CS184c: Computer Architecture [Parallel and Multithreaded]

Day 11: May10, 2001

Data Parallel

(SIMD, SPMD, Vector)

Today

- Data Parallel
  - Model
  - Application
  - Resources
  - Architectures
    - Abacus
    - T0

Data Parallel Model

- Perform same computation on multiple, distinct data items
- SIMD
  - recall simplification of general array model
  - every PE get same instruction
    - feed large number of PEs with small instruction bandwidth

CS184a

Architecture Instruction Taxonomy

Control Threads (PCs)	*pinsts* per Control Thread	Instruction Depth	Granularity	Architecture/Examples
0	0	0	n/a	Hardwired Functional Unit (*e.g.* ECC/EDC Unit, FP MPY)
	n	1	1	FPGA
			w	Reconfigurable ALUs
1	1	c	$n_v \cdot 1$	Bitwise SIMD
			w	Traditional Processors
			$n_v \cdot w$	Vector Processors
	n	c	1	DPGA
		8	16	PADDI
		c	w	VLIW
m	n	1	1	HSRA/SCORE
	1	c	$n_v \cdot w$	MSIMD
m	1	c	1	VEGA
		8	16	PADDI-2
		c	w	MIMD (traditional)

Example

- Operations on vectors
  - vector sum
  - dot, cross product
  - matrix operations
- Simulations / finite element...
  - same update computation on every site
- Image/pixel processing
  - compute same thing on each pixel

Model

- Zero, one, infinity
  - good model has unbounded number of processors
  - user allocates virtual processors
  - folded (as needed) to share physical processor se

How do an if?

- Have large set of data
- How do we conditionally deal with data?

ABS example

- Saw hoops had to jump through to compute absolute value w/out conditional branching

Key: Local State

- Set state during computation
- Use state to modify transmitted instruction
  - Could simply be PE.op(inputs,state)
  - Often mask
    - select subset of processors to operate
    - like predicated operations in conventional processor

Local State Op

- Consider 4-LUT with two states
  - w/ local state bit, can implement a 3-LUT function with one state bit
  - state bit is 4th input to LUT can decide which operation to perform

ABS with Mask

- Tmp = val < 0
- rval=val
- mask tmp==true
  rval=-(val)
- unmask

Model

- Model remains
  – all PEs get same operation
  – compute on local state with operation

Synchronization

- Strong SIMD model
  - all operations move forward in lock-step
  - don't get asynchronous advance
  - don't have to do explicit synchronization

Communications

- Question about how general
- Common, low-level
  - nearest-neighbor
  - cheap, fast
  - depends on layout…
  - effect on virtual processors and placement?

Communications

- General network
  - allow model with more powerful shuffling
  - how rich? (expensive)
  - wait for longest operation to complete?
- Use Memory System?

Memory Model?

- PEs have local memory
- Allow PEs global pointers?
- Allow PEs to dereference arbitrary addresses?
  - General communications
  - Including conflicts on PE/bank
    - potentially bigger performance impact in lock-step operation
- Data placement important

Vector Model

- Primary data structure
- Memory access very predictable
  - easy to get high performance on
    - e.g. burst memory fetch, banking
  - one address and get stream of data

How effect control flow?

- Predicated operations take care of local flow control variations
- Sometimes need to effect entire control stream
- E.g. relaxation convergence
  - compute updates to refine some computation
  - until achieve tolerance

Flow Control

- Ultimately need one bit (some digested value) back at central controller to branch upon
- How get?
  - Pick some value calculated in memory?
  - Produce single, aggregate result

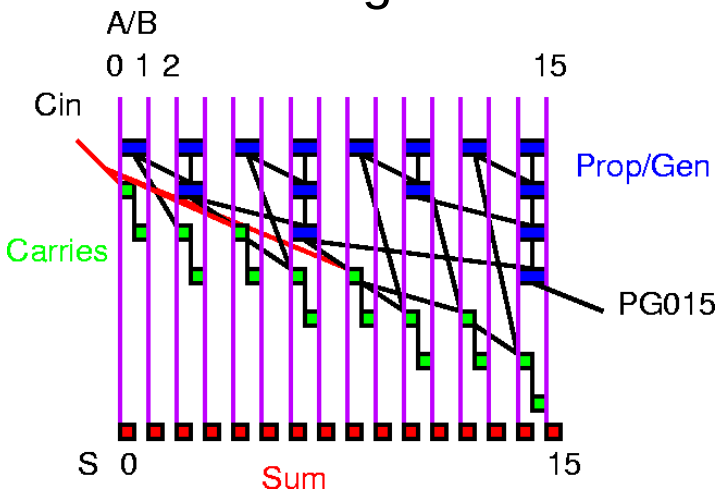
Reduction Value

- Example: summing-or
  - Or together some bit from all Pes
    - build reduction tree….log depth
  - typical usage
    - processor asserts bit when find solution
    - processor deassert bit when solution quality good enough
      - detect when all processors done

Key Algorithm: Parallel Prefix

- Often will want to calculate some final value on aggregate
  - dot product: sum of all pairwise products
  - Karl showed us: saturating sums
    - for example in ADPCM compression
  - Already saw in producing log-depth carries

Resulting RPA



Parallel Prefix

- Calculate **all** intermediate results in log depth
  - e.g. all intermediate carries
  - e.g. all sums to given point in vector
- More general than tree reduction
  - tree reduction (sum, or, and) uses commutativity
  - parallel prefix only requires associativity

Parallel Prefix...

- Count instances with some property
- Parsing
- List operations
  - pointer jumping, find length, matching

Resources

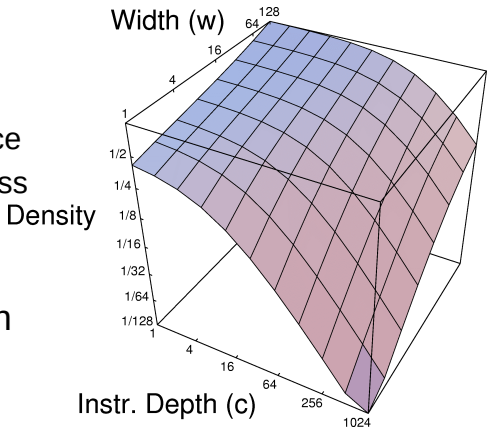
Contrast VLIW/SS

- Single instruction shared across several ALUs
  - (across more bits)
- Significantly lower control
- Simple/predictable control flow
- Parallelism (of data) in model

CS184a

Peak Densities from Model

- Only 2 of 4 parameters
  - small slice of space
  - 100× density across
- Large difference in peak densities
  - large design space!



CS184a

Calibrate Model

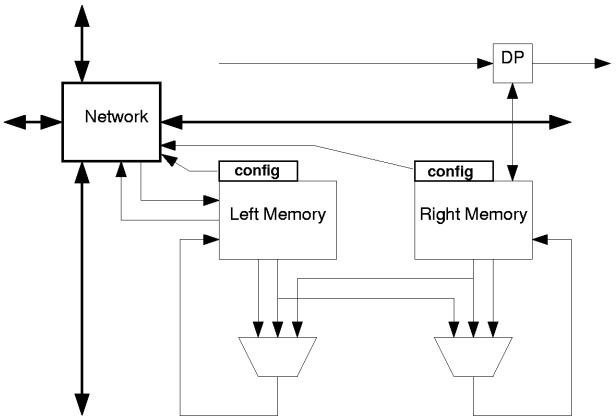
FPGA	**model** $w = 1, d = c = 1, k = 4$	**880K**λ^2
	Xilinx 4K	**630K**λ^2
	Altera 8K	**930K**λ^2
SIMD	**model** $w = 1000, c = 0, d = 64, k = 3$	**170K**λ^2
	Abacus	**190K**λ^2
Processor	**model** $w = 32, d = 32, c = 1024, k = 2$	**2.6M**λ^2
	MIPS-X	**2.1M**λ^2

Examples

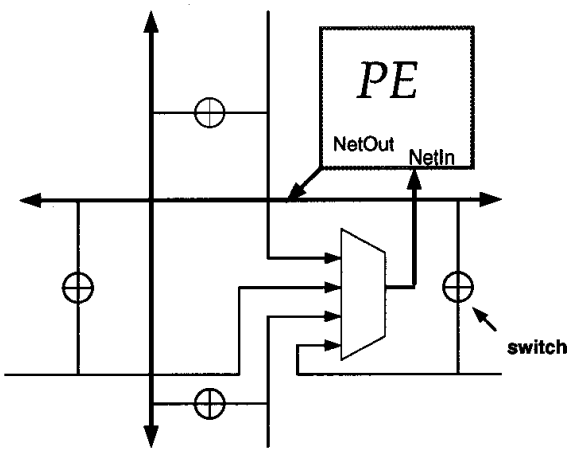
Abacus: bit-wise SIMD

- Collection of simple, bit-processing units
- PE:
  - 2x3-LUT (think adder bit)
  - 64 memory bits, 8 control config
  - active (mask) register
- Network: nearest neighbor with bypass
- Configurable word-size
- [Bolotski et. al. ARVLSI'95]

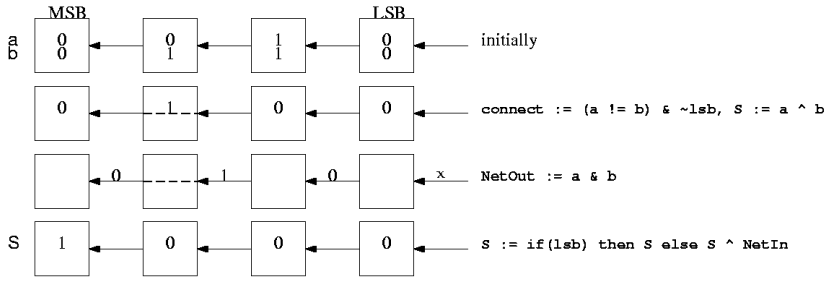
Abacus: PE



Abacus: Network



Abacus: Addition



Abacus: Scan Ops

$$
\begin{array}{lcl}
A & = & [4 \quad 1 \quad 7 \quad 8 \quad 3 \quad 2 \quad 1 \quad 5] \\
+\text{-}scan & = & [4 \quad 5 \quad 12 \quad 20 \quad 23 \quad 25 \quad 26 \quad 31] \\
max\text{-}scan & = & [4 \quad 4 \quad 7 \quad 8 \quad 8 \quad 8 \quad 8 \quad 8] \\
+\text{-}reduce & = & 31
\end{array}
$$

Abacus: bit-wise SIMD

- High raw density:
  - 660 ALU Bit Ops/λ^2-s
- Do have to synthesize many things out of several operations
- Nearest neighbor only

Abacus: Cycles

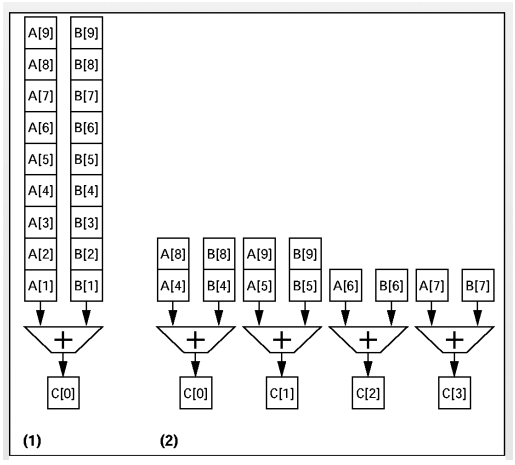
Operation	8-bit		16-bit		32-bit	
	Cycles	GOPS	Cycles	GOPS	Cycles	GOPS
Add	4	4.0	4	2.0	5	0.7
Shift	2	8.0	2	4.0	2	2.0
Accumulate	3	5.2	3	2.6	3	1.3
Move	3	5.2	4	2.0	6	0.6
Compare	6	2.6	11	0.6	12	0.2
Multiply (16×16)					180	0.03

Algorithm	Cycles	Time (μsec)
Edge Detection $\sigma = 1.6$	380	3
Optical Flow, $\Delta = 2$, 5×5 region	3000	24
Surface Reconstruction (1 iteration)	370	3

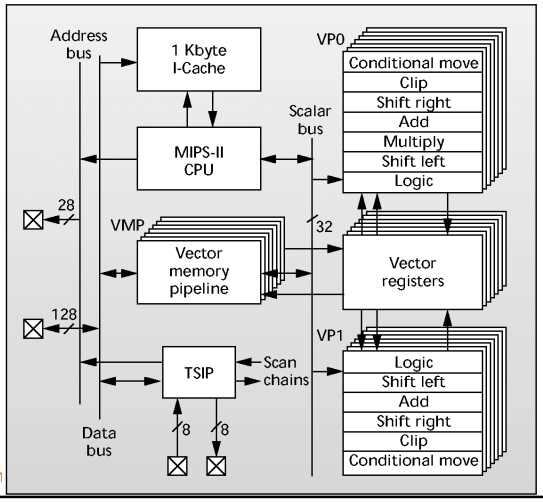
T0: Vector Microprocessor

- Word-oriented vector pipeline
- Scalable vector abstraction
  - vector ISA
  - size of physical vector hardware abstracted
- Communication mostly through memory
- [Asanovic et. al., IEEE Computer 1996]
- [Asanovic et. al., Hot Chips 1996]

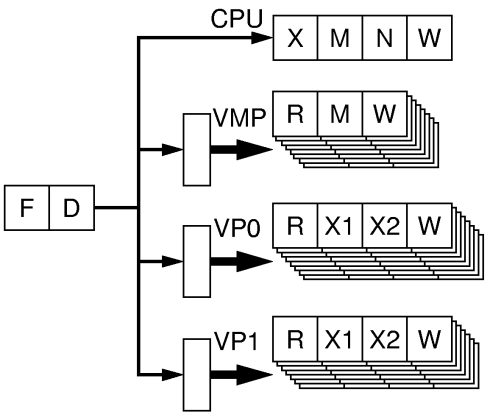
Vector Scaling



T0 Microarchitecture



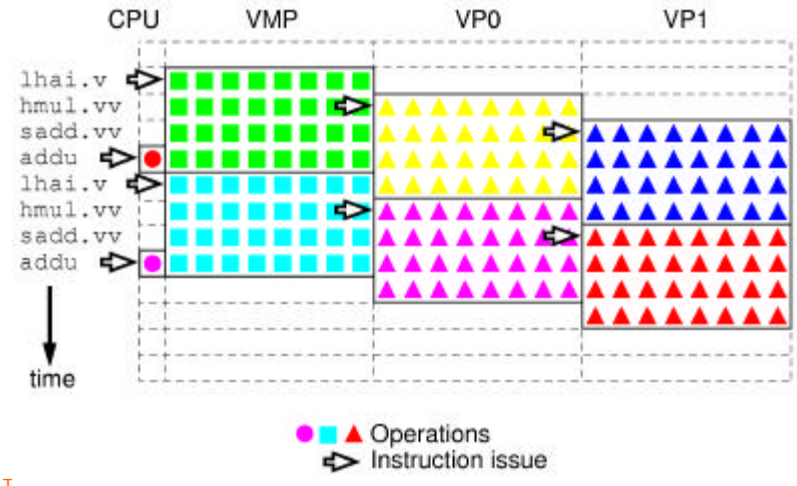
T0 Pipeline



T0 ASM example

```
lhai.v vv1, t0, t1      # Vector load.
hmul.vv vv4, vv2, vv3   # Vector mul.
sadd.vv vv7, vv5, vv7   # Vector add.
addu t2, -1             # Scalar add.
lhai.v vv2, t0, t1      # Vector load.
hmul.vv vv5, vv1, vv3   # Vector mul.
sadd.vv vv8, vv4, vv8   # Vector add.
addu t7, t4             # Scalar add.
```


T0 Execution Example



T0: Vector Microprocessor

- Higher raw density than (super)scalar microprocessors
  - 22 ALU Bit Ops/λ^2-s (vs. <10)
- Clean ISA, scaling
  - contrast VIS, MMX
- Easy integration with existing μP/tools
  - assembly library for vector/matrix ops
  - leverage work in vectorizing compilers

Big Ideas

- Model for computation
  - enables programmer think about machine capabilities a high level
  - abstract out implementation details
  - allow scaling/different implementations
- Exploit structure in computation
  - use to reduce hardware costs