0
 13958: 9210000a  mov  %o2, %o1
 1395c: 9410000b  mov  %o3, %o2
 13960: 9610000c  mov  %o4, %o3
 13964: 91d02008  ta  8
 13968: 1a800005  bcc  1397c <noerr>
 1396c: 0300004e  sethi  %hi(0x13800), %g1
```

```
#define SYS_exit     1
#define SYS_read     3
#define SYS_write    4
#define SYS_open     5
#define SYS_close    6
#define SYS_creat    8 N.A.
#define SYS_link     9 N.A.
#define SYS_unlink  10
#define SYS_chdir   12 N.A.
#define SYS_time    13
#define SYS_mknod   14 N.A.
#define SYS_chmod   15 N.A.
#define SYS_chown   16 N.A.
#define SYS_brk     17
#define SYS_stat    18
#define SYS_lseek   19
#define …
```

Instruction Level Simulators for Validating Tools

After finishing executable files, what you should do is just executing instructions from the beginning step by step.

```
union insn {
  struct base {
    Uint   sls    : 1;
    Uint   opcd   : 4;
    Uint   type1  : 1;
    Uint   type0  : 2;
    Uint   cond   : 4;
  } base;
  …
while (p[pid].status == STATUS_NORMAL) {
  p[pid].pc = p[pid].npc;
  p[pid].npc = p[pid].pc + 4;
  mp = &mem[p[pid].pc];
  insn.raw = *(mp+3)<<24|*(mp+2)<<16|*(mp+1)<<8|*(mp+0);
  insn_decode(pid, insn, &rtl_decode);
  insn_exec(pid, &rtl_decode);
  p[pid].step++;
}
```

ARM-V4	**2.0K lines**
SH2	**2.6K lines**
SPARC-V9	**3.5K lines**

3days are enough to complete ☺

Summary: Main Flow of Simulation

```
p[0].pc     = read_elf(prog);
p[0].npc    = p[0].pc + 4;
p[0].status = STATUS_NORMAL;
while (alive) {
  alive = 0;
  for (pid=0; pid<PENUM; tid++) sim_core(pid);
  for (pid=0; pid<PENUM; pid++) {
    switch (p[pid].status) {
    case STATUS_EXCSVC:
      switch (exec_svc(pid)) {
      case 0: p[pid].status = STATUS_NORMAL; break;
      case 2: printf("%d:EXCSVC SPARC normal end¥n", pid);
              p[pid].status = STATUS_COMPLETE; break;
      }
    case STATUS_NORMAL: case STATUS_IQWAIT: case STATUS_IMWAIT: case STATUS_CHECK:
    case STATUS_EXEC:   case STATUS_FLUSH:  case STATUS_WOVFLOW:case STATUS_WUDFLOW:
      alive++; break;
    }
  }
}
```

```
exec_svc(pid)
{
  switch (val) {
  case 1: /* exit */
    return(2);
  case 4: /* write */
    getmp(grr(pid, 0, 0, 9), &mp2);
    grw(pid,0,0,8,(Sll)write((int)grr(pid,0,0,8),mp2,(int)grr(pid,0,0,10)));
    return(0);
  }
}
```

```
sim_core(pid)
{
  p[pid].opc = p[pid].pc;
  p[pid].pc  = p[pid].npc;
  p[pid].npc = p[pid].pc+4;
  o_ifetch(pid,p[pid].opc);
  (p[pid].sop->func)(pid,p[pid].ib.insn,'c');
  (p[pid].sop->func)(pid,p[pid].ib.insn,'x');
}
```

Summary: Load Instruction

```
int i_ldx(tid, i, stage) Uint tid, i, stage;
{ int cid = tid % CORENUM;
  Ull  S1, S2, D;

  switch (stage) {
  case 'c': /* l1rq-load */
    CHECK_REGDEPEND(1, i>>14&31);
    if (i&0x2000)
      return (0); /* ok */
    CHECK_REGDEPEND(1, i&31);
    return (0); /* ok */
  case 'x':
    S1 = grr(tid, 0, 0, i>>14&31);
    S2 = (i&0x2000)?(Sll)((int)(i<<19)>>19):grr(tid, 0, 0, i&31);
    c[cid].mpipe[0].v = 1;
    c[cid].mpipe[0].tid = tid;
    c[cid].mpipe[0].drt = 1;
    c[cid].mpipe[0].drp = i>>25&31;
    switch (o_ldst(tid, 8, 11, S1+S2, 0xffffffffffffffffLL, &D, 0, 0, 1, i>>25&31)) {
    case 0: /* normal end */
      grw(tid, 0, 0, i>>25&31, D);
      return (0);
    case 1: /* mem_wait(queue-full) */
      return (1);
    case 2: /* mem_wait(queue-ok) */
      return (2);
    case 3: /* error */
      return (3);
    }
  }
}
```

Summary: Adds Instruction

```
i_addcc(tid, i, stage) Uint tid, i, stage;
{
  Ull  S1, S2, D;
  Uint s1, s2, d;
  Uchar N, Z, V, Ca, n, z, v, ca;

  switch (stage) {
  case 'c':
    CHECK_REGDEPEND(1, i>>14&31);
    if (i&0x2000)
      return (0); /* ok */
    CHECK_REGDEPEND(1, i&31);
    return (0); /* ok */
  case 'x':
    S1 = grr(tid, 0, 0, i>>14&31);
    S2 = (i&0x2000)?(Sll)((int)(i<<19)>>19):grr(tid, 0, 0, i&31);
    D = S1 + S2;
    grw(tid, 0, 0, i>>25&31, D);
    s1 = S1; s2 = S2; d = D;
    n = d>>31;
    z = d==0;
    v = (s1>>31&&s2>>31&&!(d>>31)) || (!(s1>>31)&&!(s2>>31)&&d>>31);
    ca= (s1>>31&&s2>>31) || (!(d>>31)&&(s1>>31||s2>>31));
    N = D>>63;
    Z = D==0;
    V = (S1>>63&&S2>>63&&!(D>>63)) || (!(S1>>63)&&!(S2>>63)&&D>>63);
    Ca= (S1>>63&&S2>>63) || (!(D>>63)&&(S1>>63||S2>>63));
    ccw(tid, 0x000000ff, (N<<7)|(Z<<6)|(V<<5)|(Ca<<4)|(n<<3)|(z<<2)|(v<<1)|ca);
    return (0);
  }
}
```

Cycle Accurate Simulators for Performance Evaluation

```
switch (p[pid].status) {
   case STATUS_NORMAL:
     p[pid].opc = p[pid].pc;
     p[pid].pc  = p[pid].npc;
     p[pid].npc = p[pid].pc + 4;

   case STATUS_IQWAIT:
     switch (o_ifetch(pid, p[pid].opc)) {
     case 0: /* normal_end */
     case 2: /* mem_wait(queue-ok) */
       break;
     case 1: /* mem_wait(queue-full) */
       p[pid].status = STATUS_IQWAIT;
       p[pid].i1_wait_cycle++;
       continue;
     case 3: /* error */
       p[pid].status = STATUS_STOP;
       continue;
     }
   case STATUS_IMWAIT:
     if (!p[pid].ib.v) {
       p[pid].status = STATUS_IMWAIT;
       p[pid].i1_wait_cycle++;
       continue;
```

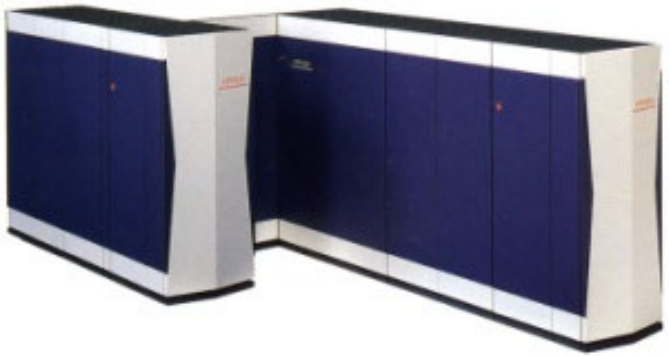
Each module works equivalent to 1 cycle of hardware.

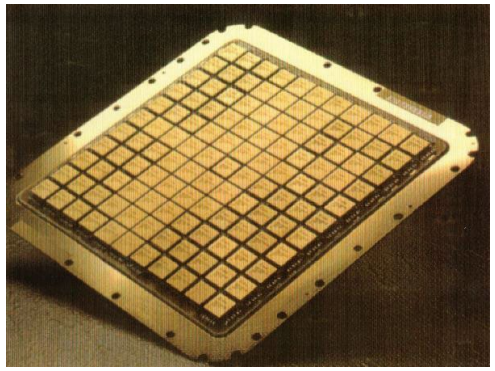
Increase “cycles” every time.

Increase “x_wait_cycle” for each overhead.

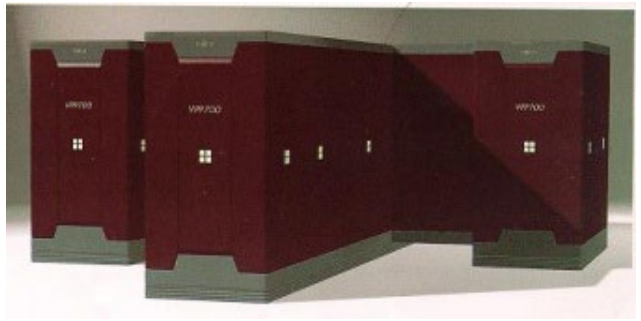
New Vector Supercomputers were Designed and Tested w/ Simulators (1988-)

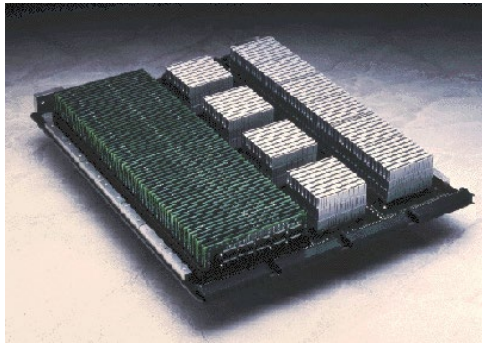
I was
24 years old

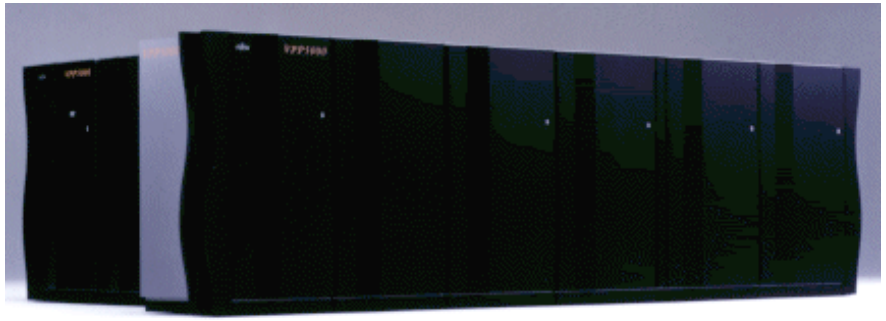


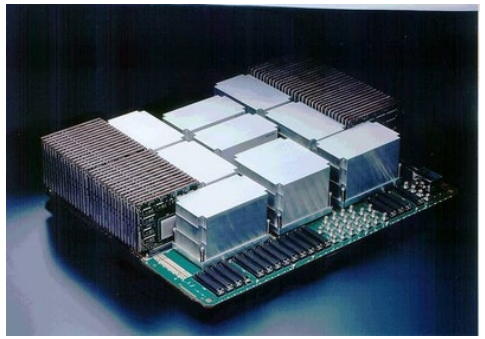


≃1M$
x220



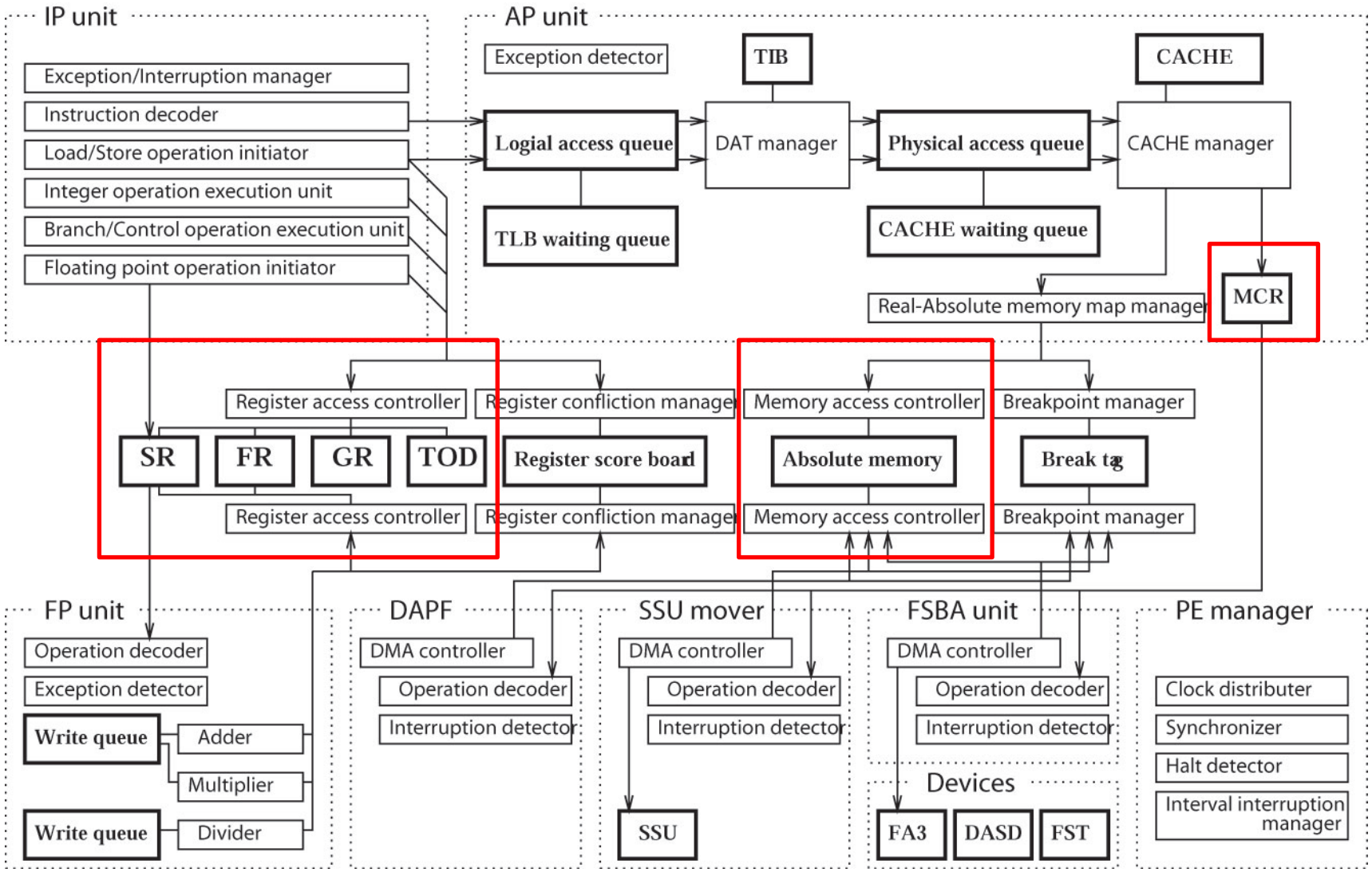






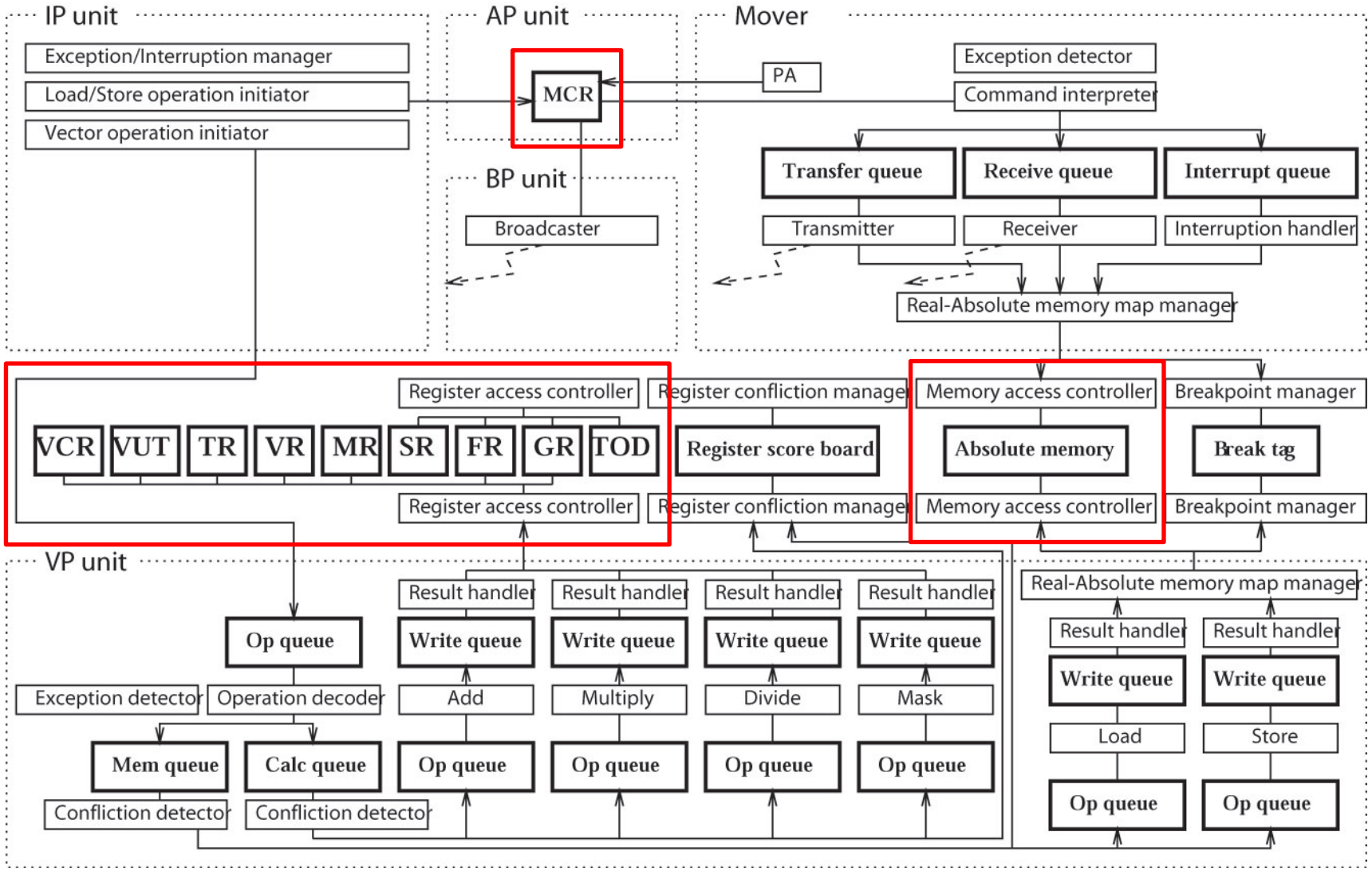
VPP Simulator（VLIW and many I/Os）

Simulation of Registers, Cache and Memory is important.



VPP Simulator（Vectors, Data Mover, Barrier）

Simulation of Registers, Cache and Memory is important.



VPP Simulator（Customizable for All Engineers）

- Application Simulator
  - Single Scalar
  - Single S+Vector
- System Simulator
  - Single Scalar
  - Parallel S+V

```
sim_pe(pe)
{
    sim_ip(pe);
    sim_ap(pe);
    sim_fp(pe);
#ifdef VCP
    sim_vp(pe);
#endif
#ifdef MOVER
    sim_mv(pe);
#endif
#ifdef BARRIER
    sim_bp(pe);
#endif
#ifdef SSUMOVER
    sim_ssumv(pe);
#endif
#ifdef GSIGP
    sim_gsigp(pe);
#endif
#ifdef FSBA
    sim_fsba(pe);
#endif
#ifdef DAPF
    sim_dapf(pe);
#endif
}
```

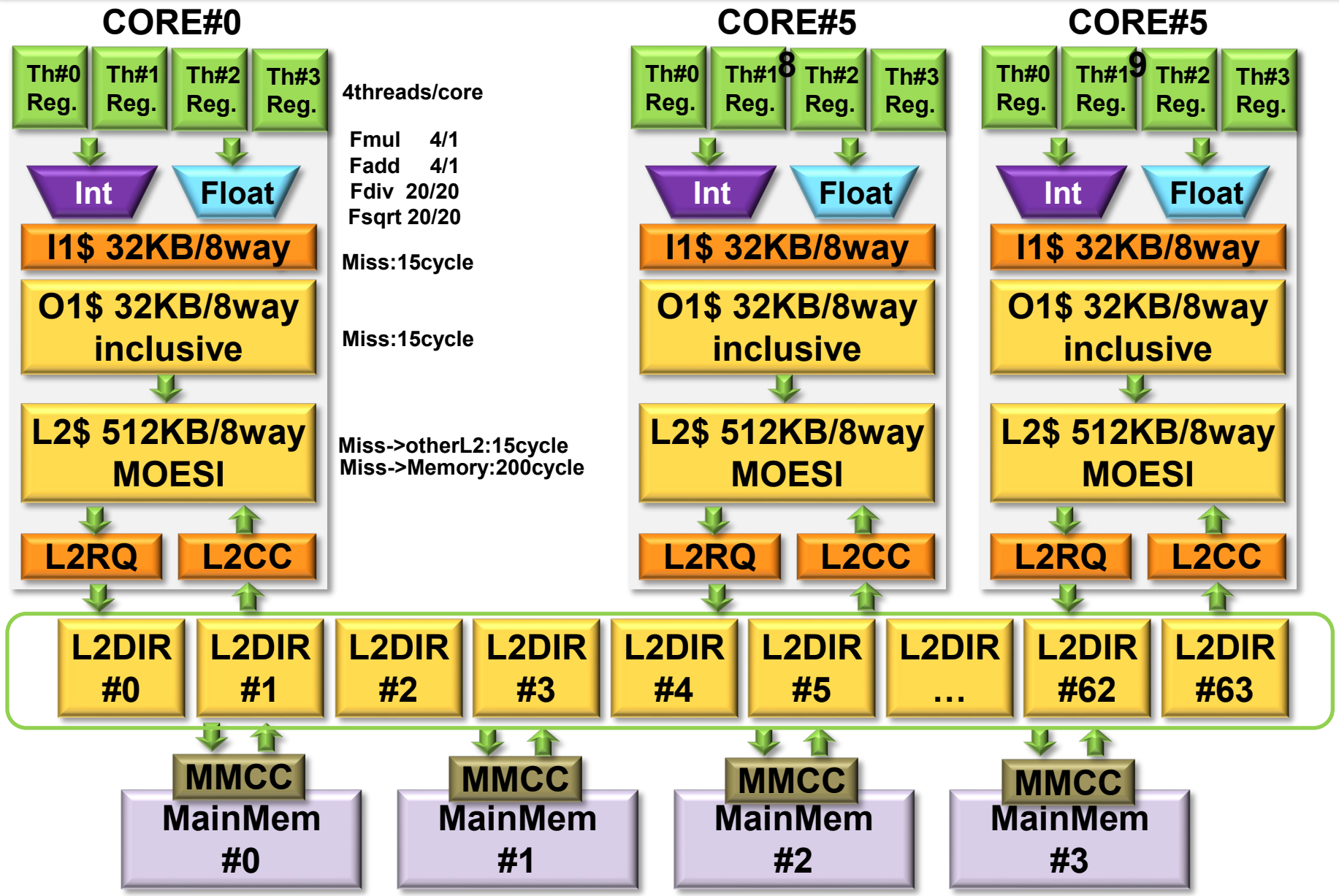
プログラム名	tim	tim+v	kic	kiv	kim
PCPU(浮動小数点演算器含む)	1台	1台	1台	1台	可変
DAT(内蔵 TLB 含む)	○	○	○	○	○
内蔵キャッシュ	○	○	○	○	○
プログラム割込み	○	○	○	○	○
SVC 割込み	○	○	○	○	○
FWC 割込み	○	○	○	○	○
ソフトウェアレベル制御割込み	○	○	○	○	○
手動外部割込み (si コマンド)	○	○	○	○	○
論理 DASD	○	○	○	○	○
実メモリ写像機構 (sm コマンド)	○	○	○	○	○
絶対メモリ容量 (起動時指定省略時)	8MB	8MB	8MB	8MB	8MB
MSMAP 初期値 (起動時指定省略時)	×	×	8MB	8MB	8MB
保護 HSA 領域 (メモリ先頭 512KB)	×	×	○	○	○
HSA 保護機構 (VCP,MOVER,FSBA)	×	×	○	○	○
MCR 空間	×	×	○	○	○
インターバル割込み	×	×	○	○	○
WDT 割込み	×	×	○	○	○
ベクトルコプロセッサ	×	○	×	○	○
MOVER	×	×	○	○	○
ハードウェアバリア機構	×	×	○	○	○
SSUMOVER	×	×	○	×	○
GSIGP HANDLER	×	×	○	×	○
FSBA(PCPU 毎)	×	×	4台	×	4台
FST(FSBA 毎)	×	×	1台	×	1台
物理 DASD	×	×	○	×	○
DAPF	×	×	○	○	○
論理コンソール	×	×	○	○	○
ファームウェアデバッグ機能	×	×	○	○	○
SAPC	×	×	○	○	○
デバッガインタフェース	×	×	○	○	○

A Performance Simulator for Black Boxed System

**If you want to tune software on black boxed system,
you have to model inside of the system.**

**Details of internal structure are unknown,
but we can imagine inside of the architecture.**

MIC Simulation Model (Cycle Accurate)



Parameters in ssim.h

- /home/nakashim/proj-sap.cent/src/ssim9-20131113/ssim.h

```
#define DDRMINADDR     0x0000000000000004LL /* minimum address is 4     */
#define DDRHDRADDR     0x0000000000001000LL
#define DDRALOCLIMIT   0x000000002fffff00LL /* data_size/all:   800MB   */
#define DDRSTACKINIT   0x000000003fffff00LL /* stack_size/thread: 1MB   */
#define DDRSIZE        0x0000000040000000LL /* DDR size            1GB   */

#define NWINDOWS       8                    /* default size is    8      */
#define I1WAYS         8                    /* I1 cache-ways      8      */
#define I1WSIZE        4096                 /* I1 way-size        4KB    */
#define O1WAYS         8                    /* O1 cache-ways      8      */
#define O1WSIZE        4096                 /* O1 way-size        4KB    */
#define L2WAYS         8                    /* L2 cache-ways      8      */
#define L2WSIZE        65536                /* L2 size/bank       64KB   */
#define LINESIZE       64                   /* L1/L2 Line-size    64B    */

#define MPIPE          3
#define FPIPE          4
#define IMULDELAY      8
#define IDIVDELAY      20
#define FDIVDELAY      20
#define L2DELAY        15                   /* L2 penalty 15cycle */
#define CCDELAY        15                   /* CC penalty 15cycle*/
#define MMDELAY        200 /*800*/         /* MM miss-penalty 800cycle*/
```

Understanding test000.c

➢ /home/nakashim/proj-sap.cent/sample/test/test000.c

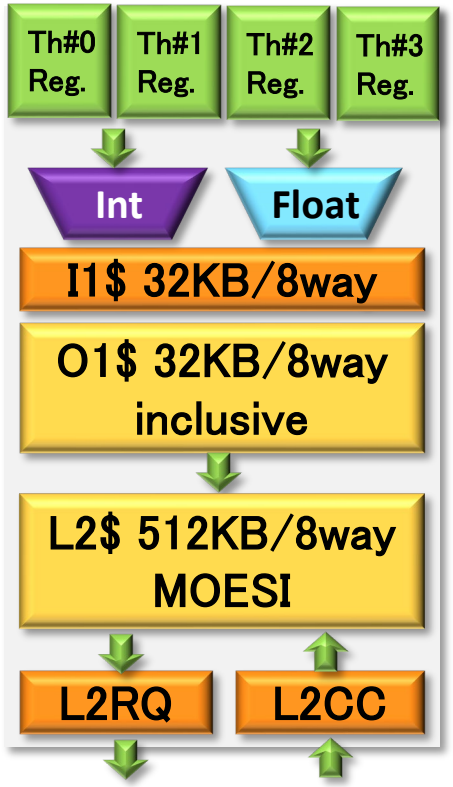
```
#include <stdio.h>
#include <math.h>
#define THNUM 240
#define SIZEY 480
#define SIZEX 256
double A[SIZEY][SIZEX]; __attribute__((aligned(8192)));
double B[SIZEY][SIZEX]; __attribute__((aligned(8192)));
double C[SIZEY][SIZEX]; __attribute__((aligned(8192)));

main(argc, argv) int argc; char **argv;
{
 int pid, i, j, k;
 _barrier(1);                                  ! All threads start execution from  here
 if ((pid=_getpid()) == 0) {                   ! Get my thread ID and save to “pid”
  printf("%s start¥n", argv[0]);
  fflush(stdout);
 }
 _barrier(0);                                  ! Wait for other threads
 if (pid==0) _getpa();                         ! Dislpay status of all threads and reset performance counters
 for (i=SIZEY/THNUM*pid; i<SIZEY/THNUM*(pid+1); i++) {
  for (j=0; j<SIZEX; j++) {
   A[i][j] = B[i][j] * C[i][j];                ! Kernel
   /*A[i][j] = sqrt(B[i][j]);*/
  }
 }
 _barrier(1);                                  ! Wait fot other threads
 if (pid == 0) {
  _getpa();                                    ! Dislpay status of all threads and reset performance counters
  printf("%s end¥n", argv[0]);
  fflush(stdout);
 }
 else
  _halt();
}
```

Structure of Thread

```
struct t {
  int      status;          /* processor status          */
  int      barrier;         /* for _barrier()            */
  Ull      steps;           /* original steps            */
  Ull      cycle;           /* real cycles               */
  Ull      i1_hit;          ! Instruction L1$ counters
  Ull      i1_miss;
  Ull      o1_hit;          ! Data L1$ counters
  Ull      o1_miss;
  Ull      l2_hit;          ! Local L2$ counters
  Ull      l2_miss;
  Ull      g2_hit;          ! Global L2$ counters
  Ull      g2_miss;

  Ull insn_count[4][64];    ! Instruction counters
  Ull insn_float[512];      ! Floating Point Instruction counters

  Ull      pc;              /* PC                        */
  Ull      *r[32];          /* regmap                    */
  Ull      g[8];            /* global                    */
  Ull      i[8*NWINDOWS];   /* in (o[i-1])               */
  Ull      l[8*NWINDOWS];   /* local                     */
  Ull      y;               /* Y reg 31-0:valid          */
  Uint     c;               /* 0-16:ELGU(fcc0,1,2,3) 24-31:NZVCnzvc */
  Uint     f[64];           /* float reg                 */
} t[THNUM];                 ! Structure of each Thread
```

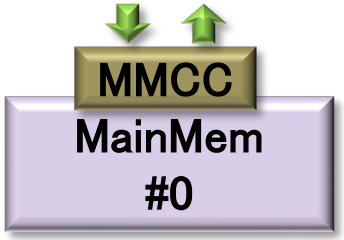
Each module corresponds to a function

```
main(argc, argv) int argc; char **argv;
{
  while (alive) {
    alive = 0;
    for (tid=0; tid<THNUM; tid++) sim_core    ((tid<<16)|STHREAD_TICKS);
    for (did=0; did<L2DIR; did++) sim_mreq    ((did<<16)|STHREAD_TICKS);
    for (mid=0; mid<MMNUM; mid++) sim_cluster((mid<<16)|STHREAD_TICKS);
  }
  printpa();
}
```



sim_core()

sim_mreq()



sim_cluster()

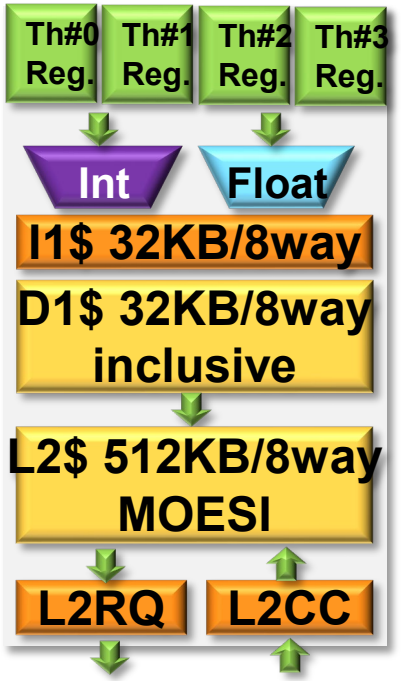
Structure of Core

```
struct c {
  struct i1_tag  {} i1_tag[I1WSIZE/LINESIZE][I1WAYS];
  struct i1_line {} i1_line[I1WSIZE/LINESIZE][I1WAYS];

  struct ib {} ib;
  struct mpipe {} mpipe[MPIPE]; /* for cycle accurate*/
  struct imulq {} imulq;        /* non-pipeline */
  struct idivq {} idivq;        /* non-pipeline */
  struct fpipe {} fpipe[FPIPE]; /* for cycle accurate*/
  struct fdivq {} fdivq;        /* non-pipeline */

  struct o1_tag {
    Uint   v     :  1;        /* valid                */
    Uint   lru   :  8;        /* lru counter          */
    Uint   dirty :  1;        /* dirty-flag           */
    Uint   share :  1;        /* shared-flag          */
    Uint   la    : 32;        /* logical addr         */
  } o1_tag[O1WSIZE/LINESIZE][O1WAYS];
  struct o1_line {} o1_line[O1WSIZE/LINESIZE][O1WAYS];
  struct l2_tag {
    Uint   v     :  1;        /* valid       */
    Uint   lru   :  8;        /* lru counter */
    Uint   dirty :  1;        /* dirty       */
    Uint   share :  1;        /* shared-flag */
    Uint   la    : 32;        /* logical addr */
  } l2_tag[L2WSIZE/LINESIZE][L2WAYS];
  struct l2_line {} l2_line[L2WSIZE/LINESIZE][L2WAYS];
  struct l1rq {} l1rq;
  struct l2rq {} l2rq;
  struct l2cc {} l2cc[L2DIR];
} c[CORENUM];                 ! Structure of Physical cores
```

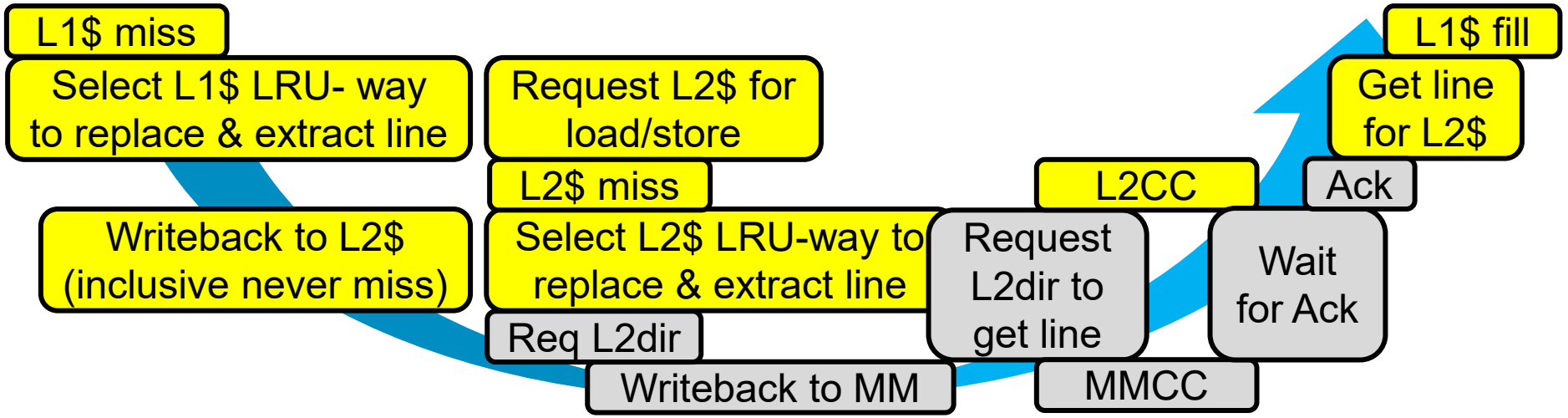
Structure of L2dir and Memory

```
struct d {
  Uchar l2rq_bitmap[CORENUM];
  Ull   l2rq_lock: CORENUM;
  Uint  root_cid;
  Uint  l2d_state;
  struct l2dir {
    Ull  l2dir_v : CORENUM; /* 1 0 0  Exclusive-clean */
    Ull  l2dir_d : CORENUM; /* 1 0 1  Shared clean */
    Ull  l2dir_s : CORENUM; /* 1 1 0  Mod */
                            /* 1 1 1  Shared Owned(mod) */
  } l2dir[DDRSIZE/L2DIR/LINESIZE];
  Ull   l2cc_req_bitmap : CORENUM;
  Uchar l2cc_ack_bitmap  [CORENUM];
  Ull   mmcc_req_bitmap : MMNUM;
  Uchar mmcc_ack_bitmap  [MMNUM];
} d[L2DIR];                ! Structure of L2DIRs

struct m {
  struct mmcc {
    Uint   v_stat     : 4;   /* stat 0:empty 1:busy 2:OP-ok 3:IF-ok */
    Uint   t          :10;   /* timer                          */
    Uint   rq         : 1;   /* 0:push, 1:pull */
    Ull    ADR;
    Ull    BUF[LINESIZE/8]; /* [LINESIZE/8] */
  } mmcc[L2DIR];
  Uchar ddr[DDRSIZE/MMNUM];
} m[MMNUM];                ! Structure of Main Memory
```

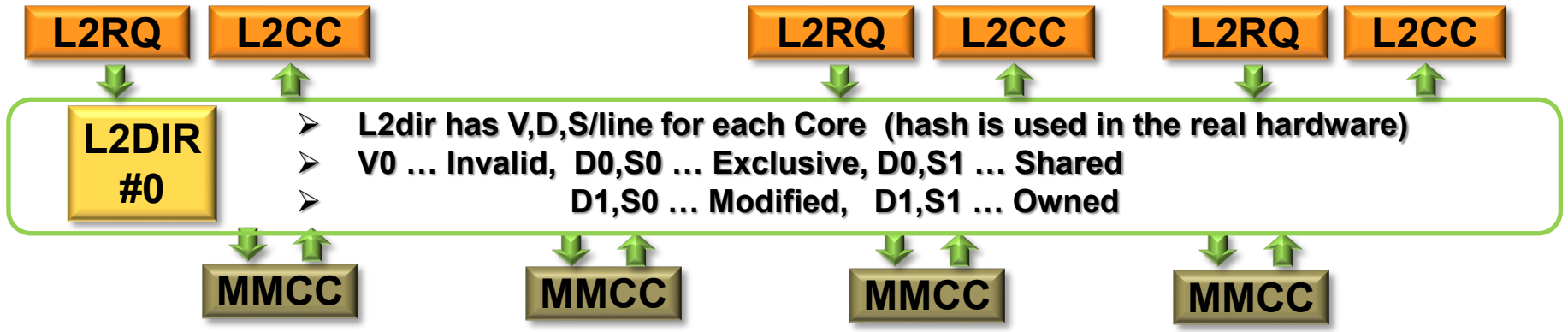
Sim_core.c



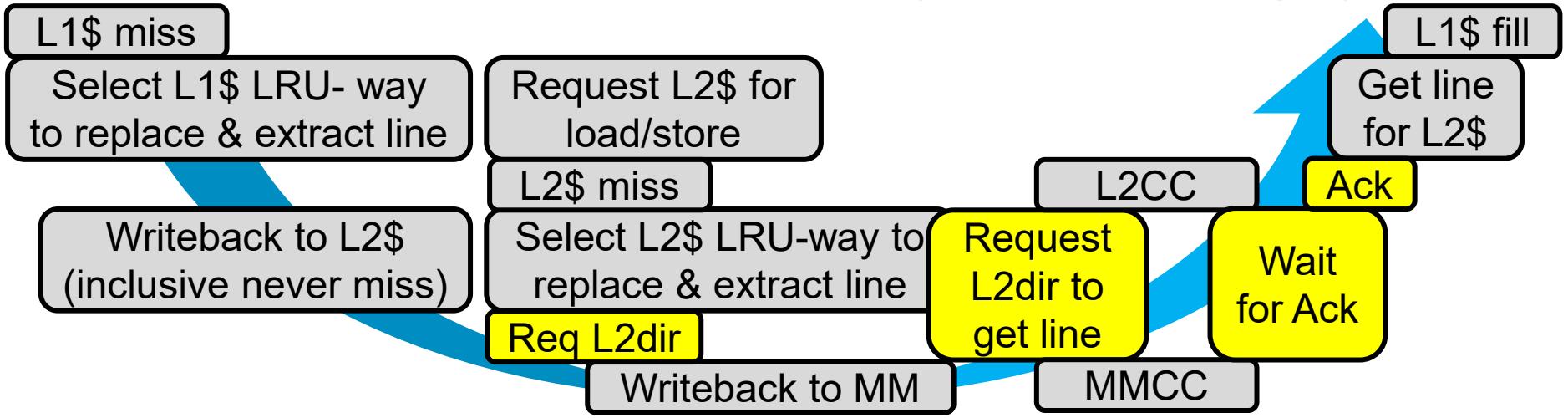
- I1$ is shared among thread and provides 1 instr./cycle (not 4 instr./cycle)
- D1$ and L2$ have Valid, Dirty, Shared bits/line
  - V0 … invalid -> miss
  - D0,S0 … load hit, store miss(update L2dir)
  - D1,S0 … load hit, store hit
  - D1,S1 … load hit, store miss(invalidate other$)
  - D0,S1 … load hit, store miss(invalidate other$)



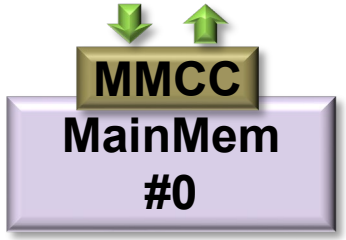
Sim_mreq.c



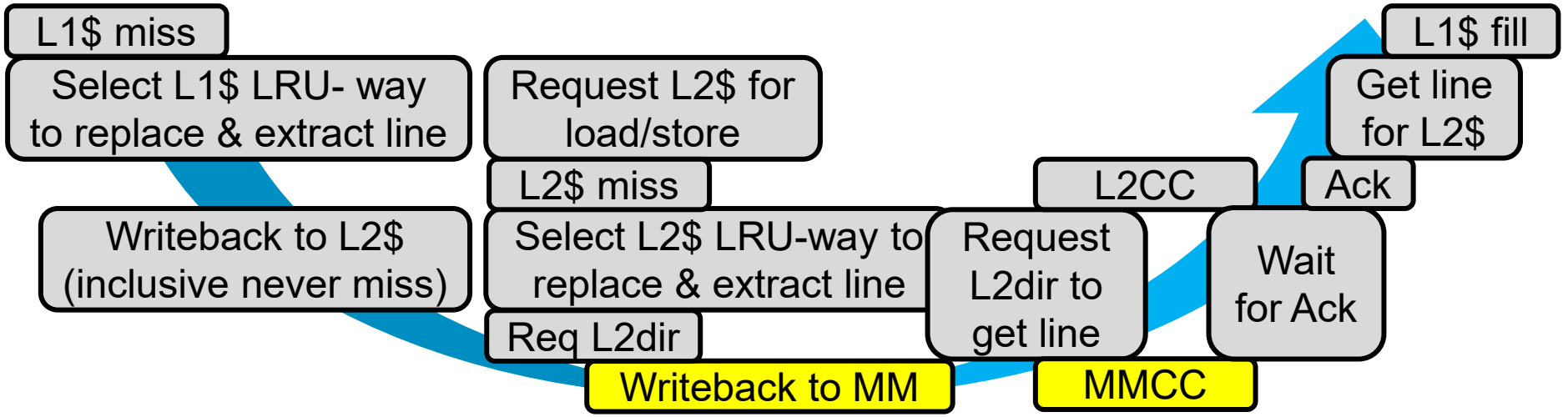
L2$ Invalidate Req:	Others: Exclusive/Shared > Invalid		Local: > Modified
MM Writeback Req:	Others: Exclusive/Shared > Invalid		Local: > Invalid
Read Req for load:	Home: > Owned	Others: no-change	Local: > Shared
	If no HOME, data should be loaded from MM (Shared can be invalidated anytime)		
Read Req for store:	Home: Copyback > Invalid	Others: > Invalid	Local: > Modified
	If no HOME, data should be loaded from MM (Shared can be invalidated anytime)		



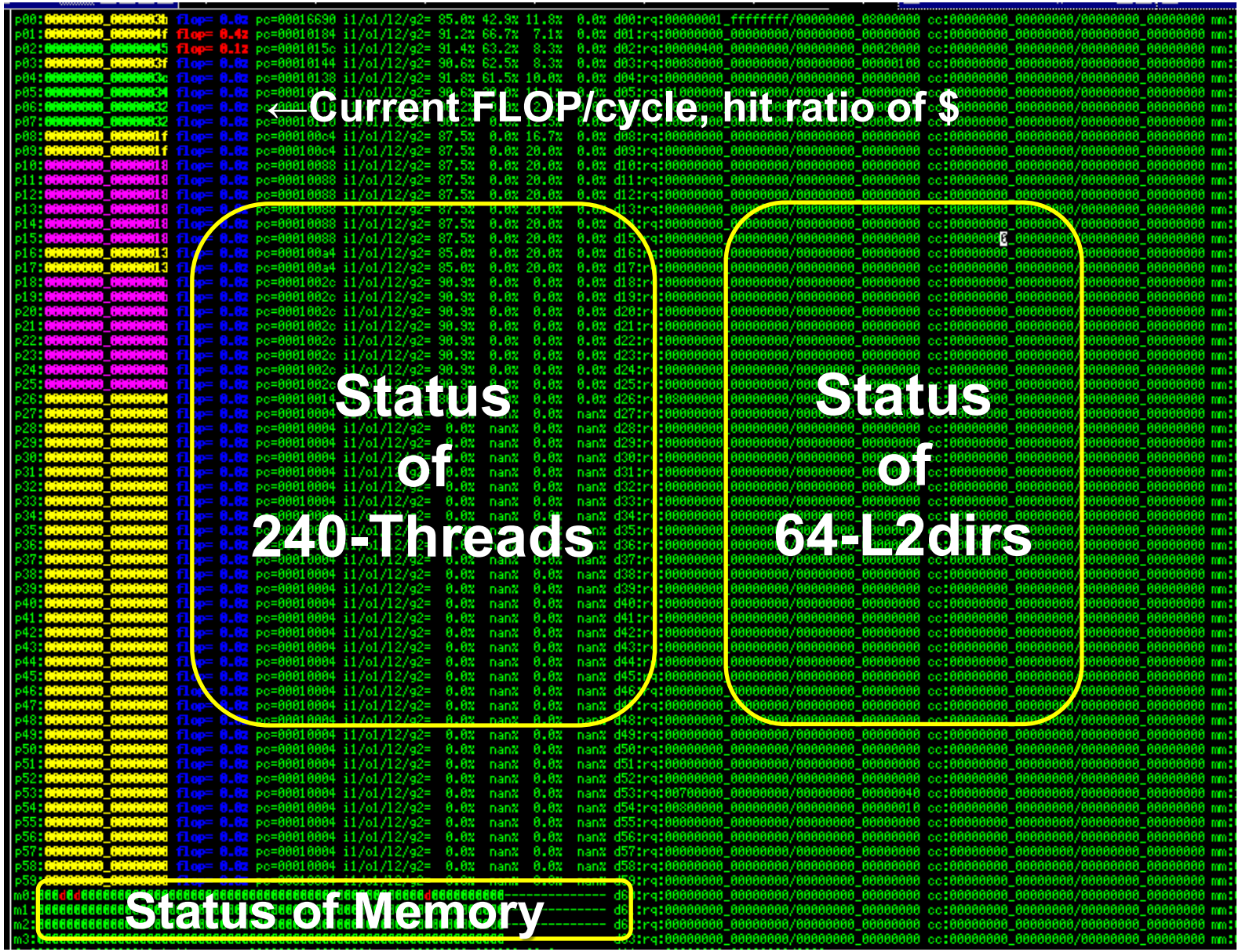
Sim_cluster.c



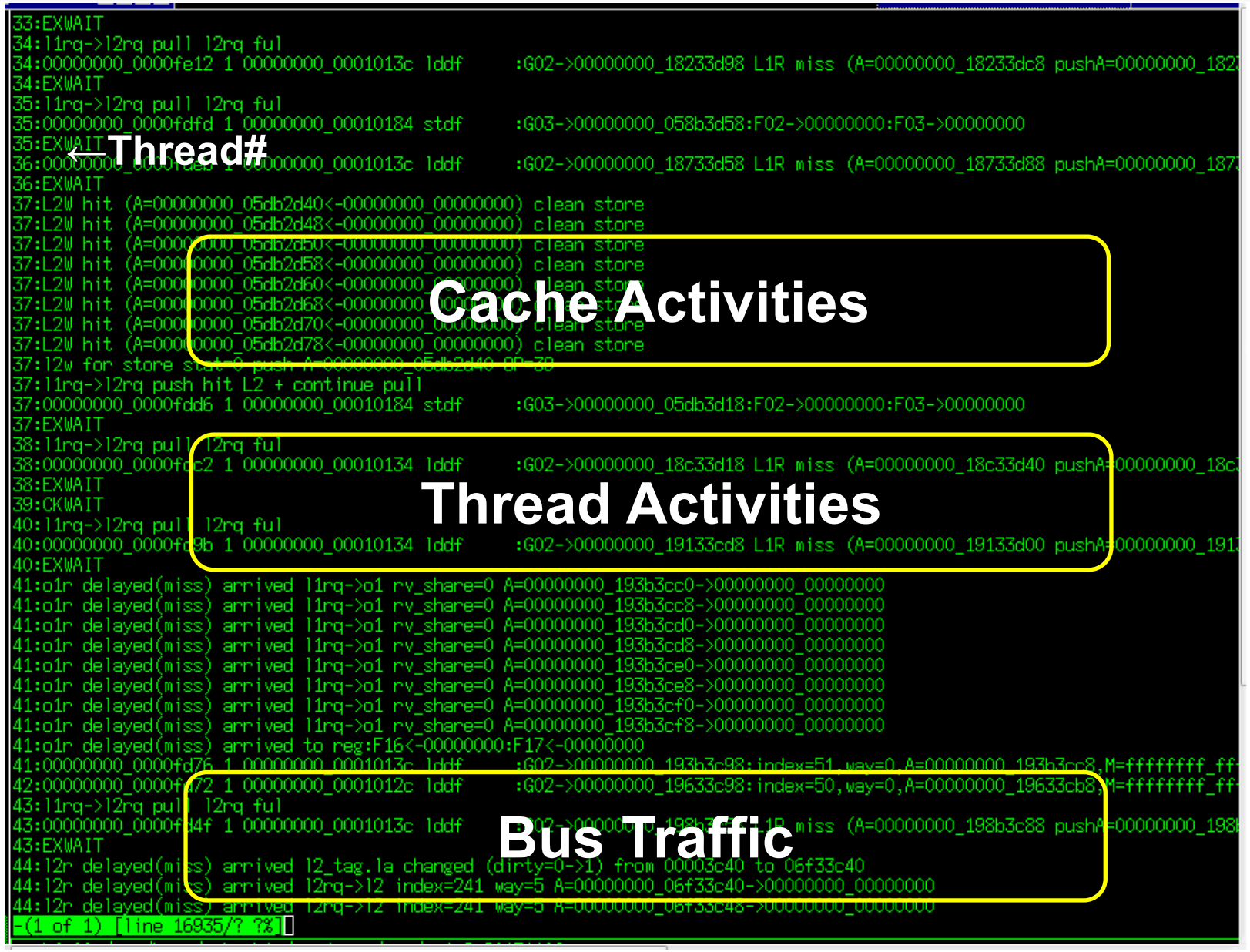
Simply gets the request from L2dir and sends ack to L2dir



Demo-mode


←Current FLOP/cycle, hit ratio of $
Status of 240-Threads
Status of 64-L2dirs
Status of Memory

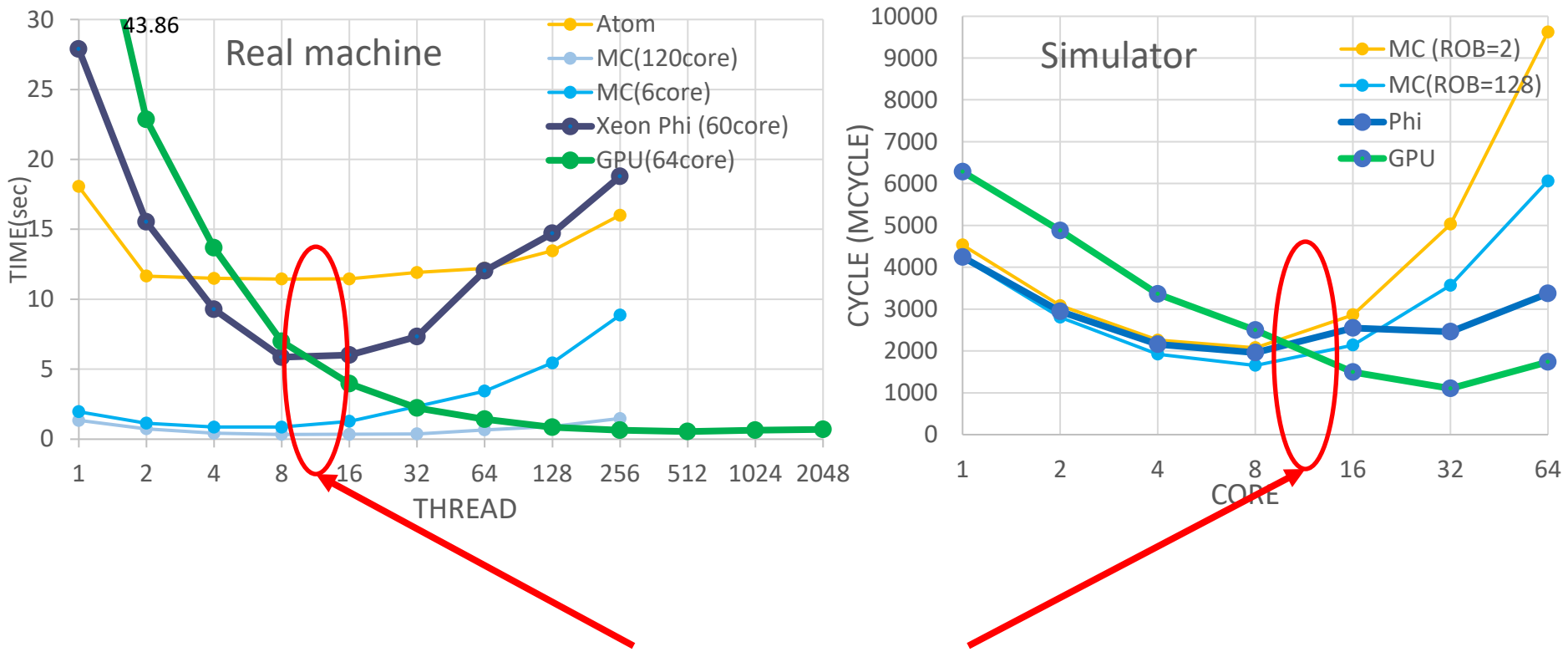
Detail-mode


33:EXWAIT
34:l1rq->l2rq pull l2rq ful
34:00000000_0000fe12 1 00000000_0001013c lddf :G02->00000000_18233d98 L1R miss (A=00000000_18233dc8 pushA=00000000_182
34:EXWAIT
35:l1rq->l2rq pull l2rq ful
35:00000000_0000fdfd 1 00000000_00010184 stdf :G03->00000000_058b3d58:F02->00000000:F03->00000000
35:EXWAIT
←Thread#
36:00000000_0000fdd8 1 00000000_0001013c lddf :G02->00000000_18733d58 L1R miss (A=00000000_18733d88 pushA=00000000_187
36:EXWAIT
37:L2W hit (A=00000000_05db2d40<-00000000_00000000) clean store
37:L2W hit (A=00000000_05db2d48<-00000000_00000000) clean store
37:L2W hit (A=00000000_05db2d50<-00000000_00000000) clean store
37:L2W hit (A=00000000_05db2d58<-00000000_00000000) clean store
37:L2W hit (A=00000000_05db2d60<-00000000_00000000) clean store
37:L2W hit (A=00000000_05db2d68<-00000000_00000000) clean store
37:L2W hit (A=00000000_05db2d70<-00000000_00000000) clean store
37:L2W hit (A=00000000_05db2d78<-00000000_00000000) clean store
37:l2w for store stdt 0 push A=00000000_05db2d40 BP=30
37:l1rq->l2rq push hit L2 + continue pull
37:00000000_0000fdd6 1 00000000_00010184 stdf :G03->00000000_05db3d18:F02->00000000:F03->00000000
37:EXWAIT
38:l1rq->l2rq pull l2rq ful
38:00000000_0000fdc2 1 00000000_00010134 lddf :G02->00000000_18c33d18 L1R miss (A=00000000_18c33d40 pushA=00000000_18c
38:EXWAIT
39:CKWAIT
40:l1rq->l2rq pull l2rq ful
40:00000000_0000fd9b 1 00000000_00010134 lddf :G02->00000000_19133cd8 L1R miss (A=00000000_19133d00 pushA=00000000_191
40:EXWAIT
41:o1r delayed(miss) arrived l1rq->o1 rv_share=0 A=00000000_193b3cc0->00000000_00000000
41:o1r delayed(miss) arrived l1rq->o1 rv_share=0 A=00000000_193b3cc8->00000000_00000000
41:o1r delayed(miss) arrived l1rq->o1 rv_share=0 A=00000000_193b3cd0->00000000_00000000
41:o1r delayed(miss) arrived l1rq->o1 rv_share=0 A=00000000_193b3cd8->00000000_00000000
41:o1r delayed(miss) arrived l1rq->o1 rv_share=0 A=00000000_193b3ce0->00000000_00000000
41:o1r delayed(miss) arrived l1rq->o1 rv_share=0 A=00000000_193b3ce8->00000000_00000000
41:o1r delayed(miss) arrived l1rq->o1 rv_share=0 A=00000000_193b3cf0->00000000_00000000
41:o1r delayed(miss) arrived l1rq->o1 rv_share=0 A=00000000_193b3cf8->00000000_00000000
41:o1r delayed(miss) arrived to reg:F16<-00000000:F17<-00000000
41:00000000_0000fd76 1 00000000_0001013c lddf :G02->00000000_193b3c98:index=51,way=0,A=00000000_193b3cc8,M=ffffffff_ff
42:00000000_0000fd72 1 00000000_0001012c lddf :G02->00000000_19633c98:index=50,way=0,A=00000000_19633cb8,M=ffffffff_ff
43:l1rq->l2rq pull l2rq ful
43:00000000_0000fd4f 1 00000000_0001013c lddf :G02->00000000_198b3c98 L1R miss (A=00000000_198b3c88 pushA=00000000_198
43:EXWAIT
44:l2r delayed(miss) arrived l2_tag.la changed (dirty=0->1) from 00003c40 to 06f33c40
44:l2r delayed(miss) arrived l2rq->l2 index=241 way=5 A=00000000_06f33c40->00000000_00000000
44:l2r delayed(miss) arrived l2rq->l2 index=241 way=5 A=00000000_06f33c48->00000000_00000000
-(1 of 1) [line 16835/? ??%]
Cache Activities
Thread Activities
Bus Traffic

Goal of the simulator

```
====PE steps, cycles, cache statistics (l1:I$ o1:D$ l2:incore-L2$ g2:other-L2$====

00:step=00000000_00006889 cycle=00000000_0000fedb
  i1( 99.6%)wait=00000000_00006f27
  o1( 84.8% hit=00000000_00000122 mis=00000000_00000034)wait=00000000_00001f2a
  l2(  1.9% hit=00000000_00000003 mis=00000000_00000097)
  g2(  0.0% hit=00000000_00000000 mis=00000000_00000097)

Thread#00 # of instr.     =0x6889
          # of cycles     =0xfedb
          Instr. L1$   hit=99.6% (issue wait cycle=0x6f27…includes multi-threading)
          Data   L1$   hit=84.8% (total wait cycle=0x1f2a)
          Local  L2$   hit= 1.9% (Refers Other L2$   0x97 times)
          Global L2$   hit= 0.0% (Refers Main Memory 0x97 times)

          Improve application to increase hit-ratio on L1$/L2$

====THREAD instruction counts (over 5%)====

th00: 31.4%:bicc 31.8%:sethi 31.4%:barrier

Thread#00 bicc    31.4% of total
          sethi   31.8% of total
          barrier 31.4% of total

          Improve application to execute more floating point instr.
```

Can explore why low-performance



The performance crosses between 8threads and 16threads

Emulation

for stopping maintenance of old computers

Motivation of developing emulators

New instruction set and architecture are developed:

- Engineers want to focus on new products.
  don't want to be maintainer of old products.
- Users want to keep the same software.
  also want to test new hardware.

Attractive software is running on other hardware:

- Hardware vendors need good software.
- Users don't want to keep many hardware.

We want to run many software on single new hardware.

- Simulator is a tool for developing new computers
- Emulator is a tool for old computers

How to execute given program

1. Recompilation
   - requires source code and corresponding OS.
   - is best for high-performance.
   - Is hard for consumers.

2. Binary interpreter (executes instructions by software)
   - requires only binary code.
   - can run with small memory.
   - is very slow (< 1/100).

How to execute given program

3. Static translation
- is possible for assembly source.
- for object code only if insn/data are distinctable.
- is often possible for application binary.
- but needs human support for OS binary.

4. Dynamic translation (Just In Time translation)
- is suitable for unstructured binary.
- needs no human support.
- is faster than interpreter.
- can cover wide range application/OS.

Example: traditional server ⇒ workstation

It is often the case only the president of the company knows how to repair old servers in customers.

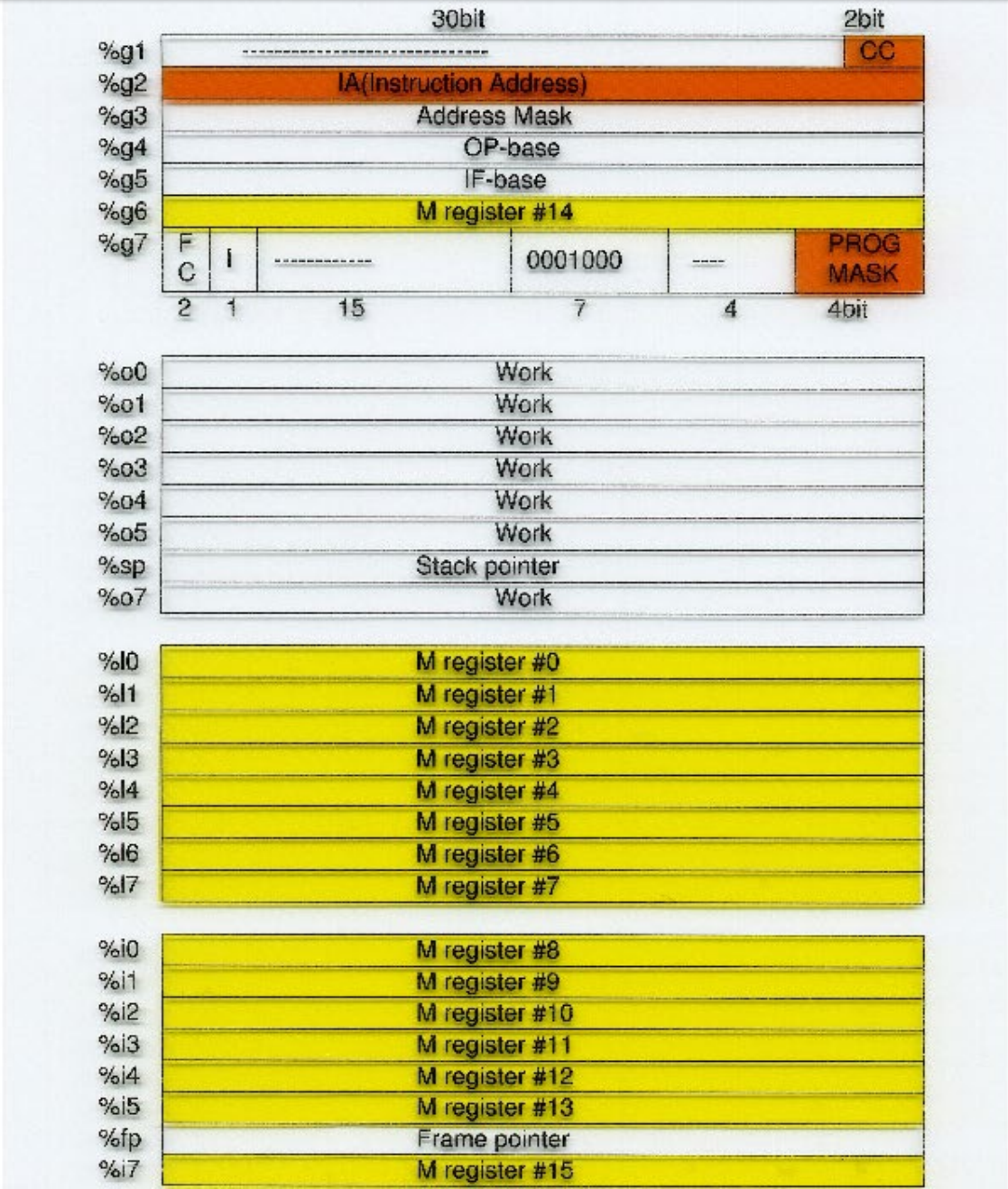
- If such servers crash, the service is stopped.
- Maybe no spare parts to repair.

Target software: IBM370 compatible

New hardware: SPARC workstation

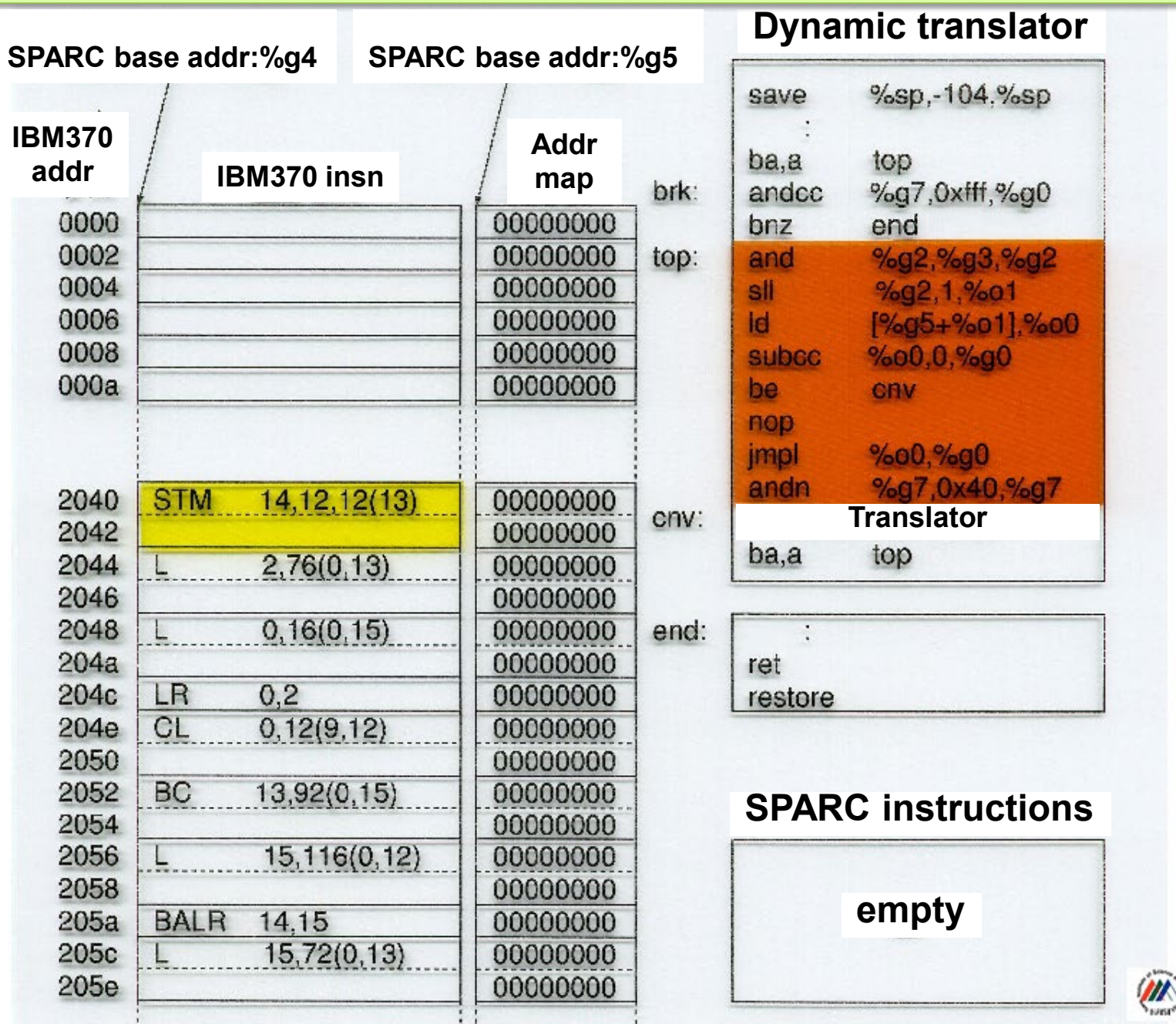
- Because OS is totally different.
- Not only application but also OS should be emulated.

Mapping IBM370 registers on SPARC



NAIST

Before starting tralslation



Translation of AR (add register)

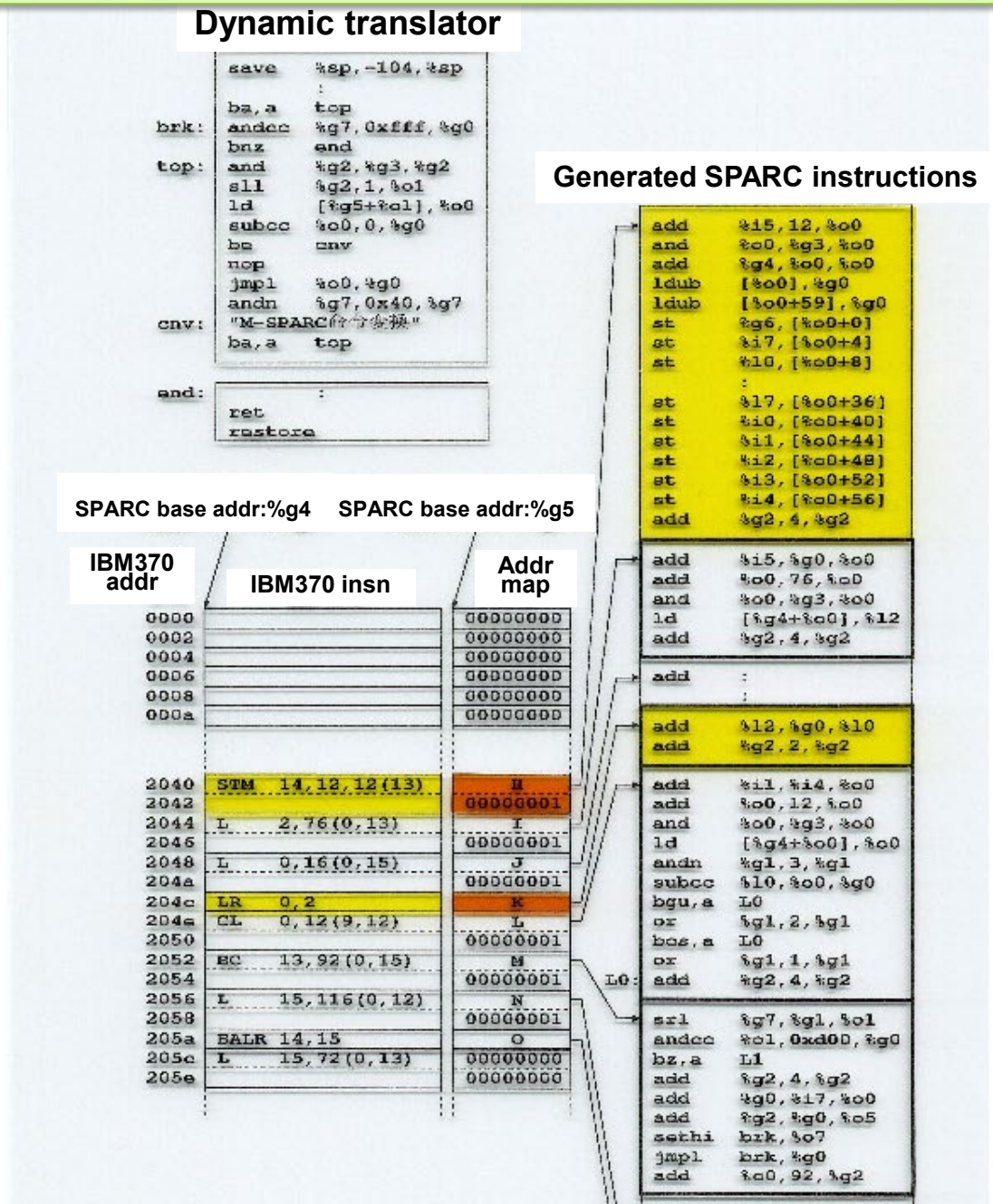
- Translator

```
ci2_ar:
        retl                                    !
        add     %g0,t.ar0e-t.ar0s,%o0           !
ti2_ar:
        sethi   %hi(t.ar0s),%o3                 !
        or      %o3,%lo(t.ar0s),%o3             !
        or      %g0,t.ar0e-t.ar0s,%o4           !
    t.ar0cp:                                    !
        subcc   %o4,4,%o4                       !
        ld      [%o3+%o4],%o5                   ! copy template
        bnz     t.ar0cp                         ! to ibuf
        st      %o5,[%o0+%o4]                   !
        ld      [%o3+t.ar0o0-t.ar0s],%o5        !
        srl     %o1,16,%o4                      !
        and     %o4,0xf,%o4                     ! R2
        subcc   %o4,0xe,%g0                     ! |
        be,a    .+8                             ! |
        add     %g0,0x16,%o4                    ! |
        xor     %o5,%o4,%o5                     ! rs2
        srl     %o1,20,%o4                      !
        and     %o4,0xf,%o4                     ! R1
        subcc   %o4,0xe,%g0                     ! |
        be,a    .+8                             ! |
        add     %g0,0x16,%o4                    ! |
        sll     %o4,14,%o4                      ! |
        xor     %o5,%o4,%o5                     ! rs1
        sll     %o4,11,%o4                      ! |
        xor     %o5,%o4,%o5                     ! rd
        st      %o5,[%o0+t.ar0t0-t.ar0s]        !
        andcc   %g7,0x40,%g0                    ! if EX=0 then IAR+=2
        ld      [%o3+t.ar0oy-t.ar0s],%o5        !
        bnz,a   .+8                             ! if EX=1 then IAR+=4
        ld      [%o3+t.ar0oz-t.ar0s],%o5        !
        st      %o5,[%o0+t.ar0tz-t.ar0s]        !
        retl                                    !
        add     %o0,t.ar0e-t.ar0s,%o0           !
```

- Template

```
        t.ar0s:
                                andn    %g1,3,%g1       ! CC=0
            t.ar0t0:            addcc   %10,%10,%10     ! ##TEMPLATE:%10,%10,%10
                                bvc     t.ar000         ! if overflow then
                                nop                     !
                                andcc   %g7,8,%g0       !  CC=3
                                bz      t.ar0tz         !  if fixed-point
                                or      %g1,3,%g1       !   overflow mask then
                                sethi   %hi(0x00080000),%o0
                                sethi   %hi(epilog0),%o7!
                                jmpl    %o7+%lo(epilog0),%g0
                                or      %g7,%o0,%g7     !   fixed-point overflow
            t.ar000:            bg,a    t.ar0tz         ! if %10 > 0 then
                                or      %g1,2,%g1       !  CC=2
                                bl,a    t.ar0tz         ! if %10 < 0 then
                                or      %g1,1,%g1       !  CC=1
            t.ar0tz:            add     %g2,2,%g2       ! ##TEMPLATE:IAR
        t.ar0e:
            t.ar0o0:            addcc   %10,%10,%10     ! ##TEMPLATE:%10,%10,%10
            t.ar0oy:            add     %g2,2,%g2       ! ##TEMPLATE:IAR
            t.ar0oz:            add     %g2,4,%g2       ! ##TEMPLATE:IAR
```

Before starting tralslation



NAIST

Performance of small program

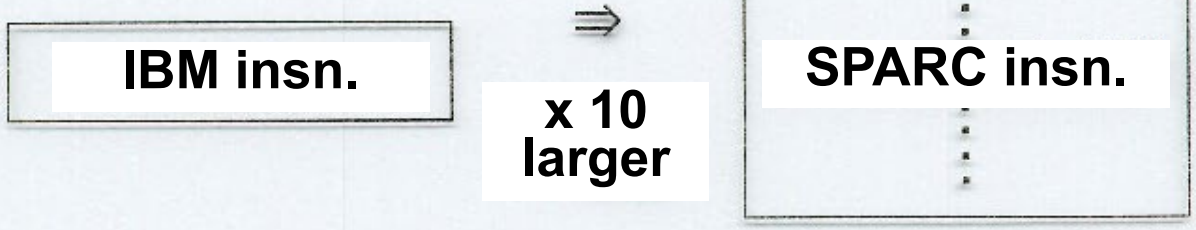
		Dhrystone-2.1 loop =10000	Stanford-integer
Execution time	(a)	1．3 sec.	2．9 sec.
Dynamic IBM steps	(b)	9．6	24．
Translated insn.	(c)	1．2 Kstep	3．2 Kstep
Reuse ratio of insn.	((b-c)/b)	99．99%	99．99%
Executed SPARC insn.	(d)	94．Mstep	223 Mstep
SPARC insn./IBM insn.	(d/b)	9．8倍	9．3倍
SPARC CPI	(a×60MHz/d)	0．83	0．78
IBM MIPS	(b/a)	7．4 MIPS	8．3 MIPS
w/o reuse SPARC insn. Execution time	(e)	16 sec.	38 sec.
	((e-a)/a)	11 x	12 x
L1-cache-miss/IBM insn.		0.52	0.17
L2-cache-miss/IBM insn.		0.048	0.012
Data alignment overhead	(f)	0．00%	0．00%

Performance of large program with OS

		w/o self-modification of insn.	w/ self-modification	
			Intr. Based	Check by insn.
Execution time	(a)	457秒	1.4K sec.	
Dynamic IBM steps	(b)	518 Mstep	←	
in OS		193 Mstep	←	
in application		325 Mstep	←	
Translated insn.	(c)	114 Kstep	←	
Reuse ratio of insn.	((b−c)/b)	99.98%	←	
IBM MIPS	(b/a)	1.1 MIPS	0.35 MIPS	
IBM MIPS w/o monitor		2.7 MIPS	0.45 MIPS	
L1-cache-miss/IBM insn.		0.88	1.09	
L2-cache-miss/IBM insn.		0.14	0.20	
Data alignment overhead	(f)	1.83%	0.63%	

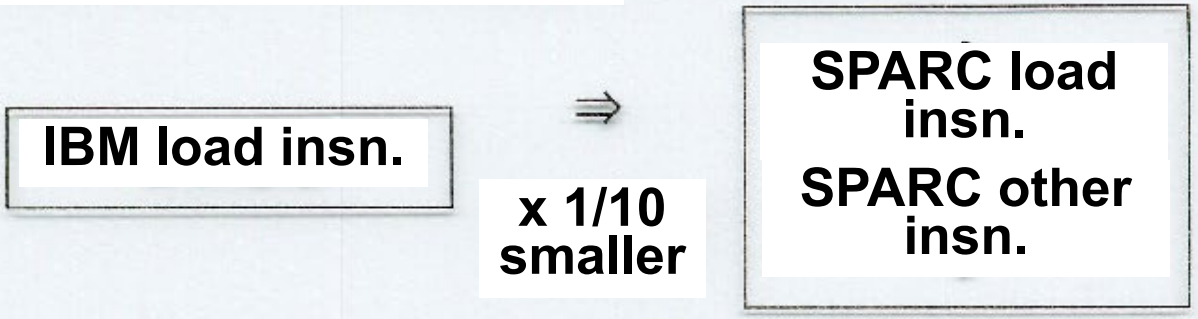
Characteristics of cache are changed

Instruction cache



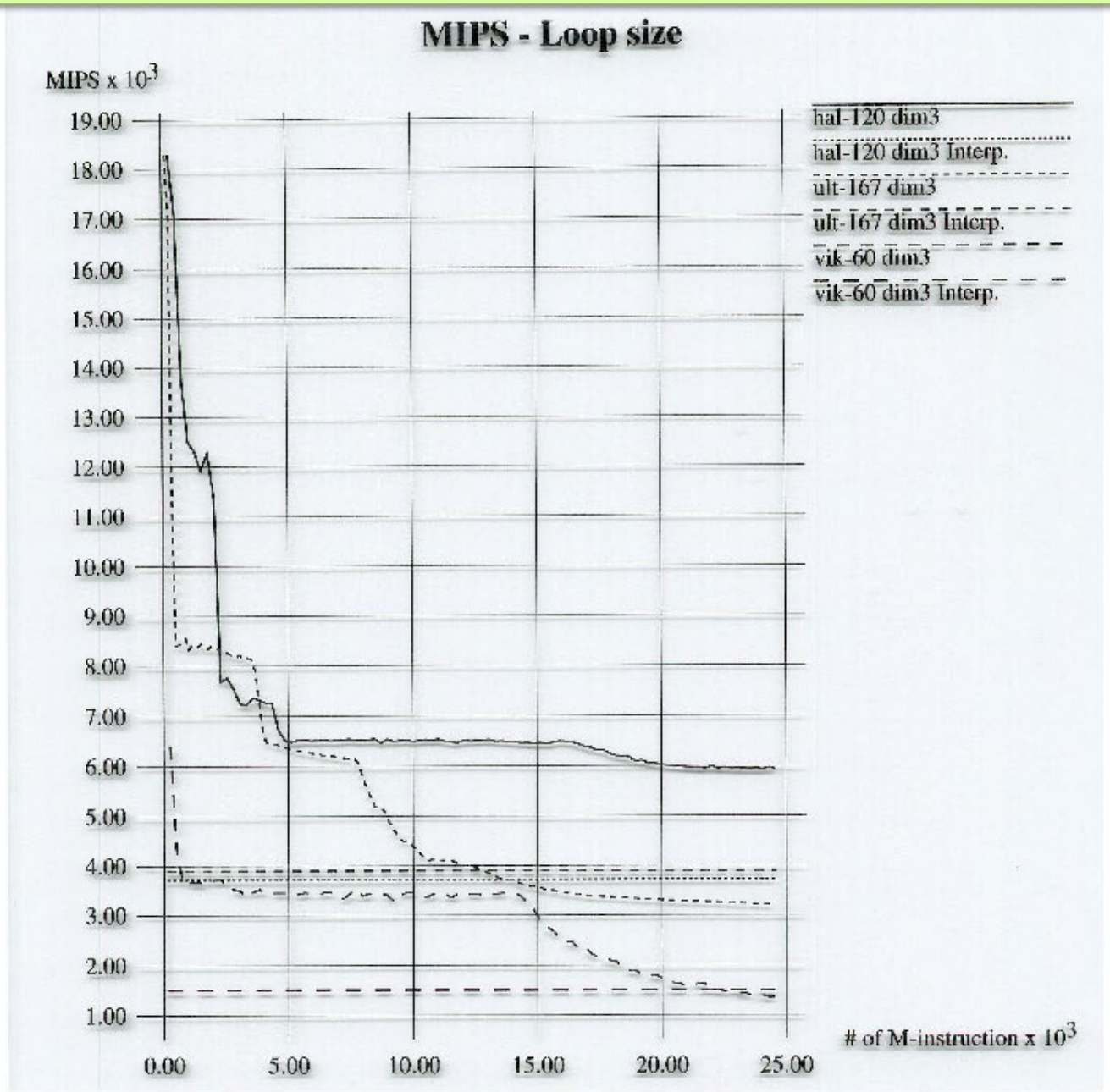
Instruction Cache Hit-ratio ↓

Data cache



Data Cache Miss-penalty ↓

Characteristics depend on host computers



NAIST

Summary

Simulator:

- is a tool for developing new computers.
- should be designed for what you want to know.

Emulator:

- is a tool for replacing old computers.
- should be designed for what you want to save.

That’s all for today