CS184a: Computer Architecture (Structure and Organization)

Day 8: January 24, 2005
Computing Requirements and Instruction Space

Previously

- Fixed and Programmable Computation
- Area-Time-Energy Tradeoffs
- VLSI Scaling

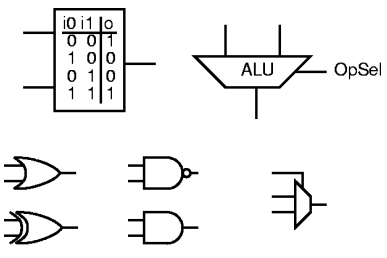
Today

- Computing Requirements
- Instructions
  - Requirements
  - Taxonomy
  - Model Architecture [if time permits]
    - implied costs
    - gross application characteristics

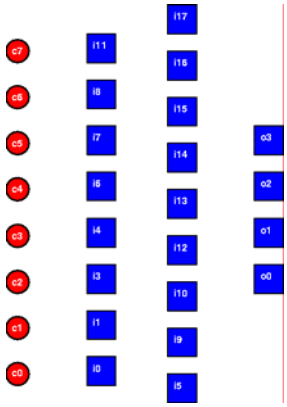
Computing Requirements (review)

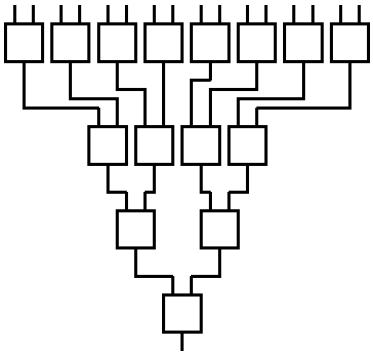
Requirements

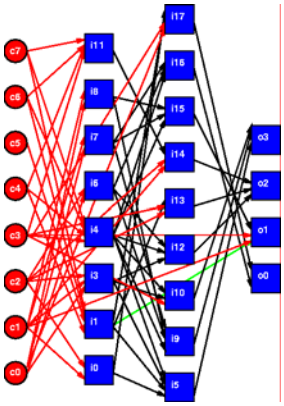
- In order to build a **general-purpose** *(programmable)* computing device, we absolutely must have?
  - _
  - _
  - _
  - _
  - _



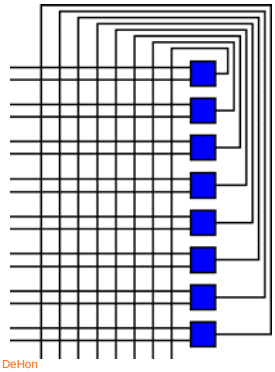
Primitive compute elements enough?



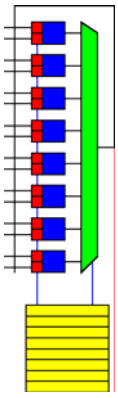




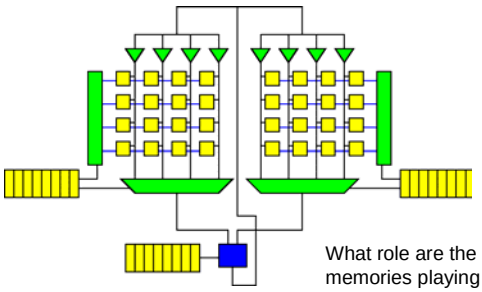
Compute and Interconnect



Sharing Interconnect Resources



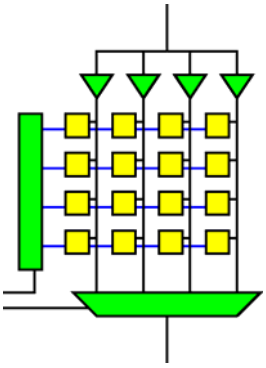
Sharing Interconnect and Compute Resources



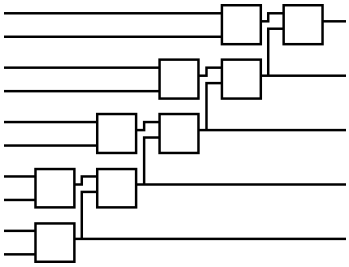
What role are the memories playing here?

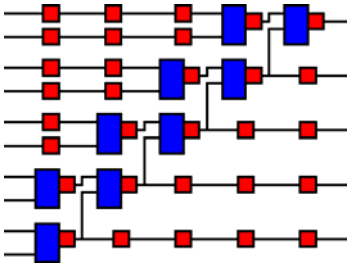
Memory block or
Register File

Interconnect:
moves data from input to storage cell;
or from storage cell to output.



What do I need to be able to use this circuit properly?
(reuse it on different data?)





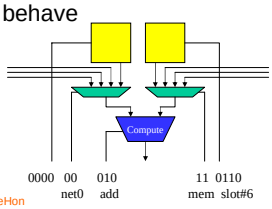
Requirements

- In order to build a **general-purpose** *(programmable)* computing device, we absolutely must have?
  - Compute elements
  - Interconnect: space
  - Interconnect: time (retiming)
  - Interconnect: external (IO)
  - Instructions

Instruction Taxonomy

Instructions

- Distinguishing feature of programmable architectures?
  - *Instructions* -- bits which tell the device how to behave

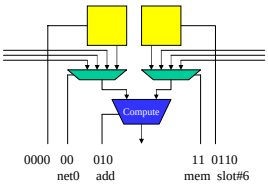


Focus on Instructions

- Instruction organization has a large effect on:
  - size or compactness of an architecture
  - realm of efficient utilization for an architecture

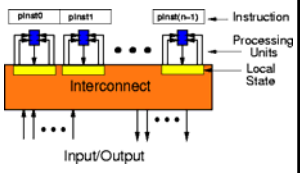
Terminology

- **Primitive Instruction (*pinst*)**
  - Collection of bits which tell a single bit-processing element what to do
  - Includes:
    - select **compute** operation
    - input sources in space
      - (**interconnect**)
    - input sources in time
      - (**retiming**)



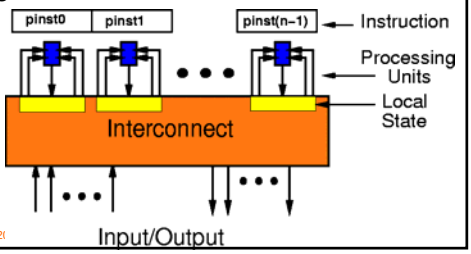
Computational Array Model

- Collection of computing elements
  - compute operator
  - local storage/retiming
- Interconnect
- Instruction



"Ideal" Instruction Control

- Issue a new instruction to every computational bit operator on every cycle



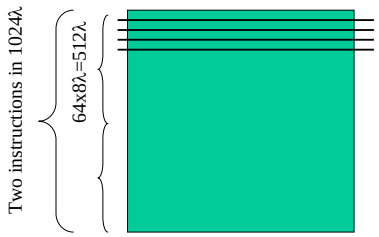
"Ideal" Instruction Distribution

- Why don't we do this?

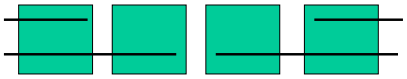
"Ideal" Instruction Distribution

- **Problem:** Instruction bandwidth (and storage area) quickly dominates everything else
  - Compute Block ~ $1M\lambda^2$ ($1K\lambda$ x $1K\lambda$)
  - Instruction ~ 64 bits
  - Wire Pitch ~ 8λ
  - Memory bit ~ $1.2K\lambda^2$

Instruction Distribution

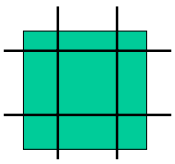


Instruction Distribution



Distribute from both sides = 2x

Instruction Distribution



Distribute X and Y = 2x

Instruction Distribution

- Room to distribute 2 instructions across PE per metal layer (1024 = 2×8×64)
- Feed top and bottom (left and right) = 2×
- Two complete metal layers = 2×

- ⇒ 8 instructions / PE Side

Instruction Distribution

- Maximum of 8 instructions per PE side
- Saturate wire channels at $8\times\sqrt{N} = N$
- ⇒ at 64 PE
  - beyond this:
    - instruction distribution dominates area
- Instruction consumption goes with area
- Instruction bandwidth goes with perimeter

Instruction Distribution

- Beyond 64 PE, instruction bandwidth dictates PE size

$$\frac{\sqrt{PE_{area}}\times 4\times\sqrt{N}}{(64\times 8\lambda)} = N$$

$PE_{area} = 16K\lambda^2\times N$

- As we build larger arrays
  ⇒processing elements become less dense

Instruction Memory Requirements

- **Idea:** put instruction memory in array
- **Problem:** Instruction memory can quickly dominate area, too
  - Memory Area = 64×1.2Kλ^2/instruction
  - PE_{area} = 1Mλ^2 + (Instructions) × 80Kλ^2

Instruction Pragmatics

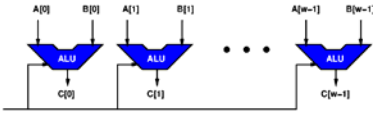
- Instruction requirements *could* dominate array size.
- Standard architecture trick:
  - Look for **structure** to exploit in "typical computations"

Typical Structure?

- What structure do we usually expect?

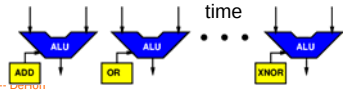
Two Extremes

- SIMD Array (microprocessors)
  - Instruction/cycle
  - share instruction across array of PEs
  - uniform operation in space
  - operation variance in time



Two Extremes

- SIMD Array (microprocessors)
  - Instruction/cycle
  - share instruction across array of PEs
  - uniform operation in space
  - operation variance in time
- FPGA
  - Instruction/PE
  - assume temporal locality of instructions (same)
  - operation variance in space
  - uniform operations in time

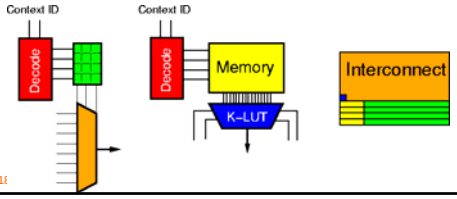


Placing Architectures

- What programmable architectures (organizations) are you familiar with?

Hybrids

- VLIW (SuperScalar)
  - Few *pinsts*/cycle
  - Share instruction across *w* bits
- DPGA
  - Small instruction store / PE



Architecture Instruction Taxonomy

Control Threads (PCs)	*pinsts* per Control Thread	Instruction Depth	Granularity	Architecture/Examples
0	0	0	n/a	Hardwired Functional Unit (e.g. ECC/EDC Unit, FP MPY)
			1	FPGA
	n	1	w	Reconfigurable ALUs
			$n_r \cdot 1$	Bitwise SIMD
	1	c	w	Traditional Processors
			$n_r \cdot w$	Vector Processors
1		c	1	DPGA
	n	8	16	PADDI
		c	w	VLIW
m	n	1	1	HSRA/SCORE
	1	c	$n_r \cdot w$	MSIMD
		c	1	VEGA
m	1	8	16	PADDI-2
		c	w	MIMD (traditional)

Instruction Message

- Architectures fall out of:
  - general model too expensive
  - structure exists in common problems
  - exploit structure to reduce resource requirements
- Architectures can be viewed in a unified design space

Quotes

- *If it can't be expressed in figures, it is not science; it is opinion.* -- Lazarus Long

Modeling

- Why do we model?

Motivation

- Need to understand
  - How costly (big) is a solution
  - How compare to alternatives
  - Cost and benefit of flexibility

What we really want:

- Complete implementation of our application
- For each architectural alternatives
  - In same implementation technology
  - w/ multiple area-time points

Reality

- Seldom get it packaged that nicely
  - much work to do so
  - technology keeps moving
- Deal with
  - estimation from components
  - technology differences
  - few area-time points

Modeling Instruction Effects

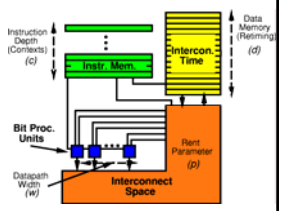
- Restrictions from "ideal" save area
- Restriction from "ideal" limits usability (yield) of PE
- Want to understand effects
  - area model
  - utilization/yield model

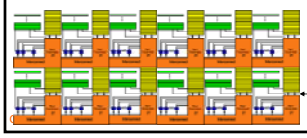
Efficiency/Yield Intuition

- What happens when
  - Datapath is too wide?
  - Datapath is too narrow?
  - Instruction memory is too deep?
  - Instruction memory is too shallow?

Computing Device

- Composition
  - Bit Processing elements
  - Interconnect: space
  - Interconnect: time
  - Instruction Memory

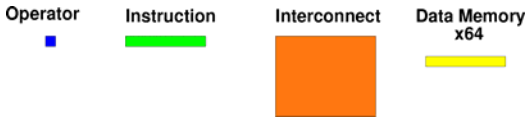




Tile together to build device

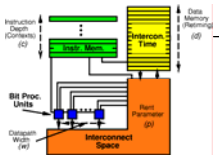
Relative Sizes

- Bit Operator 10-20Kλ^2
- Bit Operator Interconnect 500K-1Mλ^2
- Instruction (w/ interconnect) 80Kλ^2
- Memory bit (SRAM) 1-2Kλ^2



Model Area

$$A_{bit\_elm} = A_{fixed} + \underbrace{N_{SW}(N_p, w, p) \cdot A_{SW}}_{\text{interconnect}} + \underbrace{\left(\frac{c}{w}\right) \cdot n_{ibits} \cdot A_{mem\_cell}}_{\text{instruction memory}} + \underbrace{d \cdot A_{mem\_cell}}_{\text{retiming memory}}$$

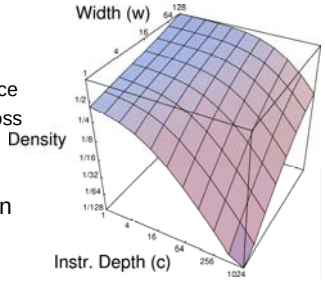


Calibrate Model

FPGA	model $w = 1, d = c = 1, k = 4$	880Kλ^2
	Xilinx 4K	630Kλ^2
	Altera 8K	930Kλ^2
SIMD	model $w = 1000, c = 0, d = 64, k = 3$	170Kλ^2
	Abacus	190Kλ^2
Processor	model $w = 32, d = 32, c = 1024, k = 2$	2.6Mλ^2
	MIPS-X	2.1Mλ^2

Peak Densities from Model

- Only 2 of 4 parameters
  - small slice of space
  - 100× density across
- Large difference in peak densities
  - large design space!



Efficiency

- What do we want to maximize?
  - Useful work per unit silicon
  - (not potential/peak work)
- Yield Fraction / Area
- (or minimize (Area/Yield))

Efficiency

- For comparison, look at relative efficiency to ideal.
- Ideal = architecture exactly matched to application requirements
- Efficiency = A_{ideal}/A_{arch}
- A_{arch} = Area Op/Yield

Efficiency Calculation

$$\text{Efficiency} = \frac{A_{matched\_arch}}{A_{arch}}$$

E.g.

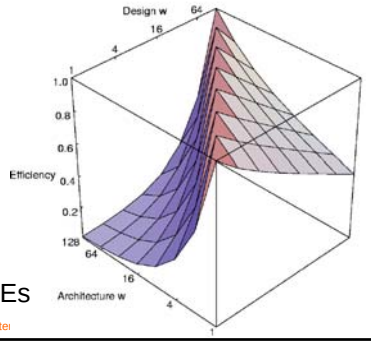
If $w_{task} > w_{arch}$:

$$\text{Efficiency} = \frac{w_{task} \times A_{bit\_elm}|_{w=w_{task}}}{\left\lceil \frac{w_{task}}{w_{arch}} \right\rceil \times w_{arch} \times A_{bit\_elm}|_{w=w_{arch}}}$$

If $w_{task} < w_{arch}$:

$$\text{Efficiency} = \frac{w_{task} \times A_{bit\_elm}|_{w=w_{task}}}{w_{arch} \times A_{bit\_elm}|_{w=w_{arch}}}$$

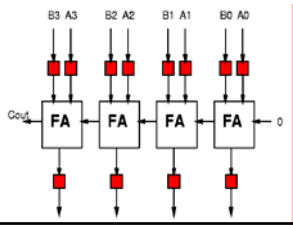
Efficiency: Width Mismatch



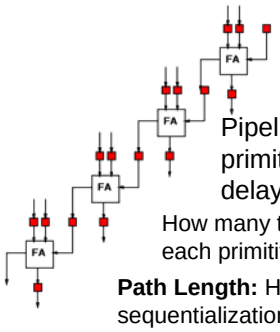
c=1,
16K PEs

Path Length

- How many primitive-operator delays before can perform next operation?
  - Reuse the resource



Reuse

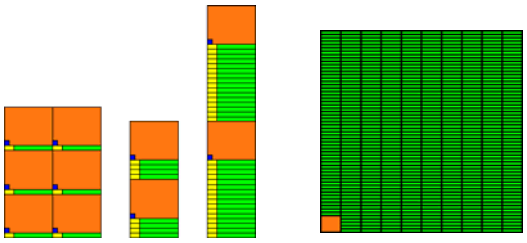


Pipeline and reuse at primitive-operator delay level.

How many times can I reuse each primitive operator?

Path Length: How much sequentialization Is allowed (required)?

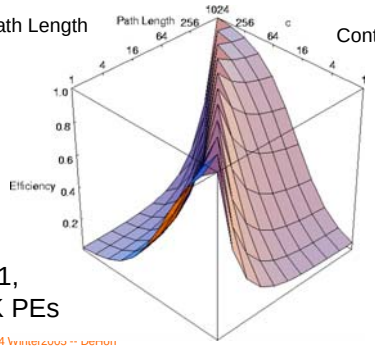
Context Depth



Efficiency with fixed Width

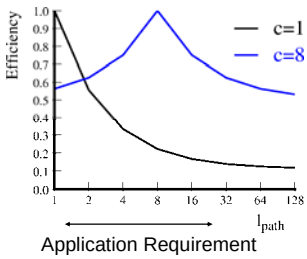
Path Length

Context Depth



w=1,
16K PEs

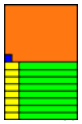
Ideal Efficiency (different model)



Application Requirement

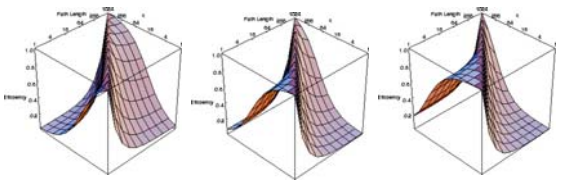
Two resources here:
- active processing elements
- operation description/state

Applications need in different proportions.



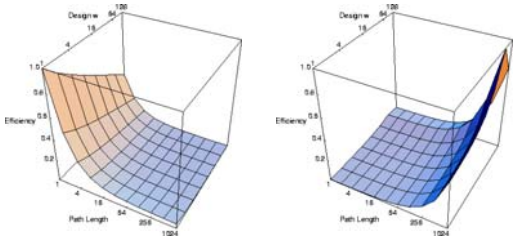
Robust point: $c \cdot A_{ctx} = A_{base}$

Robust Point depend on Width



w=1 w=8 w=64

Processors and FPGAs

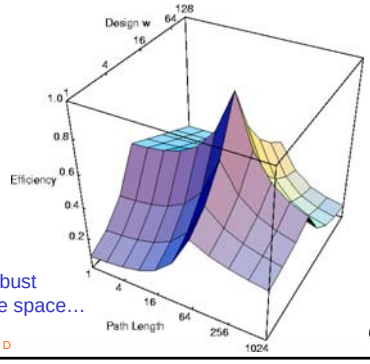


FPGA
c=d=1, w=1, k=4

"Processor"
c=d=1024, w=64, k=2

Intermediate Architecture

w=8
c=64
16K PEs



Hard to be robust
across entire space…

Caveats

- Model abstracts away many details which are important
  - interconnect (day 12--17)
  - control (day 21)
  - specialized functional units (next time)
- Applications are a heterogeneous mix of characteristics

Modeling Message

- Architecture space is **huge**
- Easy to be very inefficient
- Hard to pick one point robust across entire space
- Why we have so many architectures?

General Message

- Parameterize architectures
- Look at continuum
  - costs
  - benefits
- Often have competing effects
  - leads to maxima/minima

Big Ideas [MSB Ideas]

- Basic elements of a programmable computation
  - Compute
  - Interconnect
    - (space and time, outside system [IO])
  - Instructions
- Instruction resources can be significant
  - dominant/limiting resource

Big Ideas [MSB Ideas]

- Applications typically have structure
- Exploit this structure to reduce resource requirements
- Architecture is about understanding and exploiting structure and costs to reduce requirements

Big Ideas [MSB-1 Ideas]

- Two key functions of memory
  - retiming
  - instructions
    - description of computation

Big Ideas [MSB Ideas]

- Instruction organization induces a design space (taxonomy) for programmable architectures
- Arch. structure and application requirements mismatch $\Rightarrow$ inefficiencies
- Model $\Rightarrow$ visualize efficiency trends
- Architecture space is huge
  - can be very inefficient
  - need to learn to navigate