ESE534: Computer Organization

Day 8: February 8, 2012
Operator Sharing, Virtualization, Programmable Architectures

Preclass Parity

- How many gates?
- Draw solutions

Previously

- Pipelining – reuse in time for same operation
- Memory
- Memories pack state compactly
  - densely

What is Importance of Memory?

- Radical Hypothesis:
  - Memory is simply a very efficient organization which allows us to store data compactly
    - (at least, in the technologies we’ve seen to date)
  - A great engineering trick to optimize resources
- Alternative:
  - memory is a primary

Today

- Operator Sharing (from Day 4)
- Datapath Operation
- Virtualization
- Memory
  - …continue unpacking the role of memory…

Design Space for Computation

(from Day 4)
Compute Function

- Compute:
  \[ y = Ax^2 + Bx + C \]
- Assume
  - \( D(\text{Mpy}) > D(\text{Add}) \)
    - E.g. \( D(\text{Mpy}) = 24, D(\text{Add}) = 8 \)
  - \( A(\text{Mpy}) > A(\text{Add}) \)
    - E.g. \( A(\text{Mpy}) = 64, A(\text{Add}) = 8 \)

Spatial Quadratic

- \( D(\text{Quad}) = 2D(\text{Mpy}) + D(\text{Add}) = 56 \)
- Throughput \( \frac{1}{2D(\text{Mpy}) + D(\text{Add})} = \frac{1}{56} \)
- \( A(\text{Quad}) = 3A(\text{Mpy}) + 2A(\text{Add}) = 208 \)

Pipelined Spatial Quadratic

- \( D(\text{Quad}) = 3D(\text{Mpy}) = 72 \)
- Throughput \( \frac{1}{3D(\text{Mpy})} = \frac{1}{24} \)
- \( A(\text{Quad}) = 3A(\text{Mpy}) + 2A(\text{Add}) + 6A(\text{Reg}) = 232 \)

Quadratic with Single Multiplier and Adder?

- We’ve seen reuse to perform the same operation
  - pipelining
  - bit-serial, homogeneous datapath
- We can also reuse a resource in time to perform a different role.

Quadratic Datapath

- Start with one of each operation
- HW2.4 showed could just use adders

Multiplication serves multiple roles
- \( x \cdot x \)
- \( A \cdot (x \cdot x) \)
- \( B \cdot x \)
- Will need to be able to steer data (switch interconnections)
Quadratic Datapath

- Multiplier serves multiple roles
  - $x^2$
  - $A \times (x^2)$
  - $B \times x$
- Inputs
  a) $x, x^2$
  b) $x, A, B$

Quadratic Datapath

- Adder serves multiple roles
  - $(Bx) + C$
  - $(A \times x^2) + (Bx + C)$
- Inputs
  - one always mpy output
  - $C, Bx + C$

Quadratic Datapath

- Add input register for $x$

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 $Y$
FSMD

- FSMD = FSM + Datapath
- Stylization for building controlled datapaths such as this (a pattern)
- Of course, an FSMD is just an FSM
  - it's often easier to think about as a datapath
  - synthesis, place and route tools have been notoriously bad about discovering/exploiting datapath structure

Quadratic FSMD Control

- S0: if (go) LD_X; goto S1
  - else goto S0
- S1: MA_SEL=x, MB_SEL[1:0]=x, LD_x*x
  - goto S2
- S2: MA_SEL=x, MB_SEL[1:0]=B
  - goto S3
- S3: AB_SEL=C, MA_SEL=x*x, MB_SEL=A
  - goto S4
- S4: AB_SEL=Bx+C, LD_Y
  - goto S0

Quadratic FSM

- D(mux3)=D(mux2)=1
- A(mux2)=2
- A(mux3)=3
- A(QFSM) =~ 10
- Latency/Throughput/Area?
- Latency: 5*(D(MPY)+D(mux3)) = 125
- Throughput: 1/Latency = 1/125
- Area: A(Mpy)+A(Add)+5*A(Reg) +2*A(Mux2)+A(Mux3)+A(QFSM) = 109

Universal Sharing
Review

- Given a task: \( y = Ax^2 + Bx + C \)
- Saw how to share primitive operators
- Got down to one of each

Very naively

- Might seem we need one of each different type of operator

..But

- Doesn’t fool us
- We already know that \textit{nand} gate
  (and many other things—HW1.3)
  …. are universal
- So, we know, we can build a universal compute operator

Temporal Composition

- Don’t have to implement all the gates \textit{at once}
- Can \textit{reuse} one gate over time

Temporal Decomposition

- Take Set of gates
- Sort topologically
  – All predecessors before successors
- Give a unique number to each gate
  – Hold value of its outputs
- Use a memory to hold the gate values
- Sequence through gates
Preclass

• Number gates

Programming?

• How do we program this netlist?

Programming?

• Program gates
  – Tell each gate where to get its input
    • Tell gate n where its two inputs come from
    • Specify the memory location for the output of the associated gate
  – Each gate operation specified with
    • two addresses (the input sources for gate)
    • This is the *instruction* for the gate
**nor2 Memory/Datapath**

![Diagram of nor2 Memory/Datapath](image)

**Supply Instruction**

- How can we supply the sequence of instructions to program this operation?

**Simplest Programmable Control**

- Use a memory to “record” control instructions
- “Play” control with sequence

**Temporal Gate Architecture**

![Diagram of Temporal Gate Architecture](image)

**How program preclass computation?**

- How would we program the preclass computation?
  - Complete the memory

**Simulate the Logic**

- For Preclass
  - Go around the room calling out:
    - Identify PC
    - Identify instruction
      - Perform nor2 on slot ___ and slot ___
      - Result is ___
      - Store into slot ___
What does this mean?

- With only one **active** component
  - **nor** gate
- Can implement **any** function
  - given appropriate
    - state (memory)
    - muxes (interconnect)
    - Control

Defining Terms

*Fixed Function:*  
- Computes one function (e.g. FP-multiply, divider, DCT)  
- Function defined at fabrication time

*Programmable:*  
- Computes "any" computable function  
  (e.g. Processor, DSPs, FPGAs)  
- Function defined after fabrication

Result

- Can sequence together primitive operations in time  
- **Communicating** state through memory  
  - Memory as interconnect  
- To perform “arbitrary” operations

“Any” Computation? (Universality)

- Any computation which can “fit” on the programmable substrate  
- **Limitations:** hold entire computation and intermediate data

Temporal-Spatial Variation

- Can have any number of gates  
  - Tradeoff Area for Reduce Time….

Use of Memory?

- What did we use memory for here?  
- State  
- Instructions  
- Interconnect
“Stored Program” Computer/Processor

• Can build a datapath that can be programmed to perform any computation.
• Can be built with limited hardware that is reused in time.
• Historically: this was a key contribution from Penn’s Moore School
  – Computer Engineers: Eckert and Mauchly
  – ENIAC → EDVAC
  – (often credited to Von Neumann)

What have we done?

• Taken a computation: \( y = Ax^2 + Bx + C \)
• Turned it into operators and interconnect

Virtualization

• We’ve virtualized the computation
• No longer need one physical compute unit for each operator in original computation
• Can suffice with:
  1. shared operator(s)
  2. a description of how each operator behaved
  3. a place to store the intermediate data between operators

Why Interesting?

• Memory compactness
  • This works and was interesting because
    – the area to describe a computation, its interconnect, and its state
    – is much smaller than the physical area to spatially implement the computation
  • e.g. traded multiplier for
    – few memory slots to hold state
    – few memory slots to describe operation
    – time on a shared unit (adder, gate)
•给我们做的

• 我们可以将它实现为一个计算:
  \( y = Ax^2 + Bx + C \)
• 将它转化为操作和互连

虚拟化

• 我们已经虚拟化了这个计算
• 不再需要一个物理计算单元来执行原始计算中的每个操作
• 可以满足:
  1. 共享操作器
  2. 每个操作器的行为的描述
  3. 中间数据存储的地点

为什么有趣?

• 内存紧凑性
  • 这些工作和有趣是因为
    – 描述一个计算、其互连和状态的区域
    – 比物理上实现计算的区域小得多
  • 例如，用乘法器交易
    – 几个内存槽来存储状态
    – 几个内存槽来描述操作
    – 共享单元上的时间（加法器，门）
Big Ideas
[MSB Ideas]

• Memory: efficient way to hold state
  …and allows us to describe/implement computations of unbounded size
• State can be << computation [area]
• Resource sharing: key trick to reduce area
• Memory key tool for Area-Time tradeoffs
• “configuration” signals allow us to generalize the utility of a computational operator

Big Ideas
[MSB-1 Ideas]

• First programmable computing unit
• Two key functions of memory
  – retiming (interconnect in time)
  – instructions
    • description of computation

Questions?

Admin

• HW4 due Monday
• No new reading for Monday