A Multi-threading RISC Cluster for Video Systems

2008. 06.

Sanggyu Park and Soo-Ik Chae

Center for SoC Design Technology

Seoul National University

Seoul, Korea

Contents

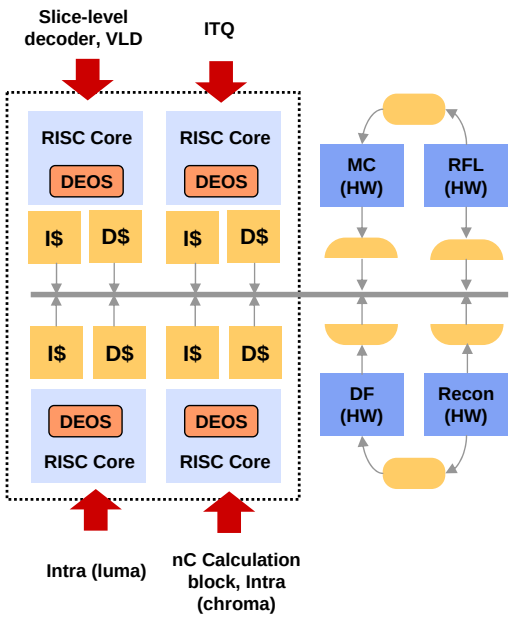
- Motivation
- Proposed cluster
  - Simple RISC cores
  - Hardware operating system kernel
  - Shared caches
  - Communication Co-processor
- Case Study: H.264 decoder
- Summary

HW-SW Partitioning for H.264 D1 decoder

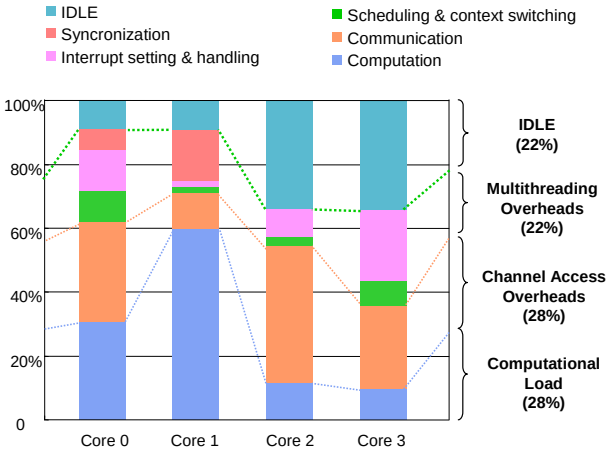
Software Simulation on ARM9 ISS Model

Computational Modules	Required cycle counts per macro-block			Required MIPS				
	I picture	P picture	(1I+9P)/10	QCIF	QVGA	D1	720p	1080p
NAL decoder	288	117	134	1	**1**	**5**	14	**33**
Sequence-level parser	3	5	4	1	**1**	**1**	1	**1**
Slice-level parser	4,254	5,549	5,419	16	**49**	219	585	1,317
VLD block	16,253	2,733	4,085	12	**37**	165	441	993
nC Calculation block	3,890	1,332	1,588	5	**14**	64	171	386
Inverse transform	37,998	4,578	7,920	24	71	321	855	1,925
Intra-prediction block	12,915	884	2,087	6	**19**	85	225	507
Inter-prediction block	645	69,623	62,725	186	565	2,540	6,774	15,242
Reference Image Loader	155	55,425	49,898	148	449	2,021	5,389	12,125
Reconstruction block	22,272	21,528	21,602	64	194	875	2,333	5,249
De-blocking Filter	116,144	84,012	87,225	259	785	3,533	9,420	21,196
Total	214,815	245,784	242,687	721	2,184	9,830	26,210	58,973

An example: H.264 D1 decoder with 4 RISC Cores



A multiprocessor-based H.264 decoder system

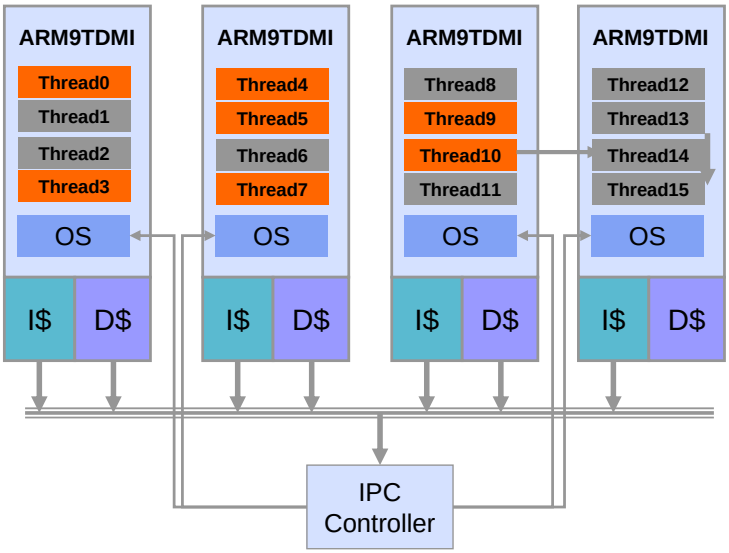


Profiling results for Intra Pictures (RT-level simulation)

IDLE cycles
multithreading overheads
channel access overheads!!

➔ only 28% for computation
➔ lower S/W performance
(D1 10 fps)

Complexity Overhead



4KB 2-way Set-Associative Cache

- 3.5K gates for controller
- 6.0K gates for TAG memory
- 32K gates for cache memory

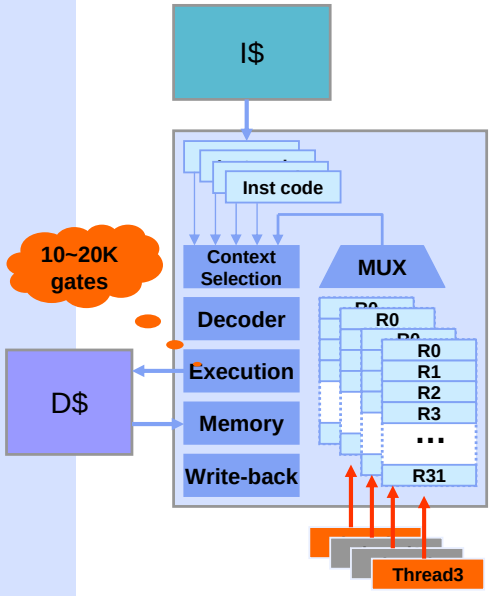
{ARM9T+4KB I$+ 4KB D$}*4
= 532Kgates

Large area overhead

We need a simpler RISC cluster with higher performance!

Solution: A cluster of SMT processors

Simplified SMT Architecture



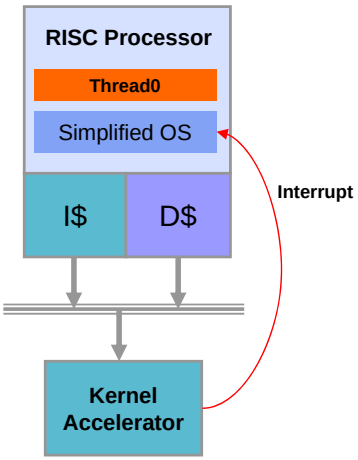
Can hide cache miss penalty with low overhead context switching

Examples: MIPS32K®, SPARC V9 processors

Register count	Read port count	Write port count	Logic complexity (Gates) and delay (ns)							
			1-bank		2-bank		3-bank		4-bank	
			Gates	delay	Gates	delay	Gates	delay	Gates	delay
16	3	2	8.4K	1 ns	18.9K	1 ns	28.8K	1 ns	36.3K	1 ns
16	4	2	9.1K	1 ns	20.0K	1 ns	29.6K	1 ns	38.0K	1 ns
32	3	2	17.0K	1 ns	45.7K	1.1ns	69.8K	1.1ns	89.6K	1.1ns
32	4	2	18.8K	1 ns	53.7K	1.1ns	78.4K	1.1ns	101.9K	1.1ns

- **can support a large number of threads**
- **large area overhead**

Solution: Hardware-accelerated Kernel



Use simplified OS + kernel accelerator

Examples: δ framework, Real-Time Task Manager
Silicon TRON, Real-time Unit

Kernel accelerator

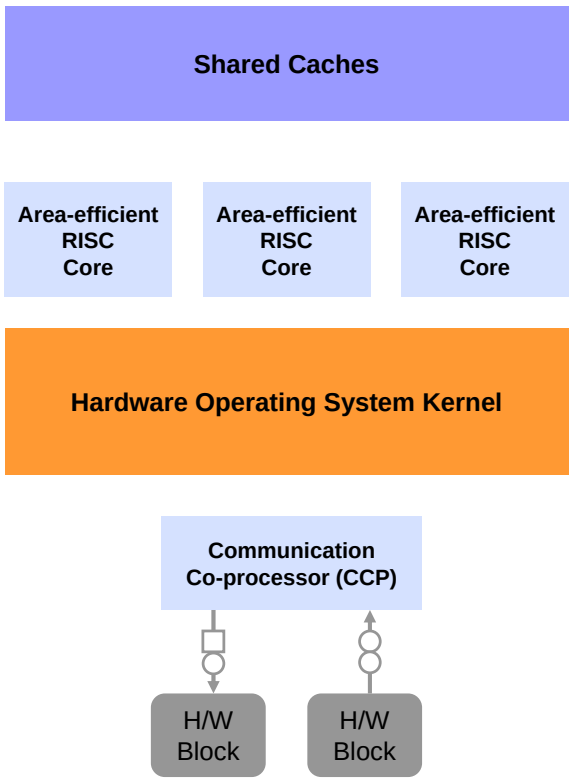
- Task scheduling
- Mutex & semaphore
- Time management

Example: Silicon TRON

# of Task	16	32	64	128
Area (Gates)	40K	100K	190K	N/A

- Reduce scheduling overhead
- Context switching overhead need to be reduced more
- Hardware kernel accelerators need to be more area-efficient

Our Solution: Configurable RISC Cluster



- **efficient use of caches**
- **more area-efficient**
- **no cache coherency problem**
- **need to reduce cache conflicts**

- **simple RISCs (25 ~ 32K gates}**
- **reduced context**

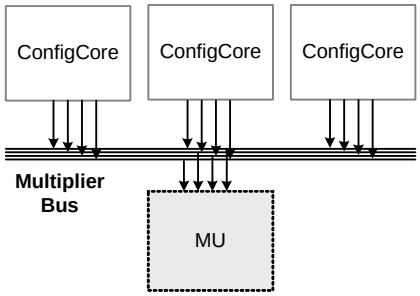
- **task scheduling**
- **context pre-fetching**
- **context switching**
- **semaphore & mutex**
- **interrupt handling**

- **direct connection with H/W blocks by channels**
- **communication with single instruction**

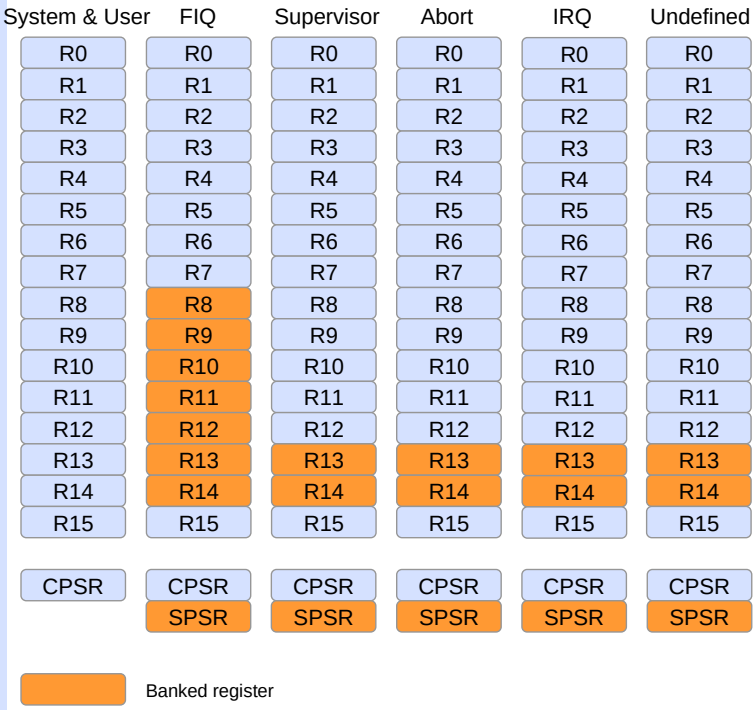
Sharing a multiplier

ARM9TDMI: a 32x16 multiplier (13,212 gates)

Applications (Mediabench-I)	Category	Half-word operand only	One word-size Operand	Two word-size operands	Total multiplications
G721 decoder	Audio	0.52%	0.01%	0.00%	0.53%
GSM encoder	Audio	3.13%	4.62%	2.12%	9.88%
JPEG decoder	Video	0.38%	0.07%	0.00%	0.45%
MPEG decoder	Video	0.22%	0.00%	0.00%	0.22%
EPIC encoder	Video	0.01%	0.00%	0.12%	0.13%
PGP	Encryption	0.43%	0.00%	0.00%	0.43%



Eliminating banked registers

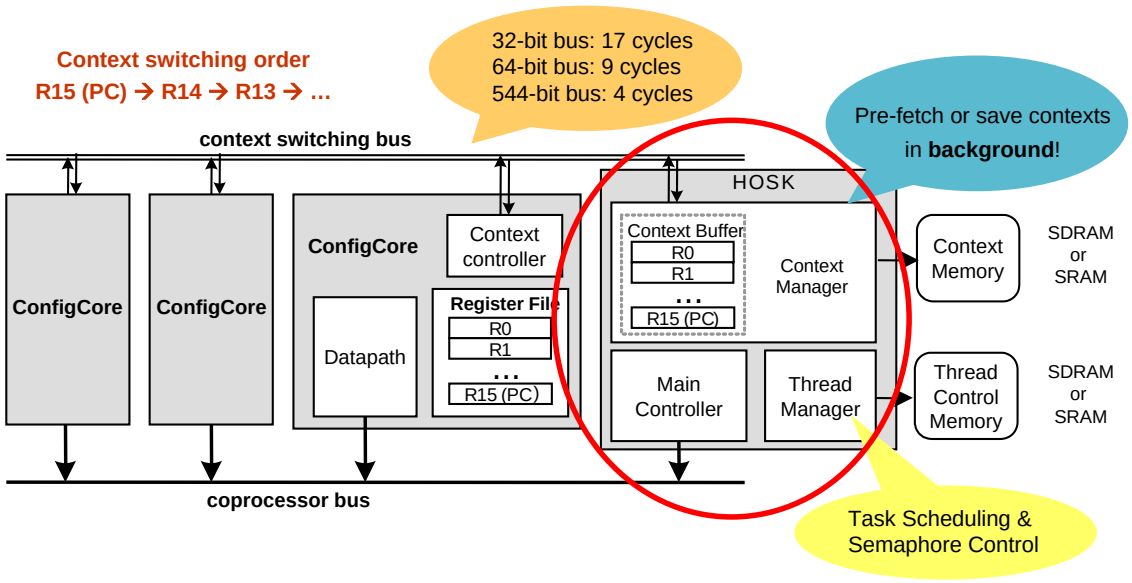


ARM9 processor supports six operation modes

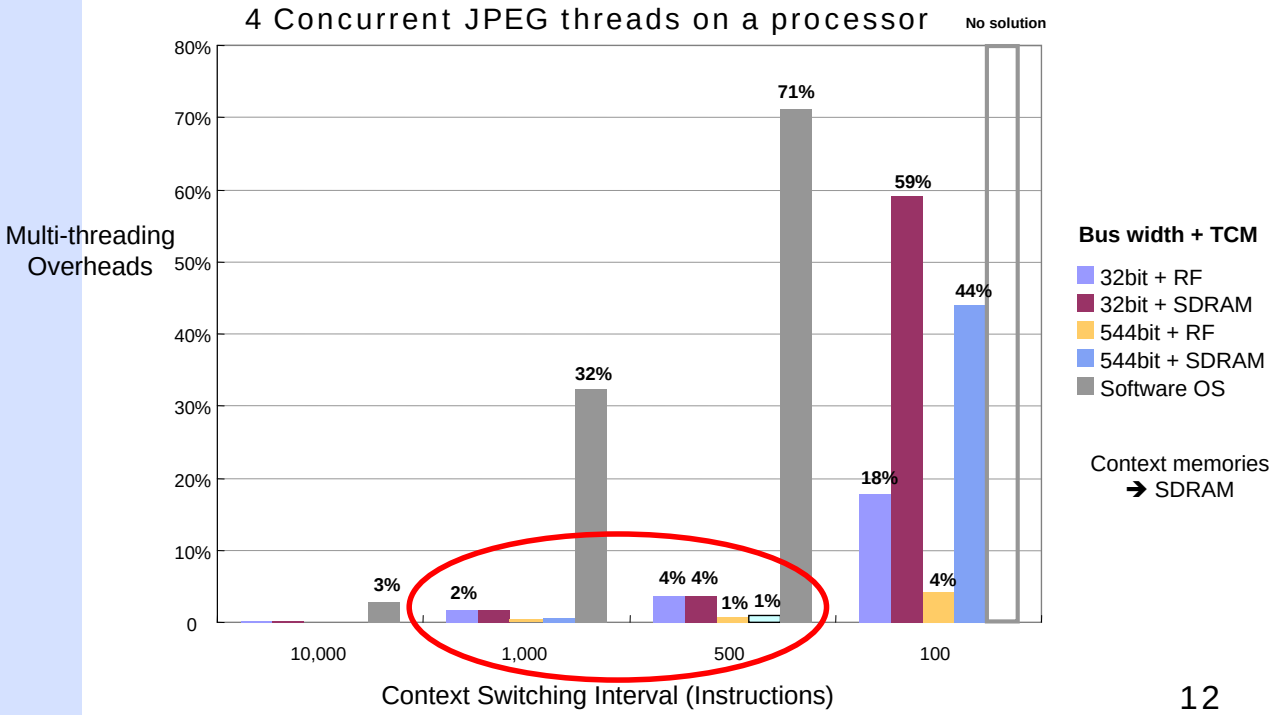
Total 37 banked registers

additional 20 registers that support multiple modes. (about 10Kgates)

Architecture of the HOSK



Multi-threading Overheads of the HOSK



Sharing I-cache

Observations

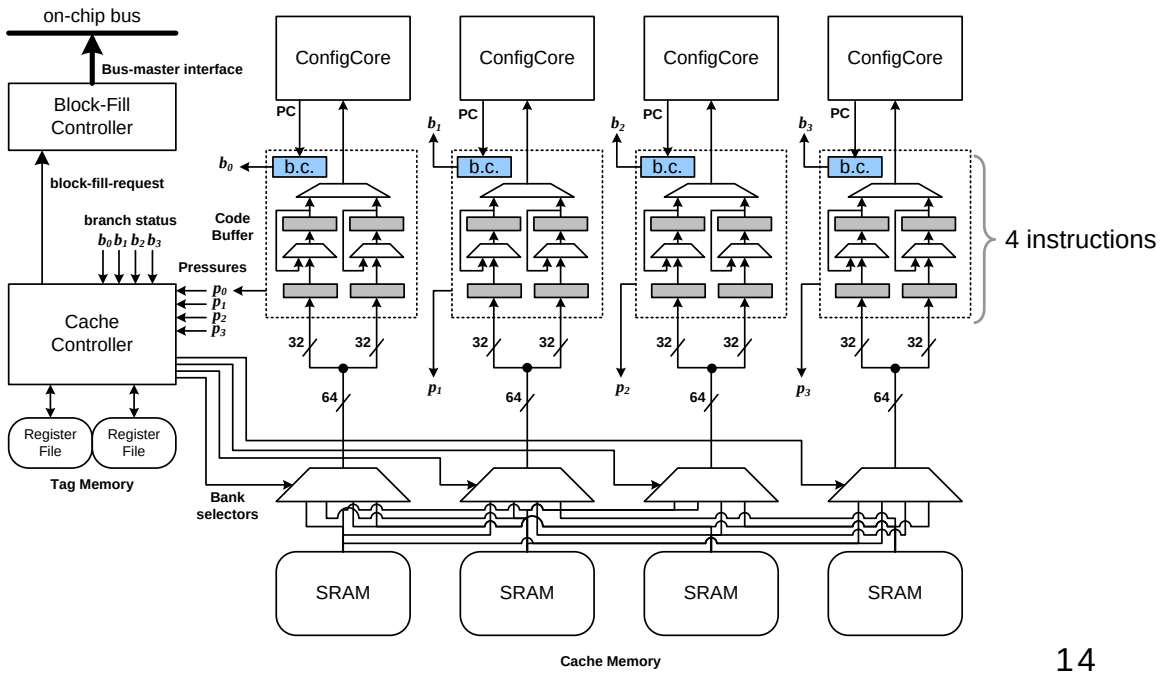
1. 90% of instructions are sequentially fetched: predictable.
2. Cycle-per-instruction (CPI) is not 1 but 1.3 ~ 1.5
3. The code size of each of the threads is different

	slice ctrl	dfsh	dfsv	mvd	VLD	Intra	Inverse transform
Code Size	6.4KB	1.0KB	0.9KB	5.9KB	2.4KB	8.0KB	2.4KB

4. A large SRAM is more compact than a set of several segmented SRAMs

32x256: 1.4 gates/bit 32x512: 1.0 gate/bit 32x1024: 0.8 gates/bit

2-way Set-associative Shared Instruction Cache

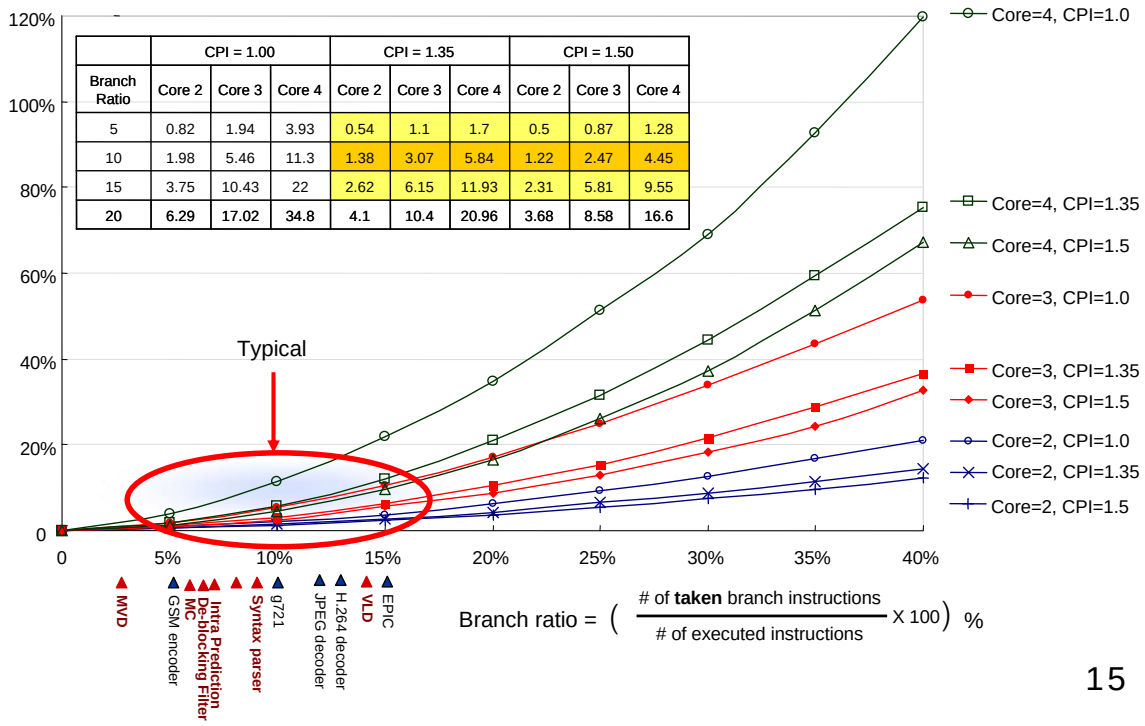


Sharing Instruction Cache

Performance degradation due to fetch conflicts

$$\text{Performance Degradation} = \left(\frac{\text{\# of stalled cycles due to I-\$ sharing}}{\text{\# of executed instructions}} \times 100 \right) \%$$

Branch Ratio	CPI = 1.00			CPI = 1.35			CPI = 1.50		
	Core 2	Core 3	Core 4	Core 2	Core 3	Core 4	Core 2	Core 3	Core 4
5	0.82	1.94	3.93	0.54	1.1	1.7	0.5	0.87	1.28
10	1.98	5.46	11.3	1.38	3.07	5.84	1.22	2.47	4.45
15	3.75	10.43	22	2.62	6.15	11.93	2.31	5.81	9.55
20	6.29	17.02	34.8	4.1	10.4	20.96	3.68	8.58	16.6



$$\text{Branch ratio} = \left(\frac{\text{\# of \textbf{taken} branch instructions}}{\text{\# of executed instructions}} \times 100 \right) \%$$

Sharing Data Cache

Sharing a data cache among the RISC cores doest not require a cache coherence protocol.

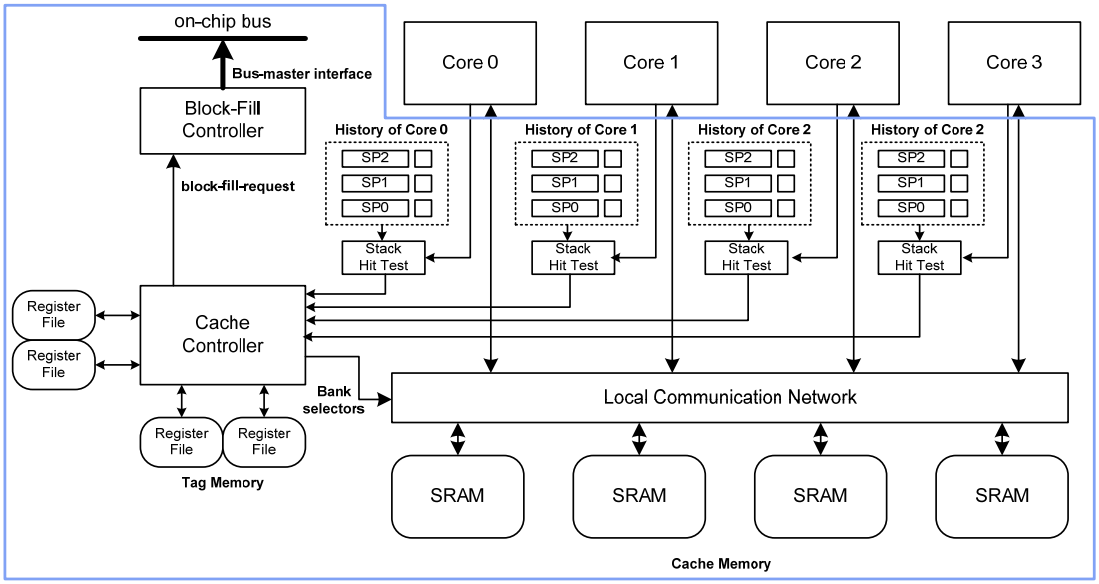
Observations:

1. Load and store instruction are about 15%
2. Prediction-based pre-fetching does not work
3. About 75% of the data accesses are for stack memory

Idea: Keep the history of data cache hits near the stack pointers
And skip tag matching if the requested block is in the history.

Sharing Data Cache

Architecture of Shared Data Cache

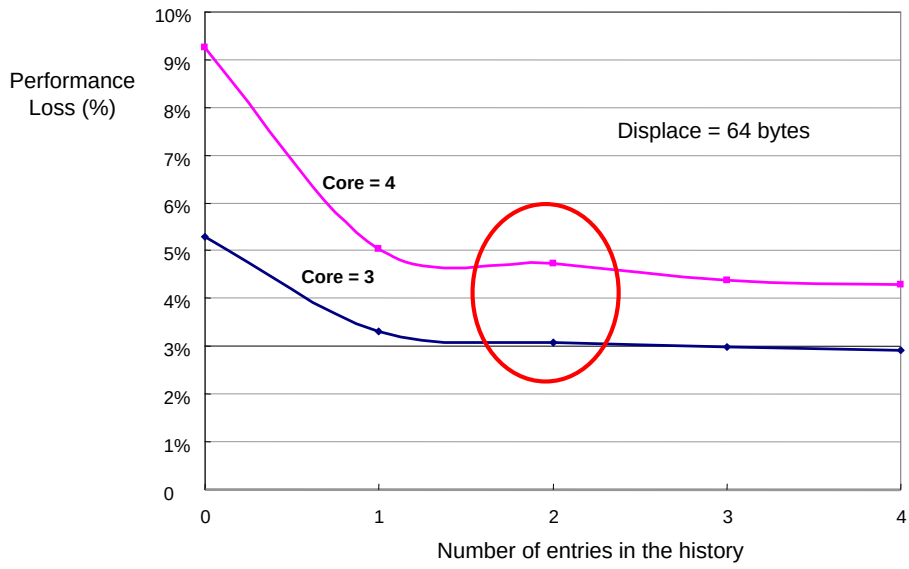


Logic Complexity: **27KGates** for 4-core 4-way 8KB shared data cache with 2 histories for each core

Sharing Data Cache

Performance loss due to access conflicts

In a H.264 decoder with the proposed multithreading cluster, load/store instructions are about 15%



Performance losses with different number of entries in the history. (H.264 decoder example)

S/W Code for Inter-processor Communication

```
int get(void)
{
    u_int status;
    u_int* base = BASE_ADDR;
    status = base[CTRL_REG];
    while((status & READY) == 0)
    {
        wait_interrupt(INTR_SRC);
        status = base[CTRL_REG];
    }
    u_int value = base[DATA_REG];
    return value;
}
```

```
get: stmfd   sp!, {r4, lr}          (2)
     mov     r4, #BASE_ADDR         (1)
.L3: ldr     r3, [r4, #CTRL_REG]    (1)
     tst     r3, #READY             (1+1)
     bz      .L5                    (1)
.L4: ldr     r0, [r4, #DATA_REG]    (1)
     ldmfd   sp!, {r4, pc}          (5)
.L5: mov     r0, #INTR_SRC
     bl      wait_interrupt
     b       .L3
```

13 inst.

```
SC_MODULE(cmpt)
{
   sc_port<sgi<u_short> > i_code;
   sc_port<spi<u_int> > o_value;

   void do_cmpt(void) {
      while(1) {
         ....
         u_short code = i_code->get();
         ....
         o_value->put(result);
   }
};
```

```
class sc_port
{
   sc_module* operator->(void)
   {
      return m_object;
   }
}
```

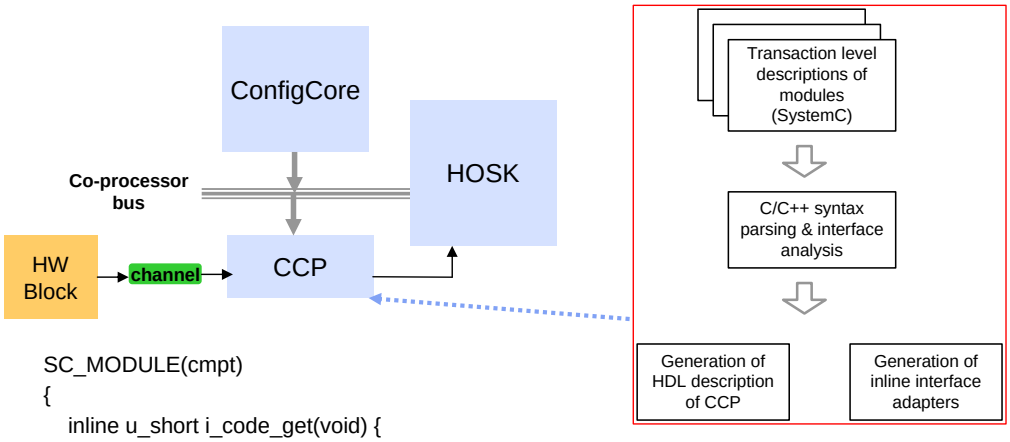
Virtual method call

```
ldr r0, [fp, #52]   # 1 cycle
ldr ip, [r0]        # 2 cycle
mov lr, pc          # 1 cycle
ldr pc, [r5]        # 5 cycle
```

```
stmfd  sp!, {r4, r5, lr}    # 3 cycle
...
ldmia  sp!, {r4, r5, lr}    # 3 cycle
```

Total 15 cycle + cache misses
+ caller-save registers

Automatic CCP Generation




```
SC_MODULE(cmpt)
{
   inline u_short i_code_get(void) {
        int result;
   __asm__ ("mrc  p0, 1, %0, c9, c0, 2\n"  : "=r"(result));
   return result;
   }
   void do_cmpt(void) {
      while(1) {
         ....
         u_short code = i_code_get();
         ....
```

compile

```
mrc  p0, 1, %0, c9, c0, 2
```

(no cache misses)
(no saving caller-save registers)

Case Study: H.264 decoder

D1 H.264 decoder using a RISC cluster

Software Modules

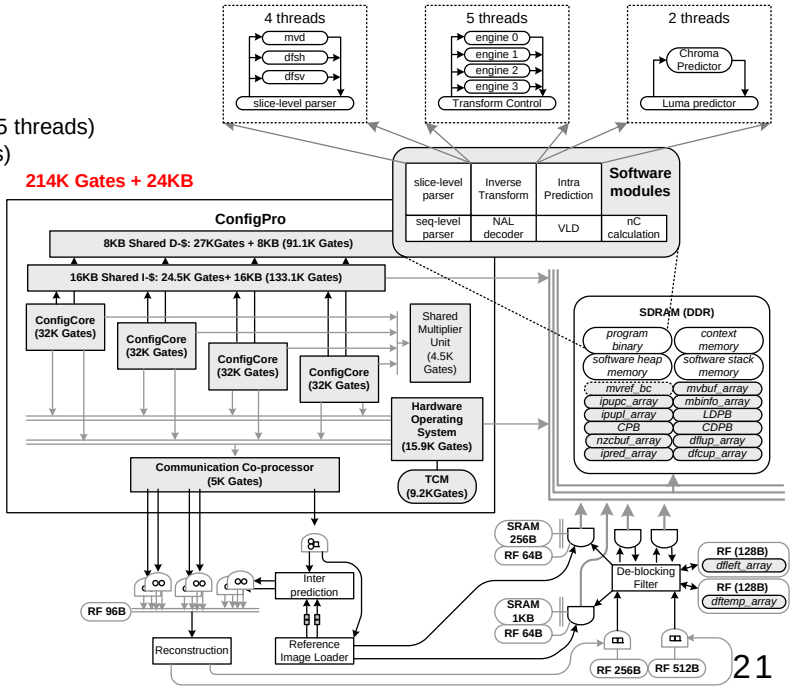
- Sequence-level parser
- NAL decoder
- VLD
- nC calculation block
- Slice-level parser (4 threads)
- Inverse transform block (ITQ) (5 threads)
- Intra prediction block (2 threads)

Hardware Modules

- Inter prediction block
- Reference image loader block
- De-blocking filter block
- Reconstruction block

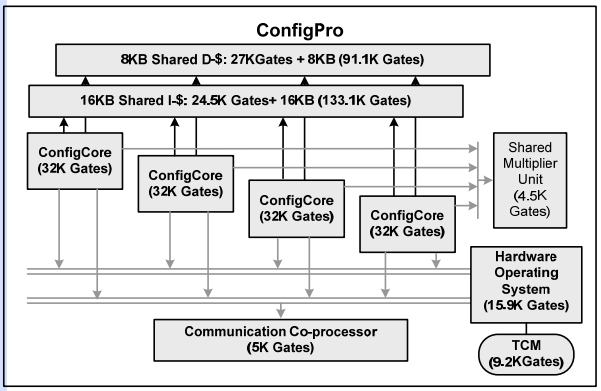
Multithreading Cluster

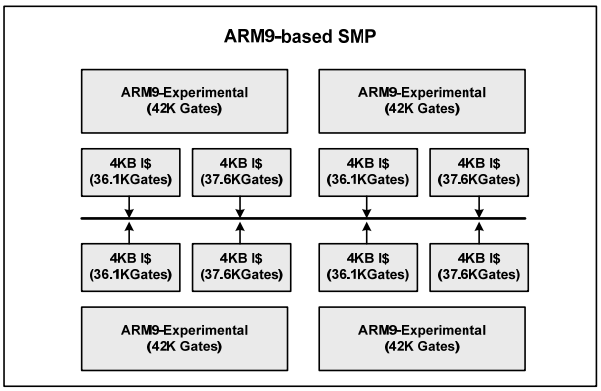
- Four RISC cores
- 16KB Shared instruction cache
- 8KB Shared data cache
- Area: 214KGates + 24KB
- Clock Frequency: 200 MHz



Case Study: H.264 decoder

Complexities





214.1K Gates + 24KB (386.8K Gates)

ConfigCore (32K Gates)

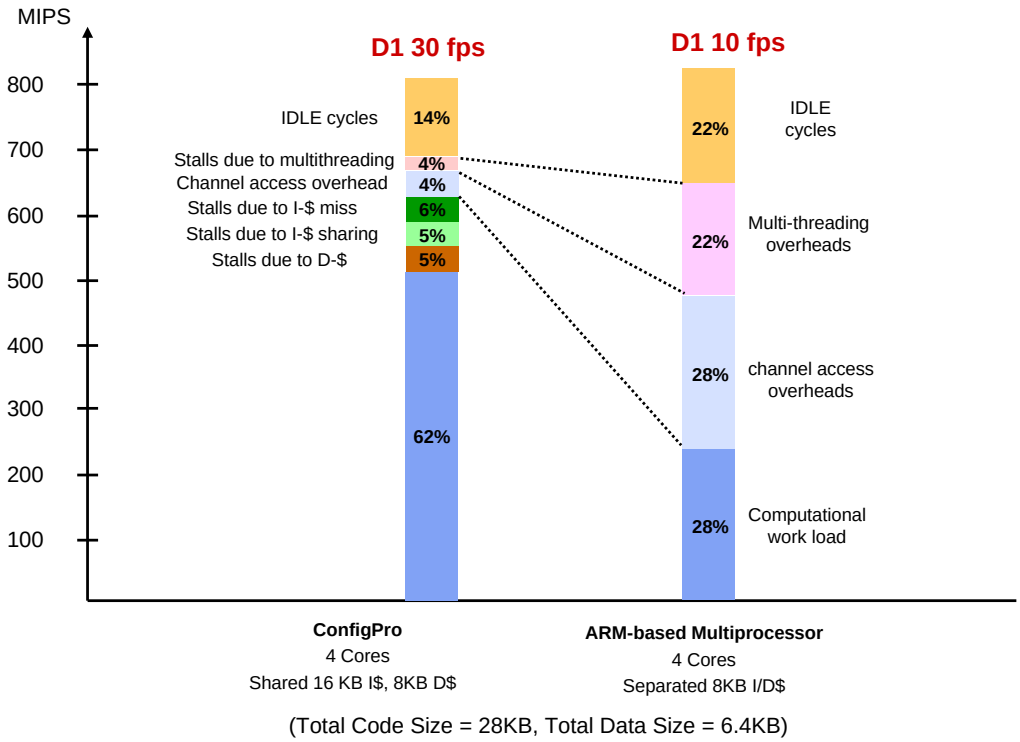
- ARM9 ISA
- Branch prediction (4-entry)
- Out-of-order completion
- Decode-stage branch
- **HOSK support**
- **Shared multiplier unit**

202K Gates + 32KB (462.8K Gates)

ARM9-Experimental (42K Gates)

- ARM9 ISA
- Branch prediction (4-entry)
- Out-of-order completion
- Decode-stage branch
- **SPSR registers, banked registers except FIQ**
- **Privileged mode LDM/STM**
- **Dedicated multiplier unit**

Performance Comparison



(Total Code Size = 28KB, Total Data Size = 6.4KB)

Summary

- For video applications we implemented an efficient RISC cluster that consists of
  - 4 simplified RISCs with shared I and D caches,
  - a hardware scheduler, and
  - a communication co-processor
- It is useful for fine-grain multi-threading applications.

Thank you!

References

[1] Andreas Gerstlauer, “SpecC modeling guidelines”, Technical report CECS-02-16, Center for Embedded Computer Systems, University of California, Irvine, Apr. 2002.

[2] K. Keutzer, A. Sangiovanni-Vincentelli, “System level design: Orthogonalization of concerns and platform-based design”, Trans. on computer aided design of integrated circuits and systems, Vol. 19, No. 12, Dec. 2000

[3] A.C.J. Kienhuis, “Design Space Exploration of Stream-based Dataflow Architectures Methods and Tools”, Ph.D Thesis, Delft University, 1999

[4] Wander O. Cesario, Sungjoo Yoo, et. al, “Multiprocessor SoC Platforms: A Component-Based Design Approach”, IEEE Design & Test, 2002

[5] Thorsten Grotker, Stan Liao, Grant Martin, Stuart Swan, “System design with SystemC”, Kluwer Academic Publisher, 2002.

[6] Simon Segars, “The ARM9E Synthesizable Processor Family”, http://www.arm.com, 1999.

[7] J.J. Lee and V.J. Mooney, III, “Hardware/software partitioning of operating systems: focus on deadlock detection and avoidance”, IEE Proc.-Comput. Digit. Tech., Vol. 152, No. 2, March 2005

[8] Paul Kohout, Brinda Ganesh and Bruce Jacob, “Hardware Support for Real-time Operating Systems”, Proc. on CODES-ISSS’03, Oct. 2003.

[9] Takumi Nakano, Andy Utama, et. al., “Hardware Implementation of a Real-time Operating System”, Proceedings of TRON, 1995.

[10] Susanna Nordstrom, Lennart Lindh, et. al., “Application Specific Real-Time Microkernel in Hardware”, Proceedings of International Conference on Computer Designs: VLSI in computers and processors, 2005.

[11] Simon Segars, “The ARM9 Family – High performance microprocessors for embedded applications”, Proceedings of international conference on computer design: VLSI in computers and processors, 1998.

[12] Kiyofumi Tanaka, “PRESTOR-1: A Processor Extending Multithreaded Architecture”, Proceedings of the Innovative Architecture for Future Generation High-Performance Processors and Systems, 2005.

[13] Sanjeev Kumar, Christopher J. Hughes, Anthony Nguyen, “Carbon: Architectural Support for Fine-Grained Parallelism on Chip Multiprocessors”, Proceedings on ISCA’07, June, 2007.

[14] Tesuo Hironaka, Moto Maeda, Kazuya Tanigawa, et. al., “Superscalar Processor with Multi-Bank Register File”, Proceedings of the Innovative Architecture for Future Generation High-Performance Processors and Systems, 2005.

[15] Sanggyu Park, “Template-based System Design Environment for Multimedia Systems”, Ph.D Thesis, Seoul National University, Feb. 2008.

[16] Sanggyu Park, Do-sun Hong, Soo-Ik Chae, “A Hardware operating system kernel for multiprocessor systems”, IEICE Electronics Express (ELEX), Vol. 5, No. 9, May. 2008.