ESE534: Computer Organization

Day 24: April 16, 2012
Specialization

Previously
• How to support bit processing operations
• How to compose any task
• Instantaneous << potential computation

Today
• What bit operations do I need to perform?
• Specialization
  – Binding Time
  – Specialization Time Models
  – Specialization Benefits
  – Expression

Quote
• The fastest instructions you can execute, are the ones you don’t.
  – …and the least energy, too!

Idea
• Goal: Minimize computation must perform
• Instantaneous computing requirements less than general case
• Opportunity: Some data known or predictable
  – compute minimum computational residue
• As know more data → reduce computation
• Dual of generalization we saw for local control

Preclass 1:
Know More → Less Compute

How does circuit simplify if know A=1?
Possible Optimization

- Once know another piece of information about a computation (data value, parameter, usage limit)
- Fold into computation producing smaller computational residue

Preclass 3

- How many 4-LUTs for 8b-equality compare?
- How many 4-LUTs for 8b compare to constant?
Pattern Match

- Savings:
  - $2N$ bit input computation $\rightarrow N$
  - if $N$ variable, maybe trim unneeded portion
  - state elements store target
  - control load target

Opportunity Exists

- Spatial unfolding of computation
  - can afford more specificity of operation

- Fold (early) bound data into problem

- Common/exceptional cases

```
MATCH_LENGTH bound?  
cread bound?  
Uses per binding?
```
Preclass 4: Circuit

Reconfiguring Logic
- Simple model:
  - Address like memory

Today’s commercial devices:
- Shift configuration in serially
  - Slower but cheaper
- Segmented
  - So can reconfigure only part of the chip at a time

Optimization Prospects
- Area-Time Tradeoff
  - \( T_{spcl} = T_{sc} \times T_{load} \)
  - \( T_{sc} = N \times T_{scycle} \)
  - \( T_{gen} = N \times T_{gcycle} \)
  - \( AT_{gen} = A_{gen} \times T_{gen} \)
  - \( AT_{spcl} = A_{spcl} \times (T_{sc} + T_{load}) \)
- If compute long enough (N large enough)
  - \( T_{sc} \gg T_{load} \) → amortize out load

Preclass 5
- \( T_{load} = 100 \mu s, T_{scycle} = 1 \text{ ns} \)
- \( T_{gload} = 0, T_{gcycle} = 2 \text{ ns} \)
- Ratio \( T_{gtask}/T_{ctask} \) for \( N = 10^6 \) ?
- Breakeven \( N \)?
Optimization Prospects

- **Area-Time Tradeoff**
  - $T_{spcl} = T_{sc} + T_{load}$
  - $T_{sc} = N \times T_{scycle}$
  - $T_{gen} = N \times T_{gcycle}$
  - $AT_{gen} = A_{gen} \times T_{gen}$
  - $AT_{spcl} = A_{spcl} \times (T_{sc} + T_{load})$

- If compute long enough ($N$ large enough)
  - $T_{sc} >> T_{load} \rightarrow$ amortize out load

Opportunity

- With bit level control
  - larger space of optimization than word level

- While true for both spatial and temporal programmables
  - **bigger** effect/benefits for spatial

Multiply Example

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Feature size (μm)</th>
<th>Area and Time</th>
<th>16-16</th>
<th>8-8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom 16-16</td>
<td>0.65 / 0.65</td>
<td>2.9M / 4.8ns</td>
<td>9.5</td>
<td>9.5</td>
</tr>
<tr>
<td>Custom 8-8</td>
<td>0.65 / 0.65</td>
<td>3.9M / 4.5ns</td>
<td>13.3</td>
<td>13</td>
</tr>
<tr>
<td>Gate-Array 16x16</td>
<td>0.75 / 0.75</td>
<td>22M / 20ns</td>
<td>1.3</td>
<td>1.3</td>
</tr>
<tr>
<td>FPGA (SCB)</td>
<td>0.65 / 0.65</td>
<td>312M / 26ns</td>
<td>0.067</td>
<td>0.067</td>
</tr>
<tr>
<td>16b DSP</td>
<td>0.65 / 0.65</td>
<td>310M / 28ns</td>
<td>0.34</td>
<td>0.30</td>
</tr>
<tr>
<td>16b (no multiply)</td>
<td>0.75 / 0.75</td>
<td>312M / 26ns</td>
<td>0.057</td>
<td>0.057</td>
</tr>
</tbody>
</table>

Multiply Show

- Specialization in datapath width
- Specialization in data

Benefit Examples

- Less than
- Multiply revisited
  - more than just constant propagation
- ATR

Benefits

Empirical Examples
Less Than (Bounds check?)

- Area depend on target value
- But all targets less than generic comparison

<table>
<thead>
<tr>
<th>Function size</th>
<th>Speed Mapped</th>
<th>Area Mapped</th>
<th>Speed Mapped</th>
<th>Area Mapped</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CLBs</td>
<td>path</td>
<td>CLBs</td>
<td>path</td>
</tr>
<tr>
<td>1 variable</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a ≤ b</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>(16)</td>
<td>18.5</td>
<td>14.5</td>
<td>16.5</td>
<td>16.5</td>
</tr>
<tr>
<td>(32)</td>
<td>35</td>
<td>20</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>(64)</td>
<td>77.5</td>
<td>20</td>
<td>74.5</td>
<td>28</td>
</tr>
</tbody>
</table>

Multiply

- How savings in a multiply by constant?
  - Multiply by 80?
    - 0101000
  - Multiply by 255?

Multiply (revisited)

- Specialization can be more than constant propagation
- Naïve,
  - save product term generation
  - complexity number of 1’s in constant input
- Can do better exploiting algebraic properties

Multiply

- Never really need more than \([N/2]\) one bits in constant
- Example: multiply by 255:
  - 256x-x = 255x
  - t1=x<<8
  - res=t1-x

Multiply

- At most \([N/2]+2\) adds for any constant
- Exploiting common subexpressions can do better:
  - e.g.
    - c=10101010
    - t1=x+x<<2 (101x)
    - t2=t1<<5+t1<<1
    - 2 adds instead of 4
Multiply Example

### Application metric:
TAPs = filter taps multiply accumulate

\[
Y_i = w_1 x_i + w_2 x_{i+1} + \ldots
\]

### Opportunity Exists
- Spatial unfolding of computation
  - can afford more specificity of operation
  \[
  Y_i = w_1 x_i + w_2 x_{i+1} + \ldots
  \]
- What opportunity do we lose if sequentializing on single multiplier?

### Example: ATR
- Automatic Target Recognition
  - need to score image for a number of different patterns
  - different views of tanks, missiles, etc.
  - reduce target image to a binary template with don't cares
  - need to track many (e.g. 70-100) templates for each image region
  - templates themselves are sparse
    - small fraction of care pixels
- 16x16x2=512 flops to hold single target pattern
- 16x16=256 LUTs to compute match
- 256 score bits→8b score ~ 500 adder bits in tree
- more for retiming
- ~800 LUTs here
- Maybe fit 1 generic template in XC4010 (400 CLBs)?
Example: UCLA ATR

- UCLA
  - specialize to template
  - ignore don’t care pixels
  - only build adder tree to care pixels
  - exploit common subexpressions
  - get 10 templates in a XC4010

[Villasenor et. al./FCCM’96]

Usage Classes

Specialization Usage Classes

- Known binding time
- Dynamic binding, persistent use
  - apparent
  - empirical
- Common case

Known Binding Time

- Sum=0
- For I=0→N
  - Sum+=V[I]
- For I=0→N
  - VN[I]=V[I]/Sum

Scope/Procedure Invocation

Scale(max,min,V)

for I=0→V.length
  - tmp=(V[I]-min)
  - Vres[I]=tmp/(max-min)

Dynamic Binding Time

- cexp=0;
- For I=0→V.length
  - if (V[I].exp!=cexp)
    - cexp=V[I].exp;
  - Vres[I]=V[I].mant<<cexp

Thread 1:
  - a=src.read()
  - if (a.newavg())
    - avg=a.avg()

Thread 2:
  - v=data.read()
  - out.write(v/avg)

Empirical Binding

- Have to check if value changed
  - Checking value O(N) area [pattern match]
  - Interesting because computations
    - can be O(2^N) [Day 12]
    - often greater area than pattern match
  - Also Rent’s Rule:
    - Computation > linear in IO
    - IO=C n^p  \propto n \propto IO^{(1/p)}
Common/Uncommon Case

- For $i=0 \rightarrow N$
  - If ($V[i]==10$)
    - $\text{SumSq}+=V[i]*V[i]$;
  - elseif ($V[i]<10$)
    - $\text{SumSq}+=V[i]*V[i]$;
  - else
    - $\text{SumSq}+=V[i]*V[i]$;
- For $i=0 \rightarrow N$
  - If ($V[i]==10$)
    - $\text{SumSq}+=100$;
  - elseif ($V[i]<10$)
    - $\text{SumSq}+=V[i]*V[i]$;
  - else
    - $\text{SumSq}+=V[i]*V[i]$;

Potential Binding Times

- What are the potential binding times for values?
  - i.e. at what points might values be defined then held constant?

Binding Times

- Pre-fabrication
- Application/algorithm selection
- Compilation
- Installation
- Program startup (load time)
- Instantiation (new ...)
- Epochs
- Procedure
- Loop

Exploitation Patterns

- Full Specialization (Partial Evaluation)
  - May have to run (synth?) p&r at runtime
- Worst-case footprint
  - e.g. multiplier worst-case, avg., this case
- Constructive Instance Generator
- Range specialization (wide-word datapath)
  - data width
- Template
  - e.g. pattern match – only fillin LUT prog.

Opportunity Example

(Lecture ended here)
Experiments

- Applications:
  - UCLA MediaBench:
    - adpcm, epic, g721, gsm, jpeg, mesa, mpeg2
    (not shown today - ghostscript, pegwit, pgp, rasta)
  - gzip, versatility, SPECint95 (parts)
- Compiler optimize → instrument for profiling → run
- analyze variable usage, ignore heap
  - heap-reads typically 0-10% of all bit-reads
  - 90-10 rule (variables) - ~90% of bit reads in 1-20% or bits

Bit-Reads Classification

- regular across programs
  - SCASI, CASI, CBD stddev ~11%
- nearly no activity in variables declared const
- ~65% in constant + signed bits
  - trivially exploited

Constant Bit-Ranges

- 32b data paths are too wide
- 55% of all bit-reads are to sign-bits
- most CASI reads clustered in bit-ranges (10% of 11%)
- CASI+SCASI reads (50%) are positioned:
  - 2% low-order 8% whole-word constant
  - 39% high-order 1% elsewhere

Expression Patterns

- Generators
- Instantiation/Immutable computations
  - (disallow mutation once created)
- Special methods (only allow mutation with)
- Data Flow (binding time apparent)
- Control Flow
  - (explicitly separate common/uncommon case)
- Empirical discovery
Benefits

- Benefits come from reduced area & energy
  - reduced area $\rightarrow$ performance
  - room for more spatial operation
  - maybe less interconnect delay
- Challenge: Fully exploiting, full specialization
  - don’t know how big a block is until see values
  - dynamic resource scheduling

Storage

- Will have to store configurations somewhere
- LUT $\sim$ 250K F$^2$
- Configuration 64+ bits
  - SRAM: 20K F$^2$ (12-13 for parity)
  - Dense DRAM: 1.6K F$^2$ (160 for parity)

Saving Instruction Storage

- Cache common, rest on alternate media
  - e.g. disk, flash
- Compressed Descriptions
- Algorithmically composed descriptions
  - good for regular datapaths
  - think Kolmogorov complexity
- Compute values, fill in template
- Run-time configuration generation

Open

- How much opportunity exists in a given program?
- Can we measure entropy of programs?
  - How constant/predictable is the data compute on?
  - Maximum potential benefit if exploit?
  - Measure efficiency of architecture/implementation like measure efficiency of compressor?

Admin

- FM1 graded
- Next priority: FM2 feedback
- Final week of discussion period
  - Ends April 24th
- Reading for Wednesday online

Big Ideas [MSB]

- Programmable advantage
  - Minimize work by specializing to instantaneous computing requirements
- Savings depends on functional complexity
  - but can be substantial for large blocks
  - close gap with custom?
Big Ideas [MSB-1]

- Several models of structure
  - slow changing/early bound data, common case
- Several models of exploitation
  - template, range, bounds, full special