CS184b: Computer Architecture
(Abstractions and Optimizations)

Day 10: April 28, 2003
Vector, SIMD

Today

• Data Parallel
  – Model
  – Application
  – Resources
  – Architectures
    • Abacus
    • T0
Data Parallel Model

- Perform same computation on multiple, distinct data items
  - Sequential set of operations (like ISA)
  - ...but on large aggregate collection
- SIMD
  - recall simplification of general array model
  - every PE get same instruction
    - feed large number of PEs with small instruction bandwidth

Architecture Instruction Taxonomy

<table>
<thead>
<tr>
<th>Control Threads (PCs)</th>
<th>Instructions per Control Thread</th>
<th>Instruction Depth</th>
<th>Granularity</th>
<th>Architecture/Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>n/a</td>
<td>1</td>
<td>FPGA</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Reconfigurable ALUs</td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Bitwise SIMD</td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Traditional Processors</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Vector Processors</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>DPGA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>PADDI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>VLIW</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>HSRA/SCORE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>MIMD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>VEGA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>PADDI-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>MIMD (traditional)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

• Operations on vectors
  – vector sum
  – dot, cross product
  – matrix operations
• Simulations / finite element / cellular automata
  – 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 (data parallel items)
  – user allocates virtual processors
  – folded (as needed) to share physical processors
How do an if?

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

Branchless Multiply Example

- Recall writing multiplication without branching
  - (CS184a, assignment 2)
  - …like hardware/spatial
  - …need to mask and multiplex
**Key: Local State**

- Set state during computation
- Use state to modify transmitted instruction
  - Operation could simply be $\text{PE.op(inputs, state)}$
  - 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

- \( \text{tmp} = \text{val} < 0 \)
- \( \text{rval} = \text{val} \)
- mask all processors with \( \text{tmp} == \text{true} \)
- \( \text{rval} = -\text{val} \)
- 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

- Vector is 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

…not always that simple…

- Often trick to making vector model apply to problems is rich data access
  - Need interconnect to permute data below the vector level
  - Typically:
    - **Gather**: create vector from this set of addresses….
    - **Scatter**: write the vector out to this set of addresses
How effect control flow?  
(SIMD and Vector)

• 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
  – E.g. dot product: sum of all pairwise products
  – Already saw in producing
    • log-depth carries
    • Arbitrary LUT cascades
    • Ultrascalar register updates
CS184a: Day 3

Resulting RPA

A/B
0 1 2

Cin

Carries

Prop/Gen

PG015

CS184a: Day 10

Parallel Prefix Mux-cascade
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
  – Locally identify property
  – Then do prefix sum
• Parsing
• List operations
  – pointer jumping, find length, matching
Terms

• SIMD – Single Instruction Multiple Data
• Vector – SIMD on 1D array of words
• SPMD – Single Program Multiple Data
  – Coined to name the programming model separate from the machine/execution model
  – (may use SPMD model on MIMD machine)

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

Peak Densities from Model

• Only 2 of 4 parameters
  – small slice of space
  – 100× density across
• Large difference in peak densities
  – large design space!
## Calibrate Model

<table>
<thead>
<tr>
<th>Model Type</th>
<th>Model</th>
<th>Equation</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPGA</td>
<td>( w = 1, \ d = c = 1, \ k = 4 )</td>
<td>( 880K\lambda^2 )</td>
<td>Xilinx 4K, Altera 8K</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>630K\lambda^2, 930K\lambda^2</td>
</tr>
<tr>
<td>SIMD</td>
<td>( w = 1000, \ c = 0, \ d = 64, \ k = 3 )</td>
<td>( 170K\lambda^2 )</td>
<td>Abacus</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>190K\lambda^2</td>
</tr>
<tr>
<td>Processor</td>
<td>( w = 32, \ d = 32, \ c = 1024, \ k = 2 )</td>
<td>( 2.6M\lambda^2 )</td>
<td>MIPS-X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2.1M\lambda^2</td>
</tr>
</tbody>
</table>

## 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: Network

Abacus: Addition

\[
\begin{align*}
\text{connect} & := \neg (a \land b) \land \neg \text{lab}, \quad S := a \land b \\
\text{NetOut} & := a \lor b \\
S & := \text{if}(\text{lab}) \text{ then } S \text{ else } S \land \text{NetIn}
\end{align*}
\]
Abacus: Scan Ops

\[
A = [4 \ 1 \ 7 \ 8 \ 3 \ 2 \ 1 \ 5] \\
\text{+}-\text{scan} = [4 \ 5 \ 12 \ 20 \ 23 \ 25 \ 26 \ 31] \\
\text{max-}\text{scan} = [4 \ 4 \ 7 \ 8 \ 8 \ 8 \ 8 \ 8] \\
\text{+}-\text{reduce} = 31
\]

Abacus: bit-wise SIMD

- High raw density:
  - 660 ALU Bit Ops/\lambda^2s
  - Compare peak of 10 bops/\lambda^2s for proc.
  - Compare \sim 100 bops/\lambda^2s for FPGAs
- Do have to synthesize many things out of several operations
- Nearest neighbor communication only
Abacus: Cycles

<table>
<thead>
<tr>
<th>Operation</th>
<th>8-bit</th>
<th>16-bit</th>
<th>32-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cycles</td>
<td>GOPS</td>
<td>Cycles</td>
</tr>
<tr>
<td>Add</td>
<td>4</td>
<td>4.0</td>
<td>4</td>
</tr>
<tr>
<td>Shift</td>
<td>2</td>
<td>8.0</td>
<td>2</td>
</tr>
<tr>
<td>Accumulate</td>
<td>3</td>
<td>5.2</td>
<td>3</td>
</tr>
<tr>
<td>Move</td>
<td>3</td>
<td>5.2</td>
<td>4</td>
</tr>
<tr>
<td>Compare</td>
<td>6</td>
<td>2.6</td>
<td>11</td>
</tr>
<tr>
<td>Multiply (16 × 16)</td>
<td>180</td>
<td>0.03</td>
<td></td>
</tr>
</tbody>
</table>

$1 \, \mu m$ CMOS (6.5mm x 7.3mm, 1000 PEs)

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

No element-to-element dependence:
• Avoid pipeline interlock
• Easy parallel dispatch

Just dependence vector-op to vector-op

T0 Microarchitecture
T0 Pipeline

```
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 ASM example
T0: Vector Microprocessor

- Higher raw density than (super)scalar microprocessors
  - 22 ALU Bit Ops/λ²s (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
Admin

Homework B5

• Group Effort
• Two Weeks
  – …start immediately
  – Set milestone for this week
    • Divide work
    • Get basic measurement and thread/comm primitives running
  – Expect will take some tuning / optimization…
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

• Vector/SIMD – simple model, admits dense implementations
  – How much fits into model?