ESE535:
Electronic Design Automation

Day 7: February 9, 2015
Scheduled Operator Sharing

Today

• Sharing Resources
• Area-Time Tradeoffs
• Throughput vs. Latency
• VLIW Architectures
• Scheduling (introduce)
  – Maybe start on

Compute Function

• Compute:
  \[ y = Ax^2 + Bx + C \]
• Assume
  – \( D(Mpy) > D(Add) \)
  – \( A(Mpy) > A(Add) \)

Spatial Quadratic

\[ A(Quad) = 3A(Mpy) + 2A(Add) \]

Latency vs. Throughput

• **Latency**: Delay from inputs to output(s)
• **Throughput**: Rate at which can introduce new set of inputs

Washer/Dryer Example

• 1 Washer Takes 30 minutes
• 1 Dryer Takes 45 minutes
• How long to do one load of wash?
  – \( \rightarrow \) Wash latency
• How long to do 5 loads of wash?
• Wash Throughput?
Pipelining

- Break up the computation graph into stages
  - Allowing us to
  - reuse portions of the graph for new data,
  - while older data is still working its way through the graph
  - Before it has exited graph
- Use registers to isolate regions
- Throughput > (1/Latency)
- Relate liquid in pipe
  - Doesn’t wait for first drop of liquid to exit far end of pipe before accepting second drop

Spatial Quadratic

- D(Quad) = 2*D(Mpy)+D(Add) = 21
- Throughput 1/(2*D(Mpy)+D(Add)) = 1/21
- A(Quad) = 3*A(Mpy) + 2*A(Add) = 32

Synchronous Discipline

- Compute
  - From registers
  - Through combinational logic
  - To new values for registers
- Delay through logic sets a lower bound on the duration of each clock – the clock cycle

Terms

- Latency: Delay from inputs to output(s)
- Cycle Time:
  - Clock period
  - Critical path delay between registers
- Throughput: Rate at which can introduce new set of inputs
  - Typically, inverse of cycle time
- Pipelining: how we separate latency from cycle time

Pipelined Spatial Quadratic

- D(Quad) = 3*D(Mpy) = 30
- Throughput = 1/D(Mpy) = 1/10
- A(Quad) = 3*A(Mpy)+2*A(Add)+6A(Reg) = 35

Quadratic with Single Multiplier and Adder?

- We’ve seen reuse to perform the same operation
  - pipelining
- We can also reuse a resource in time to perform a different role.
  - Here: x*x, A*(x*x), B*x
  - also: (Bx)+c, (A*x*x)+(Bx+c)
Quadratic Datapath

• Start with one of each operation

Multiplexer

• Gate allows us to select data from multiple sources

  • Mux
    – For short

  • Useful when sharing operators

Quadratic Datapath

• Multiplier serves multiple roles
  – \(x \times x\)
  – \(A \times (x \times x)\)
  – \(B \times x\)

  • Use multiplexer to steer data (switch interconnections)
    – \(A \times \) (mux) < \(A \times \) (multiply)

Quadratic Datapath

• Multiplier serves multiple roles
  – \(x \times x\)
  – \(A \times (x \times x)\)
  – \(B \times x\)

  • \(x, x \times x\)
  • \(x, A, B\)

Quadratic Datapath

• Adder serves multiple roles
  – \((Bx) + c\)
  – \((A \times x) + (Bx + c)\)

  • one always mpy output
  • \(C, Bx + C\)
Quadratic Datapath

- Add input register for \( x \)

Cycle Impact?
- Need more cycles
- How about the delay of each cycle?
  - Add mux delay
  - Register setup/hold time, clock skew
  - Limited by slowest operation
  - Cycle?
    - \( D(Mpy)+2D(Mux2) = 10.2 \)

Quadratic Control
- Now, we just need to control the datapath
- What control?
  - Control:
    - LD \( x \)
    - LD \( x^2 \)
    - MA Select
    - MB Select
    - AB Select
    - LD \( Bx+C \)
    - LD \( Y \)

Quadratic Memory Control

1. LD_X
2. MA_SEL=x, MB_SEL[1:0]=x, LD_x^2
3. MA_SEL=x, MB_SEL[1:0]=B
4. AB_SEL=C, MA_SEL=x^2, MB_SEL=A, LD_Bx+C
5. AB_SEL=Bx+C, LD_Y

[Could combine 1 and 5 and do in 4 cycles; analysis that follows assume 5 as shown.]
Quadratic Datapath

- Latency/Throughput/Area?
- Latency: $5*(D(MPY)+D(mux3))=51$
- Throughput: $1/Latency \approx 0.02$
- Area: $A(Mpy)+A(Add)+5*A(Reg) + 2*A(Mux2)+A(Mux3)+A(Imem)=17.5+ A(Imem)$

Quadratic with 2 Mult, 1 Add

- Latency/Throughput/Area?
- Latency: $3*(D(Mpy)+D(Mux))=30.3$
- Throughput: $1/30.3 \approx 0.03$
- Area: $2*A(Mpy)+4*A(Mux)+A(Add) + 3*A(Reg) = 26.5$

Quadratic: Area-Time Tradeoff

<table>
<thead>
<tr>
<th>Design</th>
<th>Area</th>
<th>Throughput</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>3M2A (pipe)</td>
<td>35</td>
<td>0.1</td>
<td>30</td>
</tr>
<tr>
<td>2M1A</td>
<td>26.5</td>
<td>0.03</td>
<td>30.3</td>
</tr>
<tr>
<td>1M1A</td>
<td>17.5</td>
<td>0.02</td>
<td>51</td>
</tr>
</tbody>
</table>

Registers → Memory

- Generally can see many registers
- If # registers >> physical operators
  - Only need to access a few at a time
- Group registers into memory banks

Memory Bank Quadratic

- Store x
- $x^2$
- $B*x$
- $A*x^2; B*x+c$
- $(A*x^2)+(B*x+c)$
**Memory Bank Quadratic**

- Store x
- x\(\times x\)
- B\(\times x\)
- A\(\times x^2\), B\(\times x\)+c
- (A\(\times x^2\))+(B\(\times x\)+c)

**Cycle Impact?**

How cycle changed?

- Add mux delay
- Register setup/hold time, clock skew
- Memory read/write
  - Could pipeline

**Impact**

- When have big operators
  - Like multiplier
- Can share them to reduce area
  - At cost of throughput
  - Maybe at cost of latency, energy
- This gives a rich trade space

**Details**

- At extreme, number of "big" operators is dominant cost
  - Total number for area
  - Number in path for delay
- Does cost additional area, delay to share them
  - sometimes a lower order cost

**VLIW**

- Very Long Instruction Word
- Set of operators
  - Parameterize number, distribution
    - More operators→ less time, more area
    - Fewer operators→ more time, less area
- Memories for intermediate state
**VLIW**

- Very Long Instruction Word
- Set of operators
  - Parameterize number, distribution (X, +, sqrt...
  - More operators \( \rightarrow \) less time, more area
  - Fewer operators \( \rightarrow \) more time, less area
- Memories for intermediate state
- Memory for “long” instructions

**Review**

- Reuse physical operators in time
- Share operators in different roles
- Allows us to reduce area at expense of increasing time
- Area-Time tradeoff
- Pay some sharing overhead
  - Muxes, memory
- VLIW – general formulation for shared datapaths

**Design Automation**

Sets up two problems for us:
- Provisioning
  - (Architecture Selection)
  - End of next week (after…)
- Scheduling
  - Start introducing now
  - Next two lectures
**General Problem**

- Resources are not free
  - Wires, io ports
  - Functional units
    - LUTs, ALUs, Multipliers, ....
  - Memory access ports
  - State elements
    - memory locations
    - Registers
      - Flip-flop
      - loadable master-slave latch
    - Multiplexers (mux)

---

**Trick/Technique**

- Resources can be shared (reused) in time
- Sharing resources can reduce
  - instantaneous resource requirements
  - total costs (area)
- **Pattern**: scheduled operator sharing

---

**Example**

Assume unit delay operators. How many operators do I need to evaluate this computation in ~5 time units.

---

**Sharing**

- Does not have to increase delay
  - w/ careful time assignment
  - can often reduce peak resource requirements
  - while obtaining original (unshared) delay
- **Alternately**: Minimize delay given fixed resources
Scheduling

- **Task**: assign time slots (and resources) to operations
  - **time-constrained**: minimizing peak resource requirements
    - *n.b.* time-constrained, not always constrained to minimum execution time
  - **resource-constrained**: minimizing execution time

Resource-Time Example

<table>
<thead>
<tr>
<th>Time Constraint</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;5 → --</td>
<td></td>
</tr>
<tr>
<td>5 → 4</td>
<td></td>
</tr>
<tr>
<td>6.7 → 2</td>
<td></td>
</tr>
<tr>
<td>&gt;7 → 1</td>
<td></td>
</tr>
</tbody>
</table>

Scheduling Use

- Very general problem formulation
  - HDL/Behavioral → RTL
  - Register/Memory allocation/scheduling
  - Instruction/Functional Unit scheduling
  - Processor tasks
  - Time-Switched Routing
    - TDMA, bus scheduling, static routing
  - Routing (share channel)

Two Types (1)

- **Data independent**
  - graph static
  - resource requirements and execution time
    - independent of data
  - schedule statically
  - maybe bounded-time guarantees
  - typical ECAD problem
Two Types (2)

- **Data Dependent**
  - execution time of operators variable
  - depend on data
  - if cannot bound range of variation
  - must schedule online/dynamically
  - cannot guarantee bounded-time
  - general case (i.e. halting problem)
  - typical "General-Purpose" (non-real-time) OS problem

- **Unbounded Resource Problem**
  - Easy:
    - compute ASAP schedule (next slide)
    - i.e. schedule everything as soon as predecessors allow
    - will achieve minimum time
    - won’t achieve minimum area
      - (meet resource bounds)

ASAP Schedule

As Soon As Possible (ASAP)

- For each input
  - mark input on successor
  - if successor has all inputs marked, put in visit queue
- While visit queue not empty
  - pick node
  - update time-slot based on latest input
  - mark inputs of all successors, adding to visit queue when all inputs marked

ASAP Example

Also Useful to Define ALAP

- As Late As Possible
- Work backward from outputs of DAG
- Also achieve minimum time w/ unbounded resources

Rework Example
ALAP and ASAP

- Difference in labeling between ASAP and ALAP is slack of node
  - Freedom to select timeslot
  - **Class theme:** exploit freedom to reduce costs
- If ASAP=ALAP, no freedom to schedule

ASAP, ALAP, Difference

Big Ideas:

- Scheduled Operator Sharing
- Area-Time Tradeoffs

Admin

- Assignment 2, 3 feedback on canvas
- Assignment 4 due Thursday
- Reading for Wednesday online