CprE / ComS 583
Reconfigurable Computing

Prof. Joseph Zambreno
Department of Electrical and Computer Engineering
Iowa State University

Lecture #24 – Reconfigurable Coprocessors

Quick Points

- Unresolved course issues
  - Gigantic red bug
  - Bean-leaf beetle?

- This Thursday, project status updates
  - 10 minute presentations per group + questions
  - Combination of Adobe Breeze and calling in to teleconference
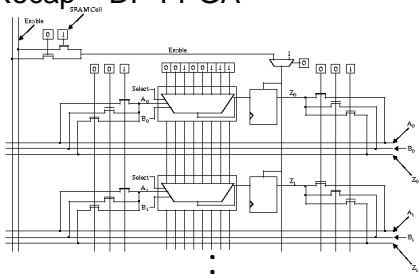 - More details later today

---

Recap – DP-FPGA

- Break FPGA into datapath and control sections
- Save storage for LUTs and connection transistors
- Key issue is grain size
- Cherepacha/Lewis – U. Toronto

Recap – RaPiD
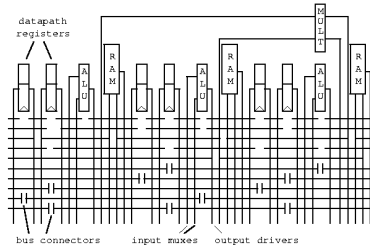
- Segmented linear architecture
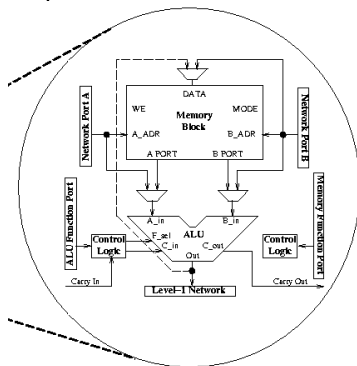- All RAMs and ALUs are pipelined
- Bus connectors also contain registers

Recap – Matrix

- Two inputs from adjacent blocks
- Local memory for instructions, data

Recap – RAW Tile

- Full functionality in each tile
- Static router located for near-neighbor communication

Outline

- Recap
- Reconfigurable Coprocessors
  - Motivation
  - Compute Models
  - Architecture
  - Examples

Overview

- Processors efficient at sequential codes, regular arithmetic operations
- FPGA efficient at fine-grained parallelism, unusual bit-level operations
- Tight-coupling important: allows sharing of data/control
- Efficiency is an issue:
  - Context-switches
  - Memory coherency
  - Synchronization

Compute Models

- I/O pre/post processing
- Application specific operation
- Reconfigurable Co-processors
  - Coarse-grained
  - Mostly independent
- Reconfigurable Functional Unit
  - Tightly integrated with processor pipeline
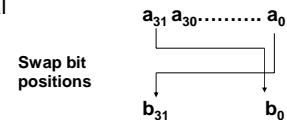 - Register file sharing becomes an issue

Instruction Augmentation

- Processor can only describe a small number of basic computations in a cycle
  - I bits -> 2^I operations
- Many operations could be performed on 2 W-bit words
- ALU implementations restrict execution of some simple operations
  - e. g. bit reversal

$a_{31} \, a_{30} \ldots\ldots\ldots \, a_0$

Swap bit positions

b_{31} b_0

Instruction Augmentation (cont.)

- Provide a way to augment the processor instruction set for an application
- Avoid mismatch between hardware/software
- Fit augmented instructions into data <u>and</u> control stream
- Create a functional unit for augmented instructions
- Compiler techniques to identify/use new functional unit

"First" Instruction Augmentation

- PRISM
  - Processor Reconfiguration through Instruction Set Metamorphosis
- PRISM-I
  - 68010 (10MHz) + XC3090
  - can reconfigure FPGA in one second!
  - 50-75 clocks for operations

PRISM-1 Results

Function Name	Description (input bytes / output bytes)	Compilation Time (mins)	% Utilization of a XC3090 FPGA	Speed-up Factor
Hamming(x,y)	Calculates the hamming metric. (4/2)	6	38%	24
Bitrev(x)	Bit-reversal function. (4/4)	2	0%	26
Neuron(x,y)	Cascadable 4-input N-Net function. (4/4)	12	52%	12
MultAccm(x,y)	Multiply/accumulate function. (4/4)	11	58%	2.9
LogicEv(x)	Logic simulation engine function. (4/4)	12	40%	18
ECC(x,y)	Error correction coder/decoder. (3/2)	6	14%	24
Find_first_1(x)	Find first '1' in input. (4/1)	3	11%	42
Piecewise(x)	5-section piecewise linear seg. (4/4)	24	77%	5.1
ALog2(x)	Computes base-2 A*log(x). (4/4)	16	74%	54

PRISM Architecture

- FPGA on bus
- Access as memory mapped peripheral
- Explicit context management
- Some software discipline for use
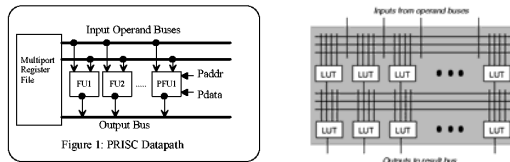- …not much of an "architecture" presented to user

PRISC

- Architecture:
  - couple into register file as "superscalar" functional unit
  - flow-through array (no state)

Figure 1: PRISC Datapath

PRISC (cont.)

- All compiled
- Working from MIPS binary
- <200 4LUTs ?
  - 64x3
- 200MHz MIPS base
- See [RazSmi94A] for more details

Optimization	CPS	EQN	EXP	GCC	L1	SC
PFU-expression	9	0	48	13	4	12
PFU-table-lookup	0	0	0	0	0	0
PFU-predication	0	1	0	13	0	0
PFU-jump	10	0	47	103	0	35
PFU-loop	0	3	0	4	0	0
TOTAL	19	4	95	133	4	47

Table 1: Static PFU optimization instances in SPECint92.

	CPS	EQN	EXP	GCC	L1	SC
Speedup	1.15	1.91	1.16	1.10	1.06	1.12

Table 2: Cycle count speedup for a PRISC-1 microarchitecture with a single PFU resource. The speedup for each application is an arithmetic average (as defined by SPEC) of all of the data sets for that application.
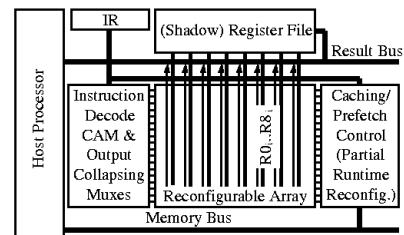
Chimaera

- Start from Prisc idea.
  - Integrate as a functional unit
  - No state
  - RFU Ops (like expfu)
  - Stall processor on instruction miss

- Add
  - Multiple instructions at a time
  - More than 2 inputs possible
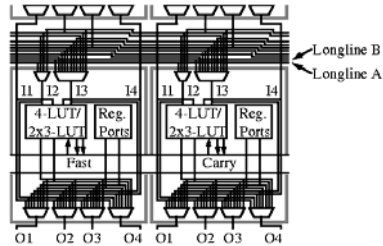
- [HauFry97A]

Chimaera Architecture

- Live copy of register file values feed into array
- Each row of array may compute from register of intermediates
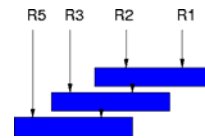- Tag on array to indicate RFUOP

Chimaera Architecture (cont.)

- Array can operate on values as soon as placed in register file
- When RFUOP matches
  - Stall until result ready
  - Drive result from matching row

Chimaera Timing

- If R1 presented late then stall
- Might be helped by instruction reordering
- Physical implementation an issue
- Relies on considerable processor interaction for support

Chimaera Speedup

- Three Spec92 benchmarks
  - Compress   1.11   speedup
  - Eqntott   1.8
  - Life   2.06
- Small arrays with limited state
- Small speedup
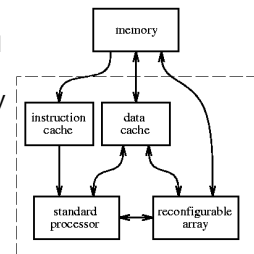- Perhaps focus on global router rather than local optimization

Garp

- Integrate as coprocessor
  - Similar bandwidth to processor as functional unit
  - Own access to memory
- Support multi-cycle operation
  - Allow state
  - Cycle counter to track operation
- Configuration cache, path to memory

Garp (cont.)

- ISA – coprocessor operations
  - Issue gaconfig to make particular configuration present
  - Explicitly move data to/from array
  - Processor suspension during coproc operation
  - Use cycle counter to track progress

- Array may directly access memory
  - Processor and array share memory
  - Exploits streaming data operations
  - Cache/MMU maintains data consistency

Garp Instructions

Instruction	Interlock?	Description
gaconf reg	yes	Load (or switch to) configuration at address given by reg.
mtga reg, array-row-reg, count	yes	Copy reg value to array-row-reg and set array clock counter to count.
mfga reg, array-row-reg, count	yes	Copy array-row-reg value to reg and set array clock counter to count.
gabump reg	no	Increase array clock counter by value in reg.
gastop reg	no	Copy array clock counter to reg and stop array by zeroing clock counter.
gacinv reg	no	Invalidate cache copy of configuration at address given by reg.
cfga reg, array-control-reg	no	Copy value of array control register array-control-reg to reg.
gasave reg	yes	Save all array data state to memory at address given by reg.
garestore reg	yes	Restore previously saved data state from memory at address given by reg.

- Interlock indicates if processor waits for array to count to zero
- Last three instructions useful for context swap
- Processor decode hardware augmented to recognize new instructions

Garp Array

- Row-oriented logic
- Dedicated path for processor/memory
- Processor does not have to be involved in array-memory path

---

Garp Results

Figure 13: Floorplan of the UltraSPARC die, and that of a hypothetical Garp die constructed in the same technology.

Benchmark	167 MHz SPARC	133 MHz Garp	ratio
DES encrypt of 1 MB	3.60 s	0.15 s	24
Dither of 640 × 480 image	160 ms	17 ms	9.4
Sort of 1 million records	1.44 s	0.67 s	2.1

Figure 14: Benchmark results. The times for Garp are obtained from program simulation.

- General results
  - 10-20X improvement on stream, feed-forward operation
  - 2-3x when data dependencies limit pipelining [HauWaw97A]

PRISC/Chimaera vs. Garp

- Prisc/Chimaera
  - Basic op is single cycle: expfu
  - No state
  - Could have multiple PFUs
  - Fine grained parallelism
  - Not effective for deep pipelines

- Garp
  - Basic op is multi-cycle – gaconfig
  - Effective for deep pipelining
  - Single array
  - Requires state swapping consideration

VLIW/microcoded Model

- Similar to instruction augmentation
- Single tag (address, instruction)
  - Controls a number of more basic operations
- Some difference in expectation
  - Can sequence a number of different tags/operations together

---

REMARC

- Array of "nano-processors"
  - 16b, 32 instructions each
  - VLIW like execution, global sequencer
- Coprocessor interface (similar to GARP)
  - No direct array⇔memory

REMARC Architecture

- Issue coprocessor rex
  - Global controller sequences nanoprocessors
  - Multiple cycles (microcode)
- Each nanoprocessor has own I-store (VLIW)

5

Common Theme

- To overcome instruction expression limits:
  - Define new array instructions. Make decode hardware slower / more complicated
  - Many bits of configuration… swap time. An issue -> recall tips for dynamic reconfiguration
- Give array configuration short "name" which processor can call out
- Store multiple configurations in array
- Access as needed (DPGA)

Observation

- All coprocessors have been single-threaded
  - Performance improvement limited by application parallelism
- Potential for task/thread parallelism
  - DPGA
  - Fast context switch
- Concurrent threads seen in discussion of IO/stream processor
- Added complexity needs to be addressed in software

Parallel Computation

- What would it take to let the processor and FPGA run in parallel?

Modern Processors

Deal with:
- Variable data delays
- Dependencies with data
- Multiple heterogeneous functional units
Via:
- Register scoreboarding
- Runtime data flow (Tomasulo)

OneChip

- Want array to have direct memory→memory operations
- Want to fit into programming model/ISA
  - Without forcing exclusive processor/FPGA operation
  - Allowing decoupled processor/array execution
- Key Idea:
  - FPGA operates on memory→memory regions
  - Make regions explicit to processor issue
  - Scoreboard memory blocks

OneChip Pipeline
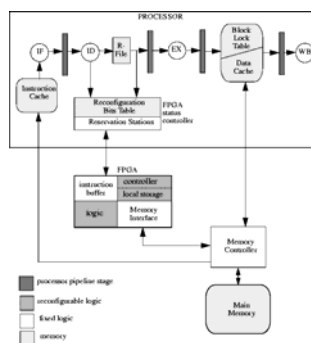
OneChip Instructions

- Basic Operation is:
  - FPGA MEM[Rsource]→MEM[Rdst]
    - block sizes powers of 2

opcode	FPGA function	misc.	R_{source}	R_{dest}	source block size	destination block size
6	4	2	5	5	5	5
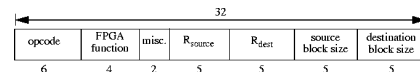
- Supports 14 "loaded" functions
  - DPGA/contexts so 4 can be cached

- Fits well into soft-core processor model

6

OneChip (cont.)

- Basic op is: FPGA MEM→MEM
- No state between these ops
- Coherence is that ops appear sequential
- Could have multiple/parallel FPGA Compute units
  - Scoreboard with processor and each other
- Single source operations?
- Can't chain FPGA operations?

OneChip Extensions

- FPGA operates on certain memory regions only
- Makes regions explicit to processor issue
- Scoreboard memory blocks

	0x0
FPGA	0x1000
Proc	0x10000

Indicates usage of data pages like virtual memory system!

Compute Model Roundup

- Interfacing
- IO Processor (Asynchronous)
- Instruction Augmentation
  - PFU (like FU, no state)
  - Synchronous Coprocessor
  - VLIW
  - Configurable Vector
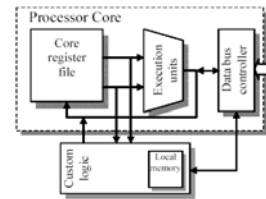- Asynchronous Coroutine/coprocessor
- Memory⇒memory coprocessor

Shadow Registers

- Reconfigurable functional units require tight integration with register file
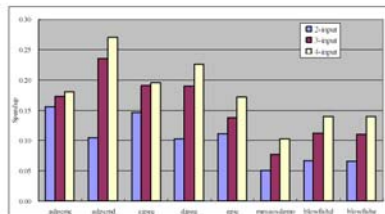- Many reconfigurable operations require more than two operands at a time

Multi-Operand Operations

- What's the best speedup that could be achieved?
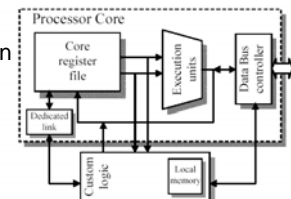 - Provides upper bound
- Assumes all operands available when needed

Additional Register File Access

- Dedicated link – move data as needed
  - Requires latency
- Extra register port – consumes resources
  - May not be used often
- Replicate whole (or most) of register file
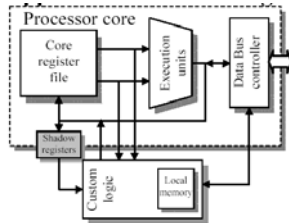 - Can be wasteful

7

Shadow Register Approach

- Small number of registers needed (3 or 4)
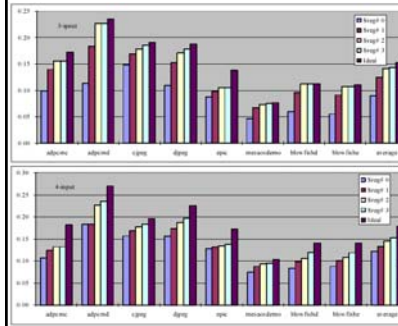- Use extra bits in each instruction
- Can be scaled for necessary port size

Shadow Register Approach (cont.)

- Approach comes within 89% of ideal for 3-input functions
- Paper also shows supporting algorithms [Con99A]

Summary

- Many different models for co-processor implementation
  - Functional unit
  - Stand-alone co-processor
- Programming models for these systems is a key
- Recent compiler advancements open the door for future development
- Need tie in with applications