CS184c: Computer Architecture [Parallel and Multithreaded]

Day 13: May ~~17~~ 22, 2001

Interfacing Heterogeneous Computational Blocks

Previously

- Interfacing Array logic with Processors
  - ease interfacing
  - better cover mix of application characteristics
  - tailor “instructions” to application
- Single thread, single-cycle operations

Instruction Augmentation

- Small arrays with limited state
  - so far, for automatic compilation
    - reported speedups have been small
  - open
    - discover less-local recodings which extract greater benefit

Today

- Continue Single threaded
  - relax single cycle
  - allow state on array
  - integrating memory system
- Scaling?

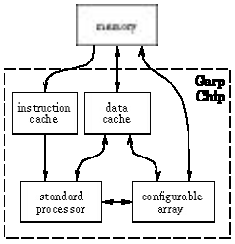
GARP

- Single-cycle flow-through
  - not most promising usage style
- Moving data through RF to/from array
  - can present a limitation
    - bottleneck to achieving high computation rate

[Hauser+Wawrzynek: UCB]

GARP

- Integrate as coprocessor
  - similar bwidth to processor as FU
  - own access to memory
- Support multi-cycle operation
  - allow state
  - cycle counter to track operation
- Fast operation selection
  - cache for configurations
  - dense encodings, wide path to memory



Garp Block Diagram

GARP

- ISA -- coprocessor operations
  - issue `gaconfig` to make a particular configuration resident (may be active or cached)
  - explicitly move data to/from array
    - 2 writes, 1 read (like FU, but not 2W+1R)
  - processor suspend during coproc operation
    - cycle count tracks operation
  - array may directly access memory
    - processor and array share memory space
      - cache/mmu keeps consistent between
    - can exploit streaming data operations

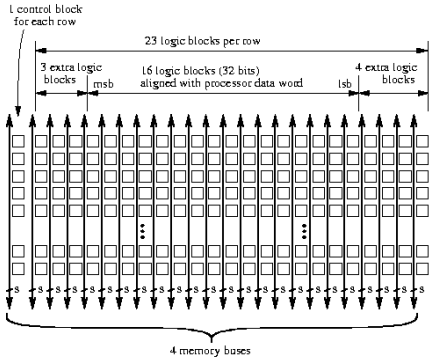
GARP

- Processor 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*.

GARP Array

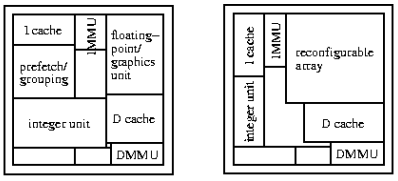
- Row oriented logic
  - denser for datapath operations
- Dedicated path for
  - processor/memory data
- Processor not have to be involved in array⇔memory path



GARP Results

- General results
  - 10-20x on stream, feed-forward operation
  - 2-3x when data-dependencies limit pipelining

[Hauser+Wawrzynek/FCCM97]



(a) UltraSPARC. (b) Hypothetical Garp.

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.

GARP Hand Results

Table 2. Garp's speedups over Ultrasparc for hand-coded functions.

Function	Data size	Speedup	Limiting factor
Image median filter	640 × 480 pixels	43	Compute throughput
DES (ECB mode)	1 Mbyte	18.7	Compute throughput
Image dithering	640 × 480 pixels	17.0	Compute throughput
strlen	1,024 chars	14.2	Memory bandwidth
strlen	16 chars	1.84	Overhead
Sort	2 Mbytes	2.2	Scattered memory accesses

[Callahan, Hauser, Wawrzynek. IEEE Computer, April 2000]

GARP Compiler Results

Table 1. Kernels from a wavelet image compression program.

Kernel	Percentage of original software execution time	Iteration Interval	No. of queues used	ILP (average operations per nonstall cycle)	No. of executions	Average no. of compute cycles per execution (including stalls)	Average no. of overhead cycles per execution	Net speedup over MIPS only
forward_wavelet_696	18.2	2	2	10.0	448	1,176	114	2.1
forward_wavelet_647	13.8	2	2	10.0	448	310	91	5.1
init_image_354	12.8	1	2	8.0	1	65,852	564	12.7
forward_wavelet_711	10.1	2	2	7.0	448	241	59	4.9
entropy_encode_544	10.0	1	1	5.0	1	65,538	989	9.9
forward_wavelet_674	9.3	1	3	13.0	448	128	76	6.6
block_quantize_411	5.5	2	0	5.5	320	353	56	2.8
entropy_encode_557	3.9	6	0	2.8	3,262	31	24	1.4
RLE_encode_509	3.8	1	1	11.0	774	22	48	4.6

[Callahan, Hauser, Wawrzynek. IEEE Computer, April 2000]

PRISC/Chimera … GARP

- PRISC/Chimaera
  - basic op is single cycle: `expfu` (`rfuop`)
  - no state
  - could conceivably have multiple PFUs?
  - Discover parallelism => run in parallel?
  - Can't run deep pipelines
- GARP
  - basic op is multicycle
    - `gaconfig`
    - `mtga`
    - `mfga`
  - can have state/deep pipelining
  - ? Multiple arrays viable?
  - Identify `mtga/mfga` w/ corr `gaconfig`?

Common Theme

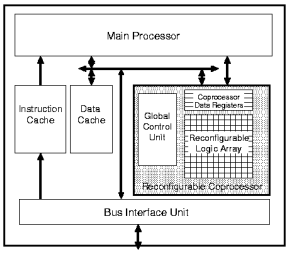
- To get around instruction expression limits
  - define new instruction in array
    - many bits of config … broad expressability
    - many parallel operators
  - give array configuration short "name" which processor can callout
    - …effectively the address of the operation

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

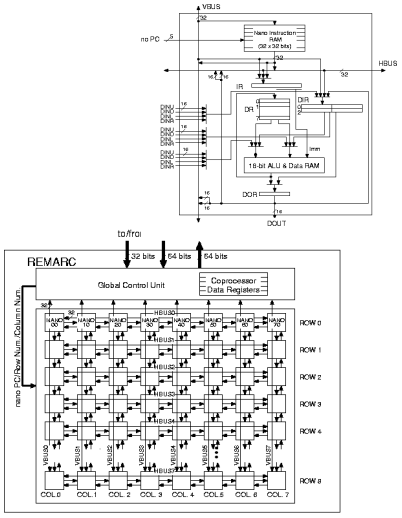
- Array of “nano-processors”
  - 16b, 32 instructions each
  - VLIW like execution, global sequencer
- Coprocessor interface (similar to GARP)
  - no direct array⇔memory



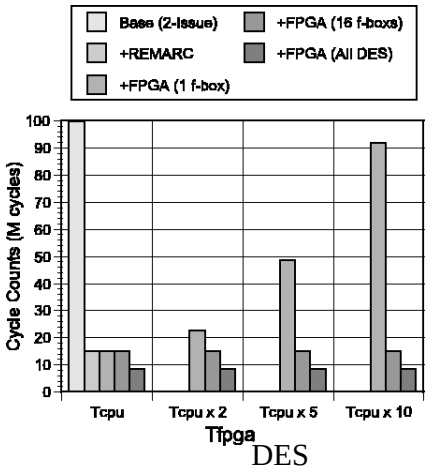
[Olukotun: Stanford]

REMARC Architecture

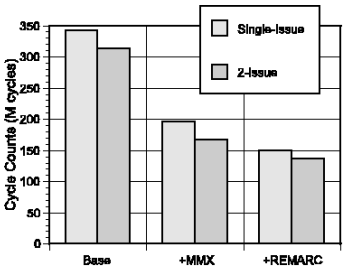
- Issue coprocessor `rex`
  - global controller sequences nanoprocessors
  - multiple cycles (microcode)
- Each nanoprocessor has own I-store (VLIW)



REMARC Results



DES



MPEG2

[Miyamori+Olukotun/FCCM98]

Configurable Vector Unit Model

- Perform vector operation on datastreams
- Setup spatial datapath to implement operator in configurable hardware
- Potential benefit in ability to chain together operations in datapath
- May be way to use GARP/NAPA?
- OneChip (to come…)

Observation

- All single threaded
  - limited to parallelism
    - instruction level (VLIW, bit-level)
    - data level (vector/stream/SIMD)
  - no task/thread level parallelism
    - except for IO dedicated task parallel with processor task

Scaling

- Can scale
  - number of inactive contexts
  - number of PFUs in PRISC/Chimaera
    - but still limited by single threaded execution (ILP)
    - exacerbate pressure/complexity of RF/interconnect
- Cannot scale
  - number of active resources
    - and have automatically exploited

Model: Autonomous Coroutine

- Array task is decoupled from processor
  - fork operation / join upon completion
- Array has own
  - internal state
  - access to shared state (memory)
- NAPA supports to some extent
  - task level, at least, with multiple devices

Processor/FPGA run in Parallel?

- What would it take to let the processor and FPGA run in parallel?
  - And still get reasonable program semantics?

Modern Processors (CS184b)

- Deal with
  - variable delays
  - dependencies
  - multiple (unknown to compiler) func. units
- Via
  - register scoreboarding
  - runtime dataflow (Tomasulo)

Dynamic Issue

- PRISC (Chimaera?)
  - register→register, work with scoreboard
- GARP
  - works with memory system, so register scoreboard not enough

OneChip Memory Interface [1998]

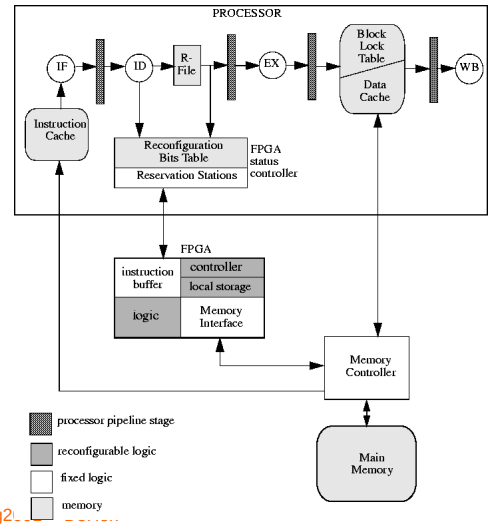
- Want array to have direct memory→memory operations
- Want to fit into programming model/ISA
  - w/out forcing exclusive processor/FPGA operation
  - allowing decoupled processor/array execution

[Jacob+Chow: Toronto]

OneChip

- Key Idea:
  - FPGA operates on memory→memory regions
  - make regions explicit to processor issue
  - scoreboard memory blocks

OneChip Pipeline



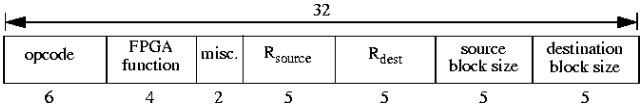
OneChip Coherency

Situation Number	Problem Situation	Actions Taken
1	FPGA read after CPU write	1. Flush FPGA source addresses from CPU cache when FPGA instruction issues 2. Prevent FPGA reads while pending CPU store instructions are outstanding
2	CPU read after FPGA write	3. Invalidate FPGA destination addresses in CPU cache when FPGA instruction issues 4. Prevent CPU reads from FPGA destination addresses until FPGA writes its destination block
3	FPGA write after CPU read	5. Prevent FPGA writes while pending CPU load instructions are outstanding
4	CPU write after FPGA read	6. Prevent CPU writes to FPGA source addresses until FPGA reads its source block
5	FPGA write after CPU write	7. Prevent FPGA writes while pending CPU store instructions are outstanding
6	CPU write after FPGA write	8. Prevent CPU writes to FPGA destination addresses until FPGA writes its destination block

Table 3.14: Actions taken to ensure memory coherence

OneChip Instructions

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



- Supports 14 “loaded” functions
  - DPGA/contexts so 4 can be cached

OneChip

- 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?

To Date...

- In context of full application
  - seen fine-grained/automatic benefits
- On computational kernels
  - seen the benefits of coarse-grain interaction
    - GARP, REMARC, OneChip
- Missing: still need to see
  - full application (multi-application) benefits of these broader architectures...

Model Roundup

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

Models Mutually Exclusive?

- E5/Triscend and NAPA
  - support peripheral/IO
  - not clear have architecture definition to support application longevity
- PRISC/Chimaera/GARP/OneChip
  - have architecture definition
  - time-shared, single-thread prevents serving as peripheral/IO processor

Summary

- Several different models and uses for a “Reconfigurable Processor”
- Some drive us into different design spaces
- Exploit density and expressiveness of fine-grained, spatial operations
- Number of ways to integrate cleanly into processor architecture…and their limitations

Next Time

- Can imagine a more general, heterogeneous, concurrent, multithreaded compute model
- SCORE
  - streaming dataflow based model

Big Ideas

- Model
  - preserving semantics
  - decoupled execution
  - avoid sequentialization / expose parallelism w/in model
    - extend scoreboarding/locking to memory
    - important that memory regions appear in model
  - tolerate variations in implementations
  - support scaling

Big Ideas

- Spatial
  - denser raw computation
  - supports definition of powerful instructions
    - assign short name --> descriptive benefit
    - build with spatial --> dense collection of active operators to support
  - efficient way to support
    - repetitive operations
    - bit-level operations