ESE534: Computer Organization

Day 22: April 16, 2014
Time Multiplexing

Tabula

- March 1, 2010
  - Announced new architecture
- We would say
  - w=1, c=8 arch.

Previously

- Basic pipelining
- Saw how to reuse resources at maximum rate to do the same thing
- Saw how to use instructions to reuse resources in time to do different things
- Saw demand-for and options-to-support data retiming

Today

- Multicontext
  - Review why
  - Cost
  - Packing into contexts
  - Retiming requirements for Multicontext
  - Some components
- [concepts we saw in overview week 2-3, we can now dig deeper into details]

How often is reuse of the same operation applicable?

- In what cases can we exploit high-frequency, heavily pipelined operation?
- …and when can we not?

How often is reuse of the same operation applicable?

- Can we exploit higher frequency offered?
  - High throughput, feed-forward (acyclic)
  - Cycles in flowgraph
    - abundant data level parallelism [C-slow]
    - no data level parallelism
  - Low throughput tasks
    - structured (e.g. datapaths) [serialize datapath]
    - unstructured
  - Data dependent operations
    - similar ops [local control -- next time]
    - dis-similar ops
Structured Datapaths

- Datapaths: same pin/st for all bits
- Can serialize and reuse the same data elements in succeeding cycles
- Example: adder

Preclass 1

- Recall looked at mismatches
  - Width, instruction depth/task length
- Sources of inefficient mapping
  - $W_{\text{task}}=4, L_{\text{task}}=4$ to $W_{\text{arch}}=1, C=1$ architecture?

Preclass 1

- How transform $W_{\text{task}}=4, L_{\text{task}}=4$ (path length from throughput) to run efficiently on $W_{\text{arch}}=1, C=1$ architecture?
- Impact on efficiency?

Throughput Yield

FPGA Model -- if throughput requirement is reduced for wide word operations, serialization allows us to reuse active area for same computation

Throughput Yield

How often is \textbf{reuse} of the \textit{same} operation applicable?

- Can we exploit higher frequency offered?
  - High throughput, feed-forward (acyclic)
  - Cycles in flowgraph
    - abundant data level parallelism [C-slow]
    - no data level parallelism
  - Low throughput tasks
    - structured (e.g. datapaths) [serialize datapath]
    - unstructured
  - Data dependent operations
    - similar ops [local control -- next time]
    - dis-similar ops
Remaining Cases

- Benefit from **multicontext** as well as high clock rate
- *i.e.*
  - cycles, no parallelism
  - data dependent, dissimilar operations
  - low throughput, irregular (can’t afford swap?)

Single Context/Fully Spatial

- When have:
  - cycles and no data parallelism
  - low throughput, unstructured tasks
  - dis-similar data dependent tasks
- Active resources sit idle most of the time
- Waste of resources
- Cannot reuse resources to perform different function, only same

Resource Reuse

- To use resources in these cases
  - must direct to do different things.
- Must be able tell resources how to behave
  - separate instructions (*pinsts*) for each behavior

Preclass 2

- How schedule onto 3 contexts?

Preclass 2

- How schedule onto 4 contexts?

Preclass 2

- How schedule onto 6 contexts?
Example: Dis-similar Operations

Multicontext Organization/Area

- $A_{\text{ctxt}} = 20\text{Kf}^2$
  - dense encoding
- $A_{\text{base}} = 200\text{Kf}^2$

- $A_{\text{ctxt}} : A_{\text{base}} = 1:10$

Preclass 3

- Area:
  - Single context?
  - 3 contexts?
  - 4 contexts?
  - 6 contexts?

Multicontext Tradeoff Curves

- Assume ideal packing: $N_{\text{active}} = N_{\text{total}} / L$

In Practice

Limitations from:
- Scheduling
- Retiming

Scheduling
Scheduling Limitations

- \( N_A \) (active)
  - size of largest stage

- Precedence:
  - can evaluate a LUT only after predecessors have been evaluated
  - cannot always, completely equalize stage requirements

Scheduling

- Precedence limits packing freedom
- Freedom do have
  - shows up as slack in network

Scheduling

- Computing Slack:
  - ASAP (As Soon As Possible) Schedule
    - propagate depth forward from primary inputs
    - depth = 1 + max input depth
  - ALAP (As Late As Possible) Schedule
    - propagate distance from outputs back from outputs
    - level = 1 + max output consumption level
  - Slack
    - slack = \( L + 1 - (\text{depth} + \text{level}) \) [PI depth=0, PO level=0]

Work Slack Example

Preclass 4

- With precedence constraints, and unlimited hardware, how many contexts?

Preclass 5

- Without precedence, how many compute blocks needed to evaluate in 4 contexts?
Preclass 6

• Where can schedule?
  – J
  – D

Preclass 6

• Where can schedule D if J in 3?
• Where can schedule D if J in 2?

Preclass 6

• Where can schedule J if D in 1?
• Where can schedule J if D in 2?
• Where schedule operations?
• Physical blocks?

Reminder (Preclass 1)

Sequentialization

• Adding time slots
  – more sequential (more latency)
  – add slack
    • allows better balance

\[ L=4 \rightarrow N_c=2 \] (4 contexts)
Multicontext Data Retiming

• How do we accommodate intermediate data?

Signal Retiming

• Single context, non-pipelined
  – hold value on LUT Output (wire)
    • from production through consumption
  – Wastes wire and switches by occupying
    • for entire critical path delay L
    • not just for 1/L'th of cycle takes to cross wire segment

  – How show up in multicontext?

Signal Retiming

• Multicontext equivalent
  – need LUT to hold value for each intermediate context

ASCII→Hex Example

Single Context: 21 LUTs @ 880K\(\times 2=18.5 M\)\(\times 2\)

ASCII→Hex Example

• All retiming on wires (active outputs)
  – saturation based on inputs to largest stage

ASCII→Hex Example

Three Contexts: 12 LUTs @ 1040K\(\times 2=12.5 M\)\(\times 2\)
Alternate Retiming

• Recall from last time (Day 21)
  – Net buffer
    • smaller than LUT
  – Output retiming
    • may have to route multiple times
  – Input buffer chain
    • only need LUT every depth cycles

Input Buffer Retiming

• Can only take K unique inputs per cycle
• Configuration depth differ from context-to-context
  – Cannot schedule LUTs in slot 2 and 3 on the same physical block, since require 6 inputs.

Reminder

ASCII → Hex Example

• All retiming on wires (active outputs)
  – saturation based on inputs to largest stage

General throughput mapping:

• If only want to achieve limited throughput
• Target produce new result every t cycles
  1. Spatially pipeline every t stages
cycle = t
  2. retime to minimize register requirements
  3. multicontext evaluation w/in a spatial stage
     try to minimize resource usage
  4. Map for depth (i) and contexts (c)

Benchmark Set

• 23 MCNC circuits
  – area mapped with SIS and Chortle

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Mapped LUTs</th>
<th>Path Length</th>
<th>Circuit</th>
<th>Mapped LUTs</th>
<th>Path Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sep3</td>
<td>46</td>
<td>13</td>
<td>dksa</td>
<td>250</td>
<td>13</td>
</tr>
<tr>
<td>9ymm</td>
<td>123</td>
<td>7</td>
<td>e04</td>
<td>230</td>
<td>9</td>
</tr>
<tr>
<td>CAPP</td>
<td>85</td>
<td>10</td>
<td>flr1m</td>
<td>45</td>
<td>17</td>
</tr>
<tr>
<td>C380</td>
<td>176</td>
<td>21</td>
<td>mresv1</td>
<td>30</td>
<td>6</td>
</tr>
<tr>
<td>atm2</td>
<td>169</td>
<td>19</td>
<td>mresv2</td>
<td>35</td>
<td>8</td>
</tr>
<tr>
<td>ospx8</td>
<td>248</td>
<td>9</td>
<td>idf3</td>
<td>105</td>
<td>10</td>
</tr>
<tr>
<td>ospx7</td>
<td>77</td>
<td>7</td>
<td>sop2</td>
<td>73</td>
<td>9</td>
</tr>
<tr>
<td>b5</td>
<td>46</td>
<td>7</td>
<td>rot</td>
<td>253</td>
<td>16</td>
</tr>
<tr>
<td>crip</td>
<td>121</td>
<td>9</td>
<td>sao2</td>
<td>73</td>
<td>9</td>
</tr>
<tr>
<td>coxic</td>
<td>367</td>
<td>13</td>
<td>vcl2</td>
<td>60</td>
<td>9</td>
</tr>
<tr>
<td>count</td>
<td>46</td>
<td>16</td>
<td>24ml</td>
<td>8</td>
<td>7</td>
</tr>
</tbody>
</table>
Multicontext vs. Throughput

General Theme

• Ideal Benefit
  – e.g. Active=N/C
• Logical Constraints
  – Precedence
• Resource Limits
  – Sometimes bottleneck
• Net Benefit
• Resource Balance

Beyond Area

Only an Area win?

• If area were free, would we always want a fully spatial design?

Communication Latency

• Communication latency across chip can limit designs
• Serial design is smaller → less latency
Optimal Delay for Graph App.

Optimal Delay Phenomena

What Minimizes Energy

Multicontext Energy

Components

DPGA (1995)
Xilinx Time-Multiplexed FPGA

- Mid 1990s Xilinx considered Multi-context FPGA
  - Based on XC4K (pre-Virtex) devices
  - Prototype Layout in F=500nm
  - Required more physical interconnect than XC4K
  - Concerned about power (10W at 40MHz)

[Trimberger, FCCM 1997]

Xilinx Time-Multiplexed FPGA

- Two unnecessary expenses:
  - Used output registers with separate outs
  - Based on XC4K design
    - Did not densely encode interconnect configuration
      - Compare 8 bits to configure input C-Box connection
        - Versus $\log_2(8)=3$ bits to control mux select
        - Approx. 200b pins vs. 64b pins

Tabula

- 8 context, 1.6GHz, 40nm
  - 64b pinsts
- Our model w/ input retiming
  - 1M$\lambda^2$ base
    - 80$\lambda^2$ / 64b pinst Instruction mem/context
    - 40$\lambda^2$ / input-retime depth
  - 1M$\lambda^2+8\times0.12M\lambda^2=2M\lambda^2$ \(\Rightarrow\) 4\times LUTs (ideal)
    - Recall ASCIItoHex 3.4, similar for thput map
  - They claim 2.8\times LUTs

[MPR/Tabula 3/29/2009]

Big Ideas

[MSB Ideas]

- Several cases cannot profitably reuse same logic at device cycle rate
  - cycles, no data parallelism
  - low throughput, unstructured
  - dis-similar data dependent computations
- These cases benefit from more than one instructions/operations per active element
  - $A_{\text{act}}<<A_{\text{active}}$ makes interesting
  - save area by sharing active among instructions

Big Ideas

[MSB-1 Ideas]

- Energy benefit for large $p$
  
- Economical retiming becomes important here to achieve active LUT reduction
  - one output reg/LUT leads to early saturation
  - $c=4\ldots8$, $l=4\ldots6$ automatically mapped designs roughly 1/3 single context size
- Most FPGAs typically run in realm where multicontext is smaller
  - How many for intrinsic reasons?
  - How many for lack of register/CAD support?

Admin

- HW9 today
- Final Exercise
- Reading for Monday on Web