

Advanced Computer Systems Architecture

Chip-Multiprocessors: Applications and Architectures

CSE 561M

Prof. Patrick Crowley

Plan for Today

- Questions
- Today's discussion

Objectives

- Read ME assembly code
- Scratch ring mechanisms
- Scratch ring performance

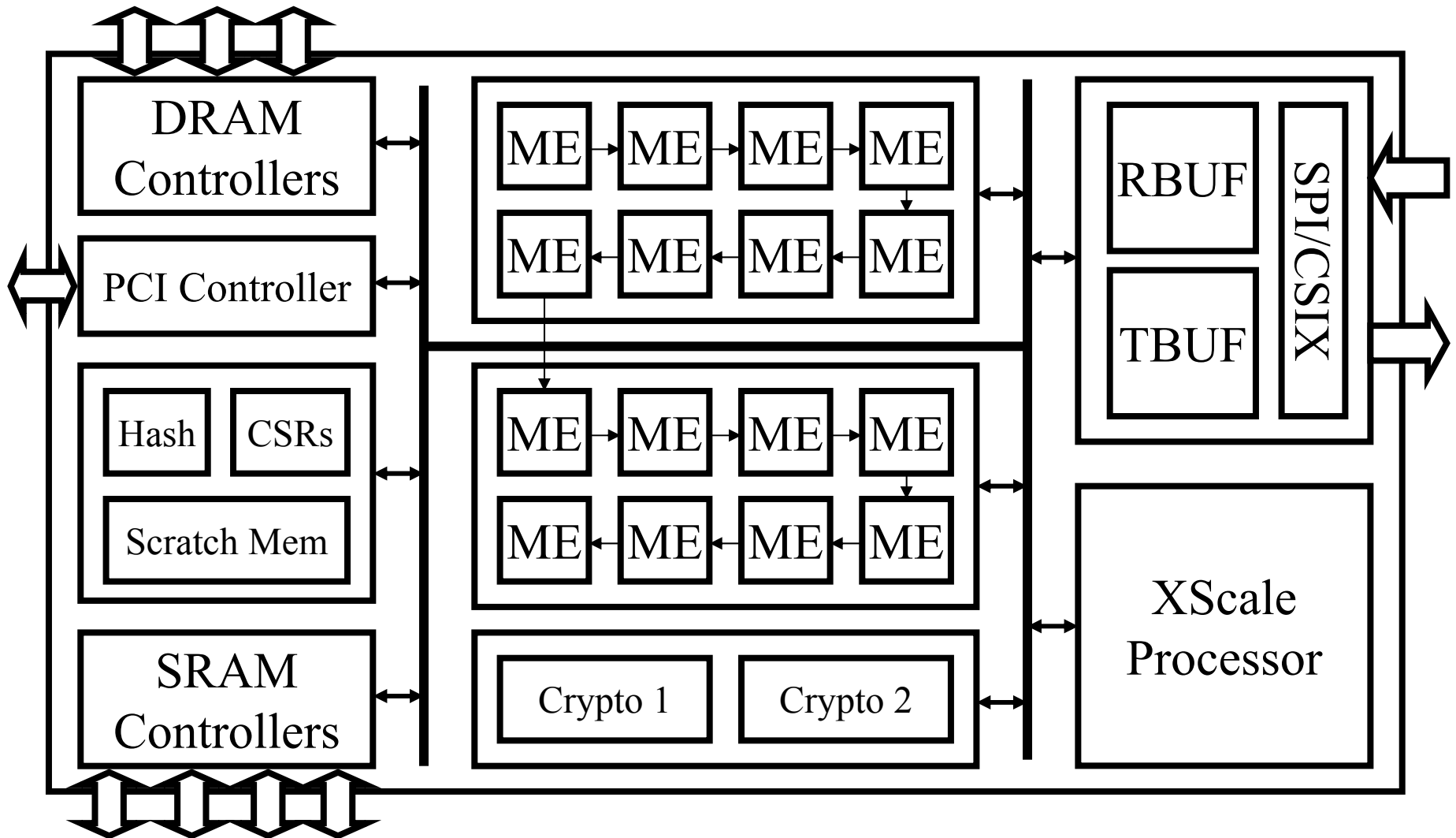
Scratchpad Rings (I)

- Purpose: share data between microengines, XScale and other units
- Ring data structure
 - Operations: get, put, full? empty?
 - Can be implemented with an array
 - Record location of head and tail
- IXP Characteristics
 - 16 (0 .. 15) scratch rings supported in HW
 - 4 configurable sizes (words): 128, 256, 512, 1024
 - The first 11 have HW support for testing fullness
 - Once initialized, a ring supports get and put operations of between 1 and 8 words

Scratchpad Rings (II)

- Ring descriptors reside in SHaC
 - base, size, head pointer, tail pointer
- Steps
 - Initialize
 - Insert Data
 - Check for Fullness/Emptiness
 - Remove Data

IXP 2850 Block Diagram



Initializing/Creating a Ring

- Must configure three Ring-specific CSRs
 - SCRATCH_RING_BASE
 - SCRATCH_RING_HEAD
 - SCRATCH_RING_TAIL
- Programmer's Reference manual has details...

Initialization Macro

```
#macro scratch_ring_init(IN_RING_NUM, IN_RING_BASE, IN_RING_SIZE)
.begin

    .reg $ring_init_xfer $ring_head $ring_tail ring_init base_shift
    .sig ring_init_sig ring_head_sig ring_tail_sig

    immed[base_shift, IN_RING_BASE]
    alu_shf[ring_init, --, B, base_shift, <</**/RING_BASE_BITPOS]
    alu_shf_left($ring_init_xfer, ring_init, OR, ((IN_RING_SIZE/128)-1), RING_SIZE_BITPOS)

    cap[write, $ring_init_xfer, SCRATCH_RING_BASE_/**/IN_RING_NUM], sig_done[ring_init_sig]

    immed32($ring_head, 0)
    cap[write, $ring_head, SCRATCH_RING_HEAD_/**/IN_RING_NUM], sig_done[ring_head_sig]

    immed32($ring_tail, 0)
    cap[write, $ring_tail, SCRATCH_RING_TAIL_/**/IN_RING_NUM], sig_done[ring_tail_sig]

    ctx_arb[ring_init_sig, ring_head_sig, ring_tail_sig]

.end
#endm
```

Scratch Ring Base Register

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
SIZE		RESERVED			RING_STATUS_FLAG	RESERVED											BASE					RESERVED										

*Intel IXP2400 and IXP2800 Network Processor
Programmer's Reference Manual, p. 341*

Head and tail registers are similar

Inserting Data (put)

- Choose a ring
- Move data into up to 8 SRAM transfer registers
 - Use `xbuf` library for this purpose
- ‘put’ data into ring

Put Macro

```
#macro scratch_ring_put_lw(IN_RING_NUM, in_data)
.begin
    .reg ring_addr
    .sig ring_sig
    // the following is a DECLARATION macro!, xbuf.uc.
    xbuf_alloc($ring_data, 1, write)

    immed32(ring_addr, (IN_RING_NUM<<2))

    move($ring_data[0], in_data)
    scratch[put, $ring_data[0], ring_addr, 0, 1], ctx_swap[ring_sig]

    xbuf_free($ring_data)
.end
#endm
```

Examining Fullness/Emptiness

- When is it safe to write/read?
- For 12 of the 16 rings, we can check ring status with a single instruction
 - `br_inp_state, br_!inp_state`
 - these are branch instructions
 - Base register “ring status flag” chooses full v. empty

Fullness Macro

- Response to scratch ring status depends on the ring status flag in the scratch ring base register
 - 0 implies test for fullness
 - 1 implies test for emptiness

```
//This macro assumes the base register status flag is 0
#macro br_scratch_ring_full(IN_RING_NUM, IN_FULL_TARGET)
    br_inp_state[SCR_Ring/**/IN_RING_NUM/**/_Status, IN_FULL_TARGET]
    // You might need to use _Full rather than _Status (SDK v. 3.1)
    // SCR_Ring... understood by the decoder
#endm

//This macro assumes the base register status flag is 0
#macro br_scratch_ring_not_full(IN_RING_NUM, IN_FULL_TARGET)
    br_!inp_state[SCR_Ring/**/IN_RING_NUM/**/_Status, IN_FULL_TARGET]
    // You might need to use _Full rather than _Status (SDK v. 3.1)
    // SCR_Ring... understood by the decoder
#endm
```

Removing Data (get)

- Choose a ring
- Prepare receive transfer registers (up to 8)
- ‘get’ the data from the ring

Get Macro

```
#macro scratch_ring_get_lw(IN_RING_NUM, out_data)
.begin
    .reg ring_addr
    .sig ring_sig

    xbuf_alloc($ring_data, 1, read)

    immed32(ring_addr, (IN_RING_NUM<<2))

    scratch[get, $ring_data[0], ring_addr, 0, 1], ctx_swap[ring_sig]
    move(out_data, $ring_data[0])

    xbuf_free($ring_data)
.end
#endm
```

Putting It All Together

- Example project

2800 Scratch Put Performance

- 128 4-byte entries enqueued in 7305 cycles
 - Data size: 512B
 - Time: 7305 cycles@1400 MHz (0.7 ns) = 5136 ns
 - Rate: 95 MB/s (760 Mb/s)
- Bulk version (8 words per “put”)
 - Time: 1510 cycles = 1057 ns
 - Rate: 462 MB/s (3696 Mb/s)
- Additional code in the project shows that this performance can be sustained between MEs

Assignment

- Thursday
 - J&K: Skim Ch 1&2, read Ch 3
- Tuesday
 - Skim Ch. 4, Read Ch. 5
 - Commentary: Choose one of the following (I'll post this project to the newsgroup)
 - How would you improve on the scratchpad ring put performance? Describe your design, implement it and report results.
 - Implement an analogous set of macros for SRAM rings. Describe the differences and performance and share your implementation in your commentary.
 - How does this compare to a dynamically-linked ring implementation on your desktop machine? Describe your experience in your commentary.