Array Processing and Reconfigurable Computing

(BRASS)

André DeHon
<amd@CS.Berkeley.edu>

Topics

- Array Processing
- Architectural Space
- Reconfigurable Computing
  (Processor+Array Hybrids (BRASS))

Array Processing – Outline

- Start with a different model of computation
- Explore limitations and costs
- Characterize architectural space
  - costs
  - efficiency

Back to Basics
Components of a General-Purpose Computation

- **Compute** – generate a result from several inputs (*e.g.* gate, ALU bit, lookup table)
- **Communicate** – move data from producer to consumer
  - **In space** – between producing and consuming processor (*e.g.* bus, crossbar, network)
  - **In time** – producing (arriving) cycle to consuming cycle (*e.g.* register, RF, memory, disk)

Terminology

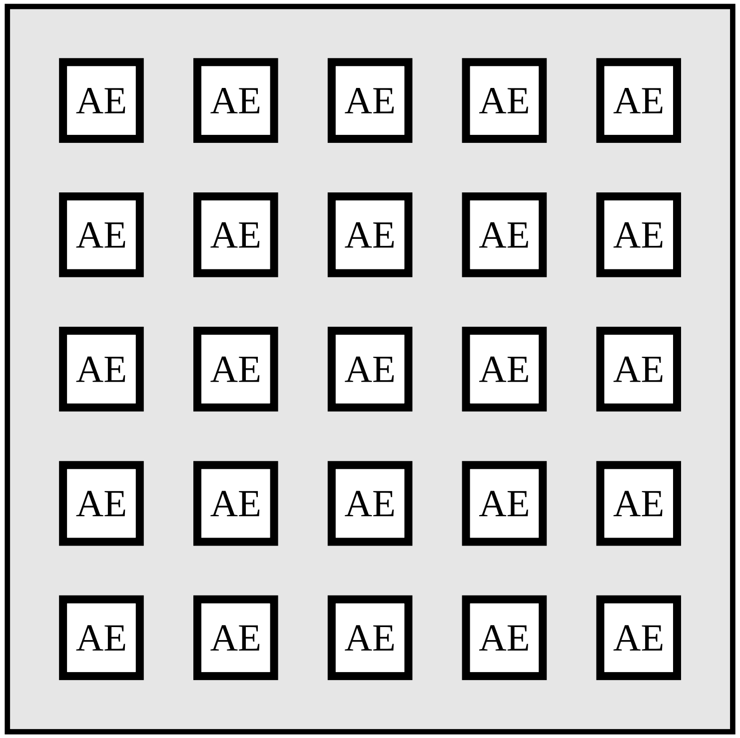
Instruction: **The collection of bits which tells a bit-processing element how to perform.**

includes:

- **selection of compute operation**
- **input sources in space (interconnect)**
- **input sources in time (retiming)**

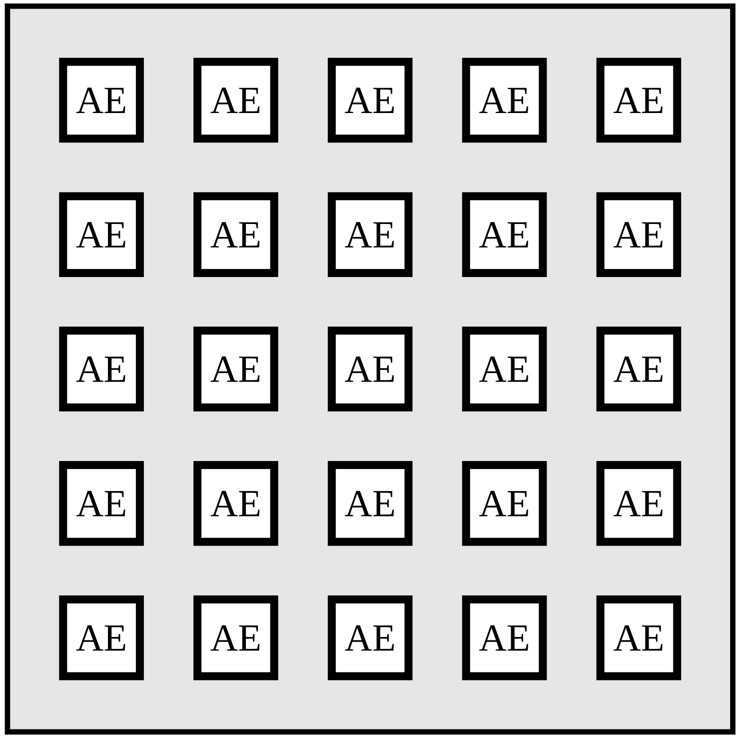
Array Processing Model

- Lattice of AEs
  - Computational Unit
  - Local storage
- Interconnection resources
- *Instruction* controls
  - AE operation
  - Interconnect
  - State manipulation



Array Processing – Ideal Model

Ideal – **Issue a new instruction to every AE on every cycle**



Array Processing – Ideal BW

Problem: Instruction bandwidth (and memory) can quickly dominate

Consider:

$$
\begin{aligned}
\text{AE} &\approx 1\text{M}\lambda^2\ (\approx 1\text{K}\lambda \times 1\text{K}\lambda) \\
\text{Instruction} &\approx 64 \text{ bits} \\
\text{Wire pitch} &\approx 8\lambda \\
\text{Memory bit} &\approx 1.2\text{K}\lambda^2
\end{aligned}
$$

Array Processing – Ideal BW

2	Room to distribute two instructions across each AE per metal layer (64 bits $\times$ 8λ/bit = 512λ)
$\times$2	Assume two complete metal layers (extreme) distributing in both X and Y
$\times$2	Assume top half of array fed from top side and bottom from bottom side *i.e.* half the array elements from each side

8/AE	total across side

Saturate wire channels when: $8 \times \sqrt{N} = N$

64 Processing elements!

Array Processing – Ideal BW

Beyond 64 processors, instruction bandwidth *dictates* processing element size.

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

$$\textbf{AE (area)} = 16K\lambda^2 \times N$$

Array Processing – Ideal Memory Requirements

Idea: Put instruction memory in array

Problem: Instruction memory can quickly dominate

Memory Area = $64 \times 1.2\mathbf{K}\lambda^2$ **/ instruction**

$$A_{AE} = \underbrace{1\mathbf{M}\lambda^2}_{\text{base AE (area)}} + \left(N_{inst} \times \underbrace{80\mathbf{K}\lambda^2}_{\text{area/instruction}} \right)$$

Array Processing – Pragmatics

Instruction requirements *could* dominate array.

Standard architecture trick:

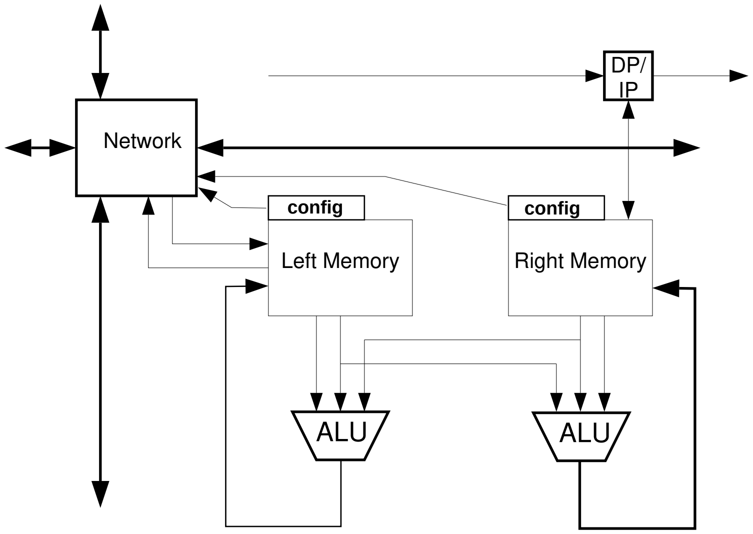
Look for structure to exploit in
"typical computations"

Array Processing – Two Extremes

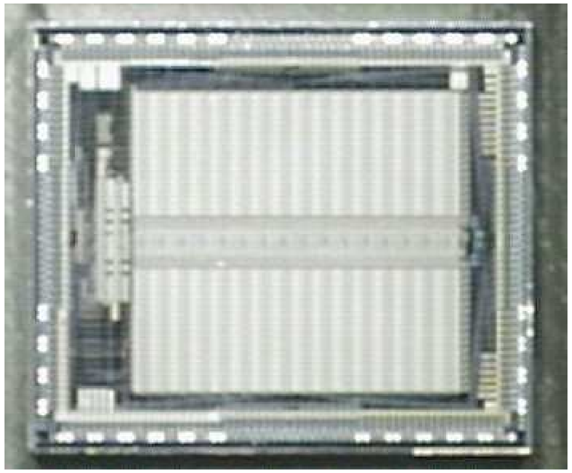
- SIMD Array
  - *Instruction*/cycle
  - share instruction across array
  - uniform in space
- FPGA
  - *Instruction*/AE
  - (extreme) temporal locality
  - uniform in time
  - Slow “programming” phase

SIMD Example – Abacus

- Pair of 3-LUTs for AE (LUT = Look Up Table)
- Fast carry support
- Reconfigurable datapaths
- 64 bit RF/AE

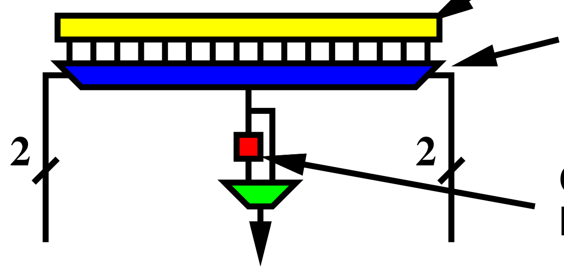


- 1000 PE's
- 8 ns cycle
- 1μ CMOS, 1995
- $6.5\text{mm}\times 7.3\text{mm}$



<http://www.ai.mit.edu/projects/abacus/home_page.html>

FPGA Example – Xilinx XC4000



- 4-LUT + fast carry support (LUT = Look Up Table)
- Optional FF on each output

- XC4025 – 2048 AEs
- 0.6μ, 1994, 11mm$\times$11mm (?)
- 7 ns LUT$\rightarrow$LUT delay

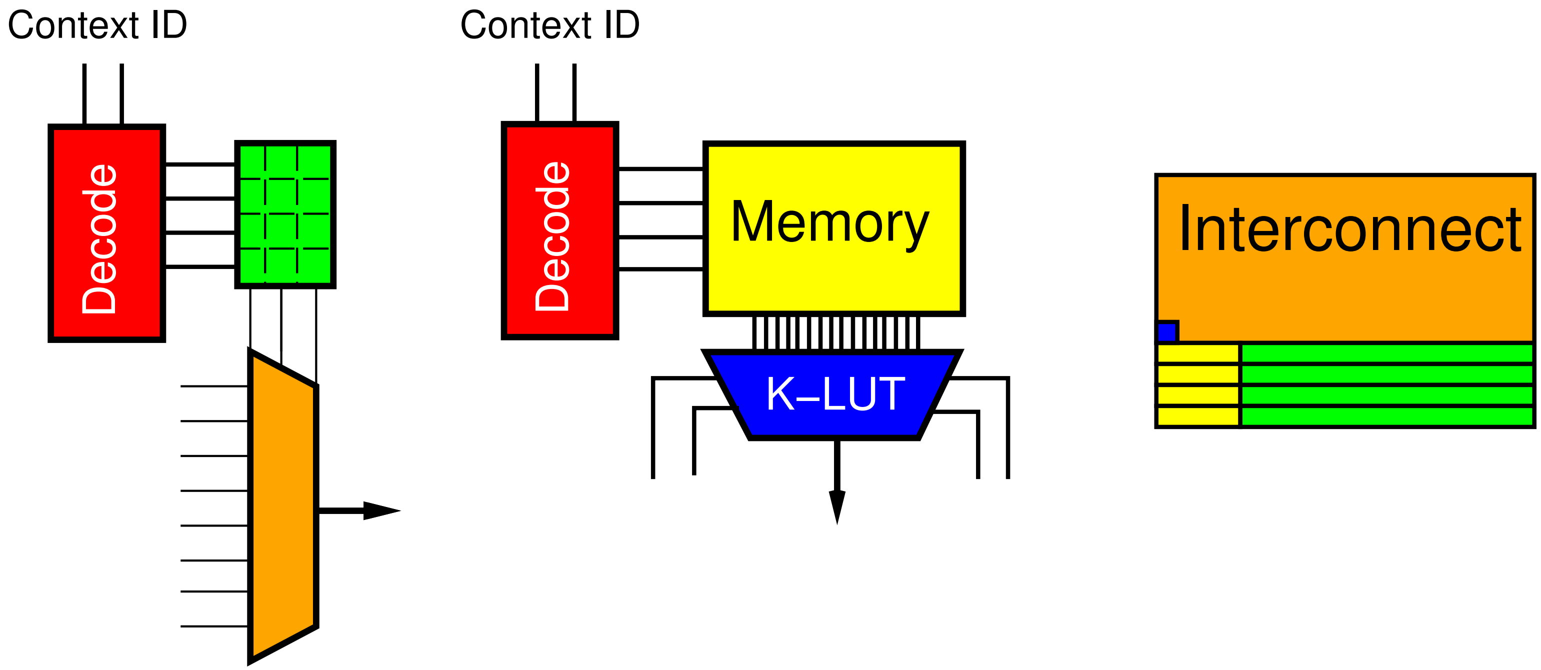
<http://www.xilinx.com/partinfo/4000.pdf>

Array Processing – Two Intermediates

- **VLIW**
  - *Instruction*/**cycle**
  - **Share instruction across w bits**

- **DPGA**
  - **Small instruction-store/AE**

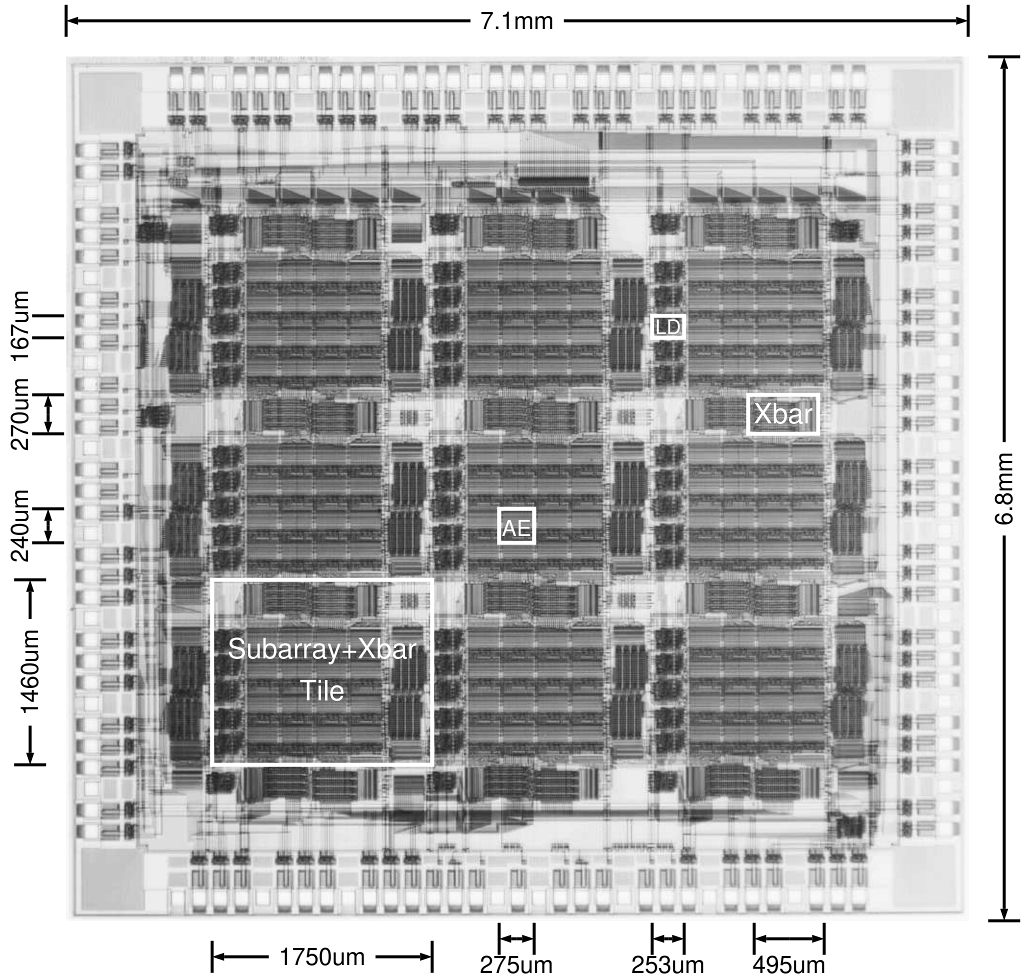
DPGA



- 4-context FPGA
- Single cycle context switch
- Started as intro VLSI project (MIT 6.371)

DPGA

4 contexts
144 AEs
1995, 1μ CMOS
$6.8\text{mm} \times 7.1\text{mm}$
7 ns LUT$\rightarrow$LUT delay
2.5 ns context read



<http://www.ai.mit.edu/projects/transit/dpga_prototype_documents.html>

Architecture Instruction Taxonomy

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

Administrative Break

Architectural Parameter Implications

What happens when:

- Datapath is too wide?
- Datapath is too narrow?
- Instruction memory is too deep?
- Instruction memory is too shallow?

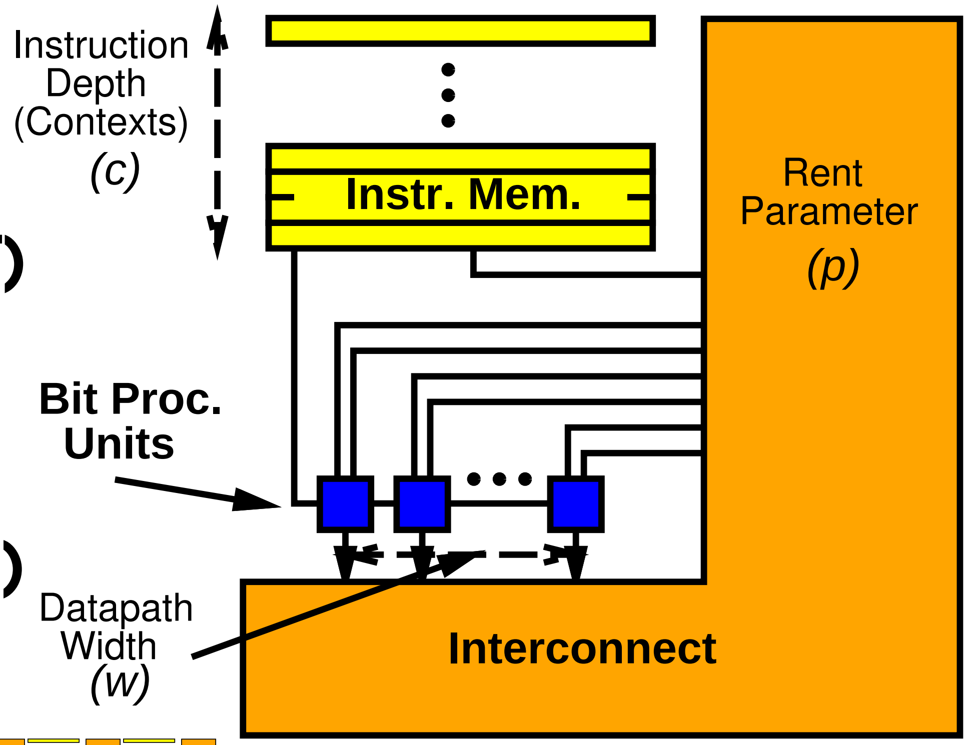
Build a Model

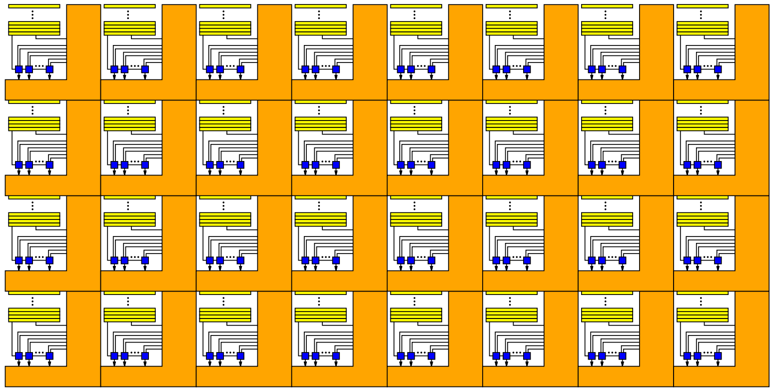
- Device Area
- Device Utilization

Computing Device Composition

Composition:

- **Bit Processing Units** (*e.g.* ALU bit, 4-LUT)
- **Interconnect**
  - Spatial
  - Temporal (retiming)
- **Instruction Memory**





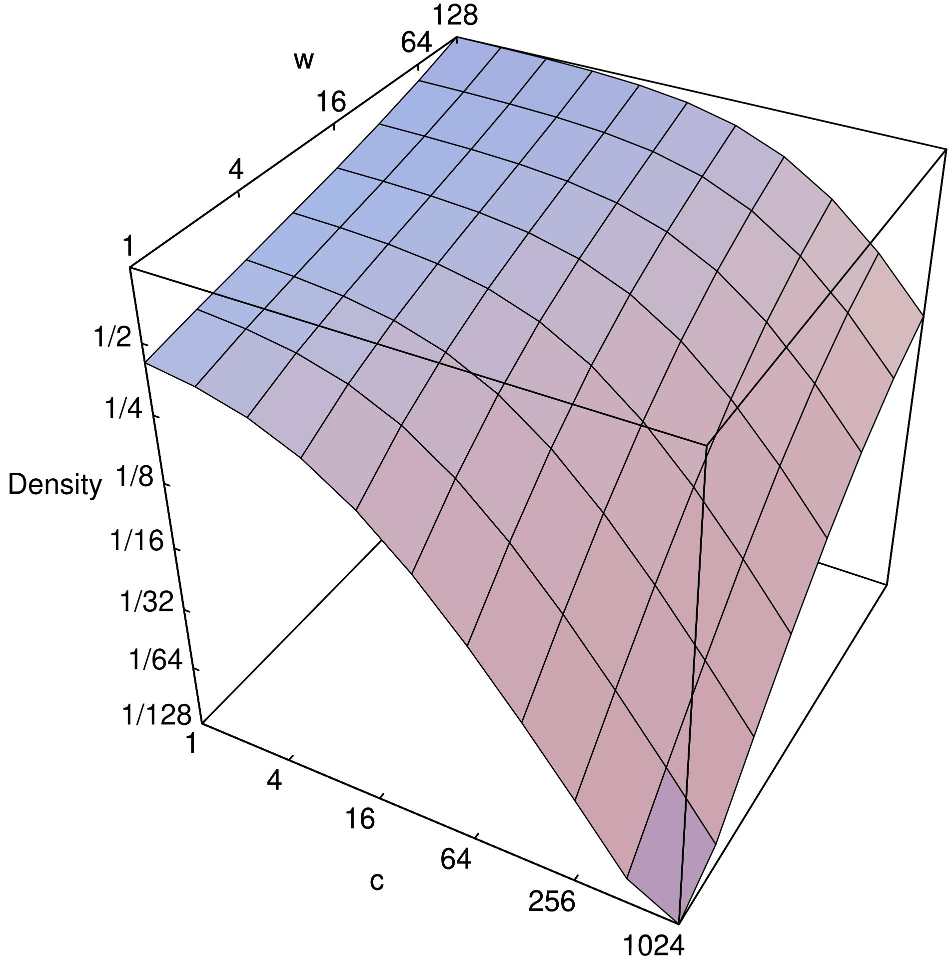
Model Area

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

Spot Check Model

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

Peak Computational Density from Model



(at 16K Bit Processing Elements)

Empirical Peak Computational Densities

Year	Design	Organization	λ^2 area	cycle	$\frac{ge's}{\lambda^2 \cdot s}$
Microprocessors					
1987	MIPS-X	1×32	68M	50ns	19
1995	Alpha	2×64	4.8G	3.3ns	18
Vector					
1995	T0	$1 \times 32 + 16 \times 32$	1.1G	22ns	45
Reconfigurable ALUs					
1995	PADDI-2	48×16	515M	20ns	150
FPGAs					
1992	Xilinx 4K	49 CLBs (2 4-LUT)	61M	7ns	230
SIMD					
1995	Abacus	1000 (2 3-LUT)	190M	8ns	660

Efficiency

Architectural parameters (c, w) have duals in tasks. Peak efficiency when matched.

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

E.g.

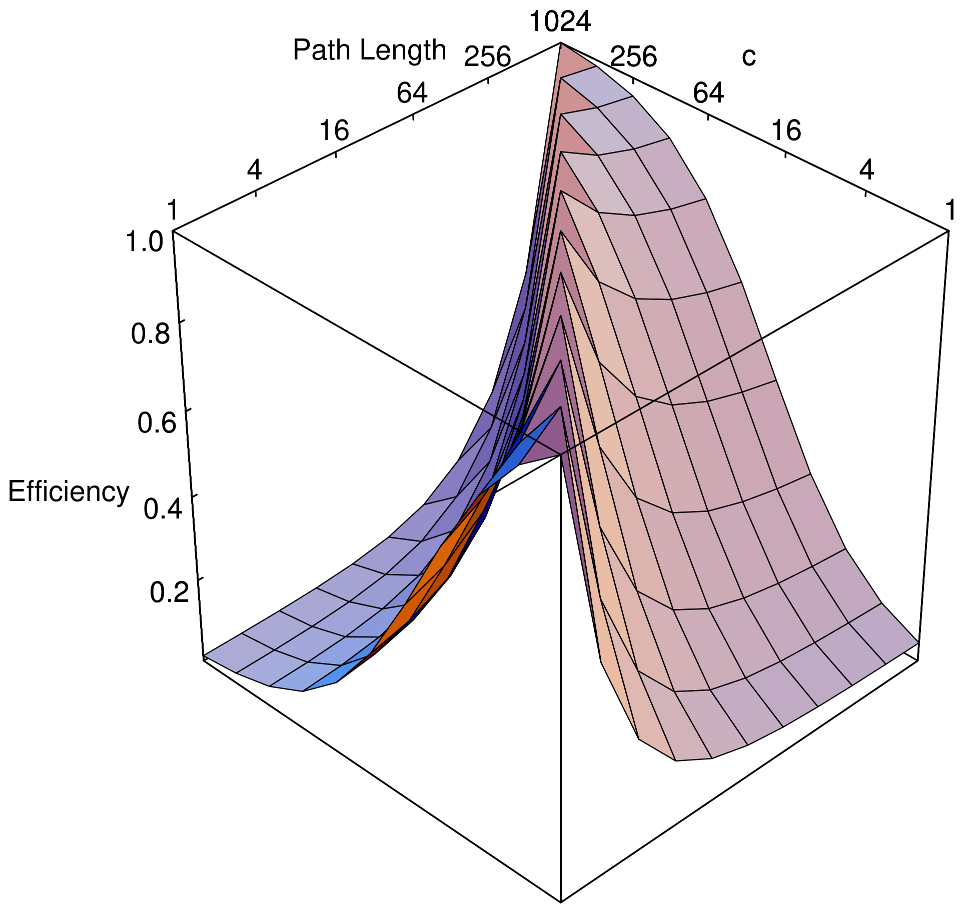
If $w_{task} > w_{arch}$:

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

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

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

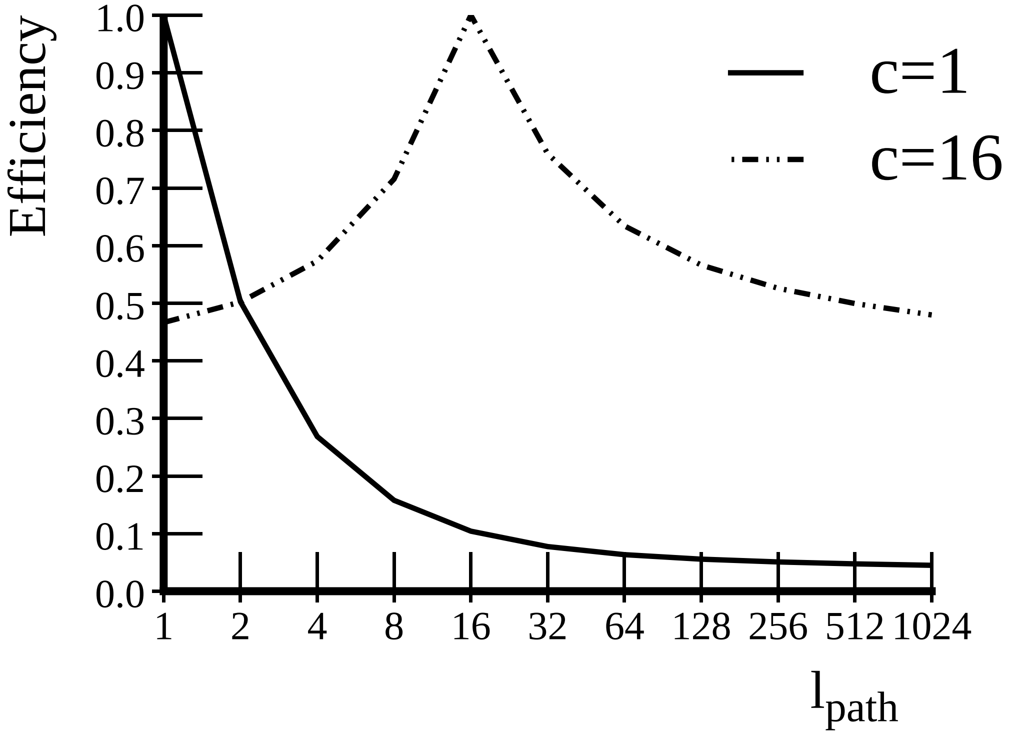
Efficiency with fixed Width ($w = 1$)



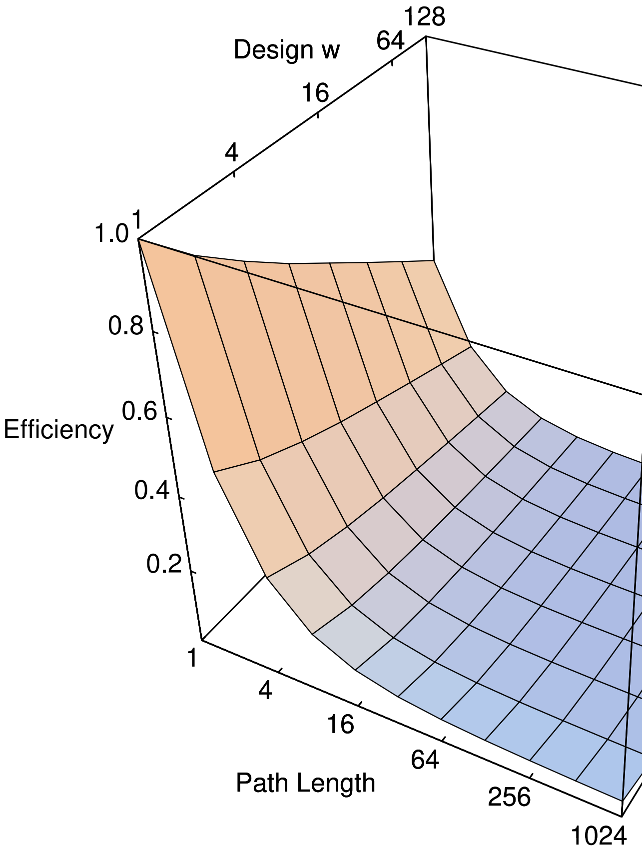
(at 16K Bit Processing Elements)

BRASS

Pedagogical FPGA v/s DPGA Comparison

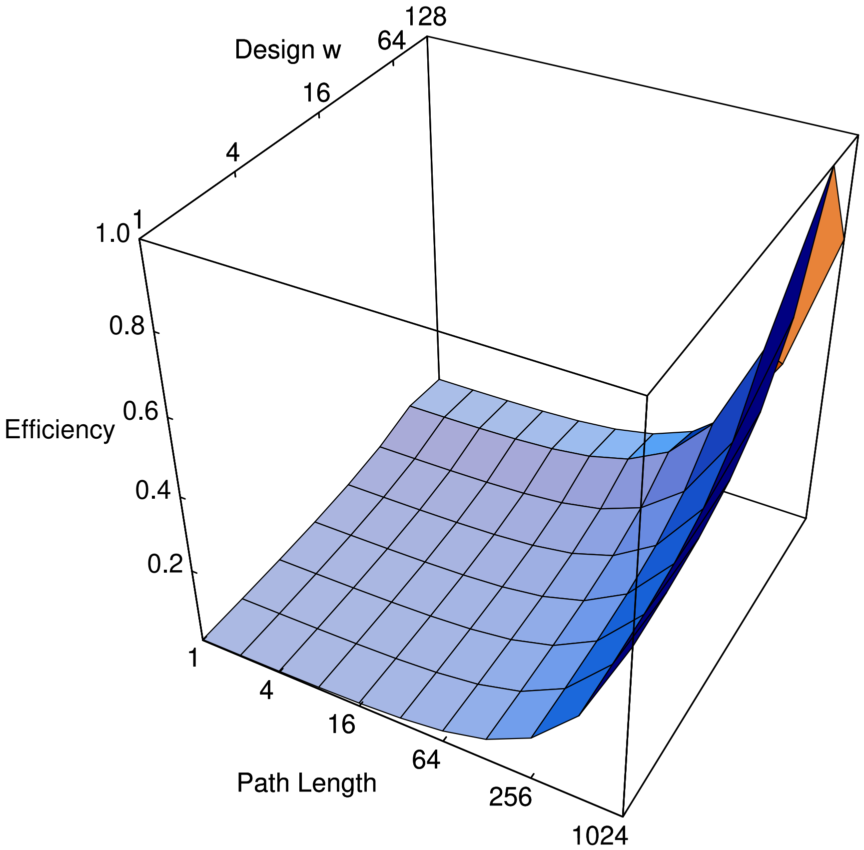


Comparing "Processors" and FPGAs



FPGA

$c = d = 1,\ w = 1$



"Processor"

$c = d = 1024,\ w = 64$

(at 16K Bit Processing Elements)

Caveats

Abstracting away many details

Notably:

- interconnect
- control
- specialized functional units

Applications are heterogeneous mix of characteristics.

Architectural space is large.

Second Break

Reconfigurable Computing

Instruction distribution and storage take up space.

If we can afford to do without them,
we can exploit higher computational density.

Shallow instructions $\rightarrow$ reconfigurable computing

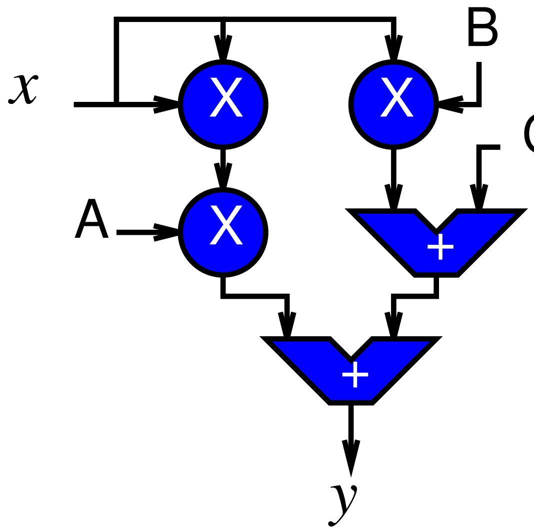
Configurable Computing

Computing via spatially programmed connection of processing elements.

Resembles "hardware" computation

"wire" up functional computing blocks

$$y = Ax^2 + Bx + C$$



Can we exploit it? – Application

- The fastest RSA implementation is on a reconfigurable machine (DEC PAM).
  `<http://www.research.digital.com/PRL/publications/pam.html>`
- Splash (SRC) performs DNA Sequence matching at 300× Cray2 speed, and 200× 16K CM2.
  `<http://www.super.org:8000/FPGA/fpga.html>`
  Computer, January 1991
- Modern processor designs are emulated on FPGA-based accelerators.
- Vendors showing high performance on FIR Filters

Finite Impulse Response Computation

$$y_i = w_1 \cdot x_i + w_2 \cdot x_{i+1} + \cdots + w_k \cdot x_{i+k-1}$$

Examples based on:

- 8-bit samples
- 16-bit accumulate

Processor FIR

Psuedo-C description:

```
for(j=0;j<SAMPLES;j++)
    {
      out[j]=0;
      for (i=0;i<FILTER_TAPS;i++)
        out[j]+=coeff[i]*sample[j+i];
    }
```

Processor FIR

For Each TAP:

```
MULT COEFF[RA],SAMPLE[RB],RC
ADD RC,RD
ADD RA,1,RA
ADD RB,1,RB
```

- If small enough, can unroll inner loop and keep coefficients in registers
- Some processors have multiply-add
- 1-4 instructions / TAP

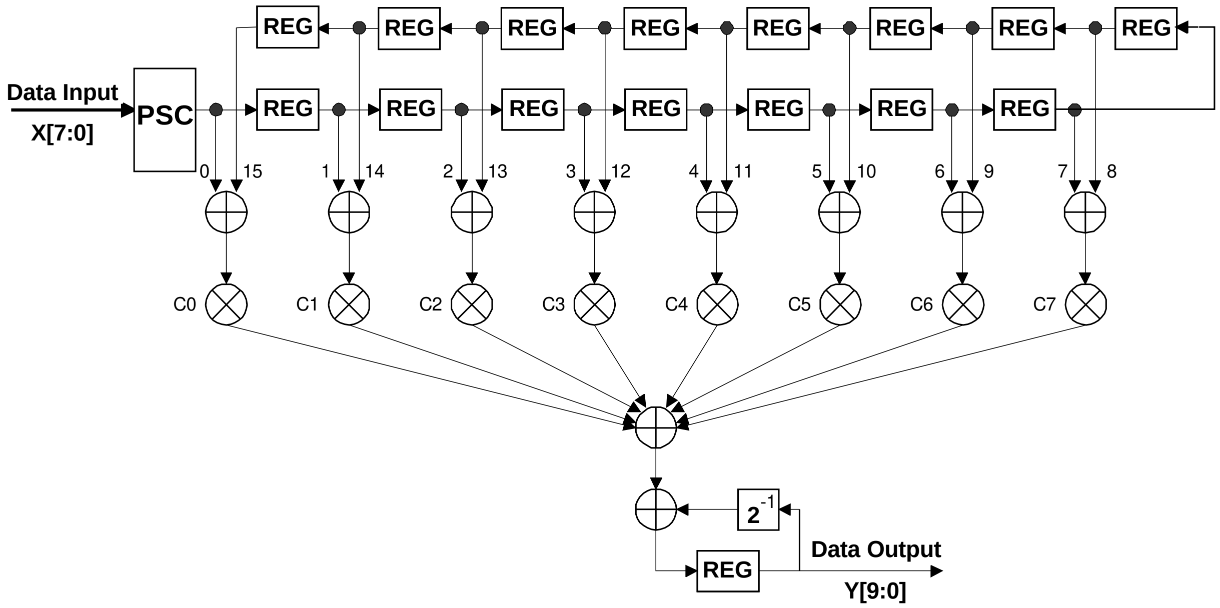
DSP FIR

Single MAC instruction:

- Multiply + accumulating add
- Increment pointers
- Test loop limits

$\Rightarrow$ 1 instruction / TAP

FPGA FIR



Data Flow Diagram for 16-Tap FIR Filter -- Source Xilinx

- Coefficients constant (simplifies multiply)
- Bit serial for low-sample rates
- All operations in parallel – 67 CLBs (134 AEs)

<http://www.xilinx.com/apps/dsp.htm>

FIR Roundup

Architecture	Reference	area and time	$\frac{\text{Filter TAPs}}{\lambda^2 s}$
16b DSP	Philips '86	125 ns/TAP	0.090
	AT&T '87	60 ns/TAP	0.082
	Toshiba '92	50 ns/TAP	0.072
32b RISC MSTEP	MIPS-X	10+ cycles/TAP	0.029
32b RISC/DSP	VSP8	40 ns/TAP	0.022
64b RISC	Alpha '96	2.3 ns/TAP	0.064
XC4K	Xilinx '94	67 CLBs, 184 ns/16-TAPs†	1.0
PADDI2	UCB '95	5 EXUs, 20 ns/TAP	0.93
Gate Array fixed coefficient	'90	10 ns/64 TAPs	21
Full Custom	LSI '89	45 ns/64 TAPs‡	3.6
	'92	50 ns/10 TAPs	2.4
fixed coefficient	'92	6.7 ns/43 TAPs§	57

† – symmetric filter only

‡ – 24-bit accumulate

§ – 16×16 architecture

Where is RC interesting?

Regular applications – need to perform the same basic operations repeatedly
(short path length)

High concurrency – large number of operations which can occur simultaneously

(for Traditional FPGAs)
Fine-grained data – small operand data widths

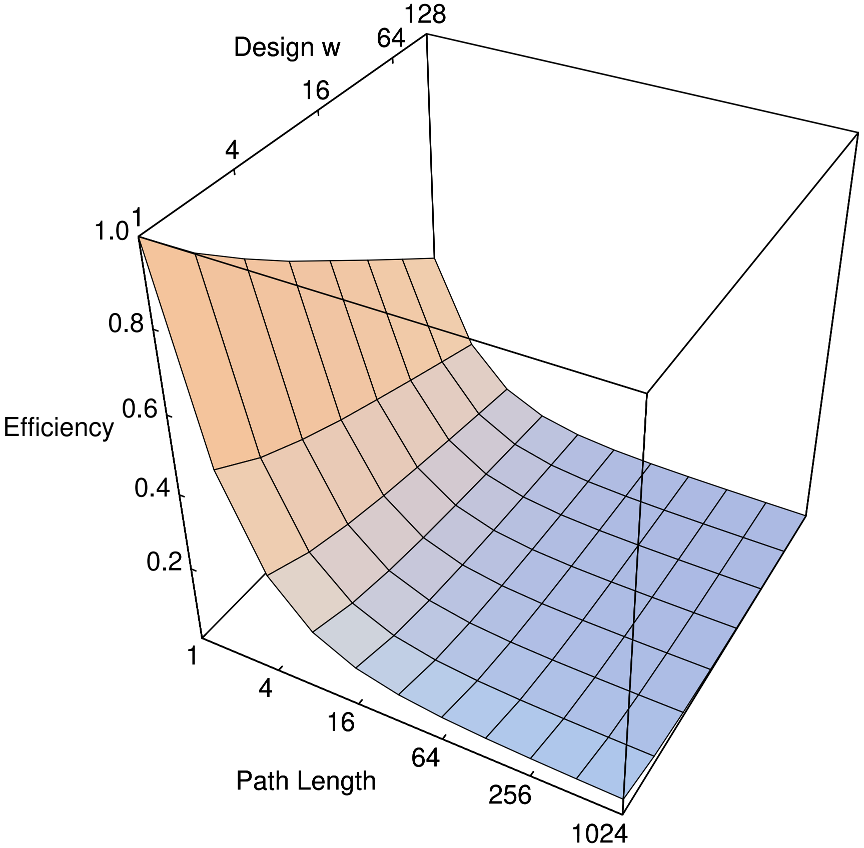
Applications have mixed requirements

90/10 Locality Rule: **A program executes about 90% of its instructions in 10% of its code.**

Suggests typical programs have:

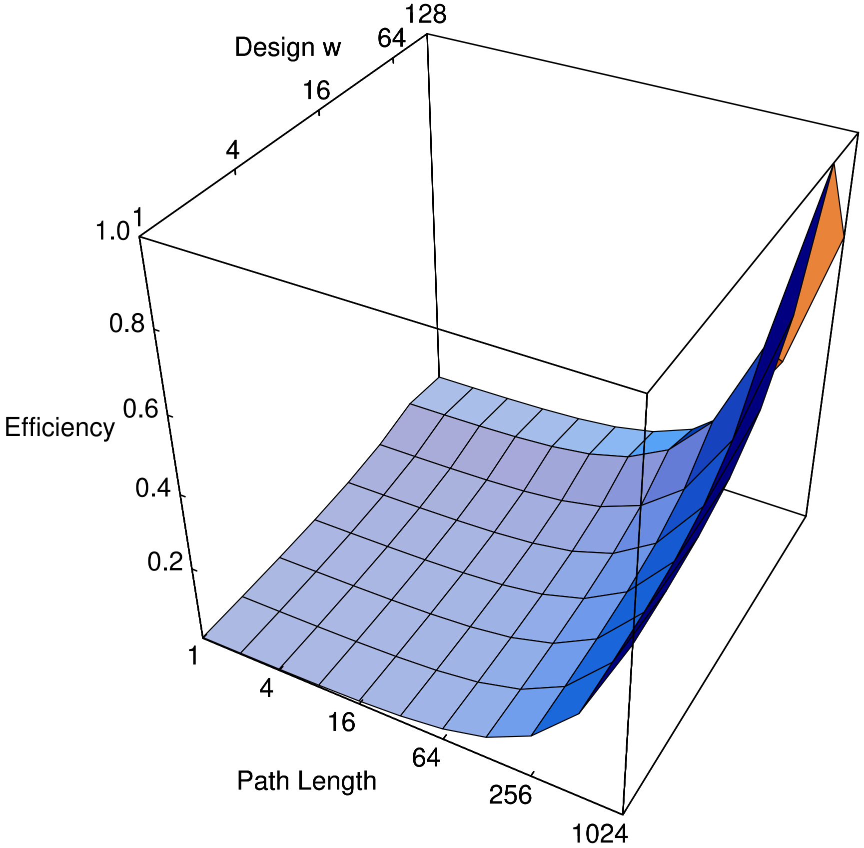
- **a “regular” portion, suitable for reconfigurable array**
- **an “irregular” portion, suitable for processor**

Reviewing "Processors" and FPGA Efficiency



FPGA

$c = d = 1, \; w = 1$

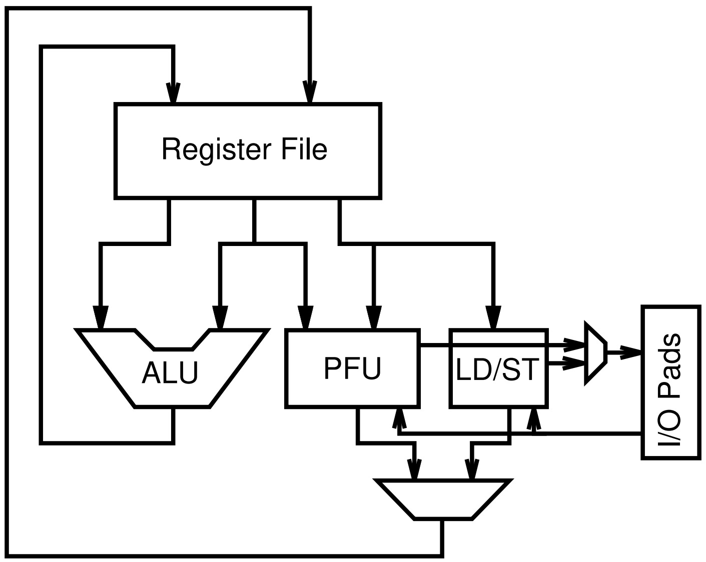


"Processor"

$c = d = 1024, \; w = 64$

(at 16K Bit Processing Elements)

BRASS



Tightly integrate:

- a conventional processor
- a reconfigurable array

High bandwidth, low-latency path between components

Array looks like functional unit or co-processor (*e.g.* FPU)

Array may aid system integration

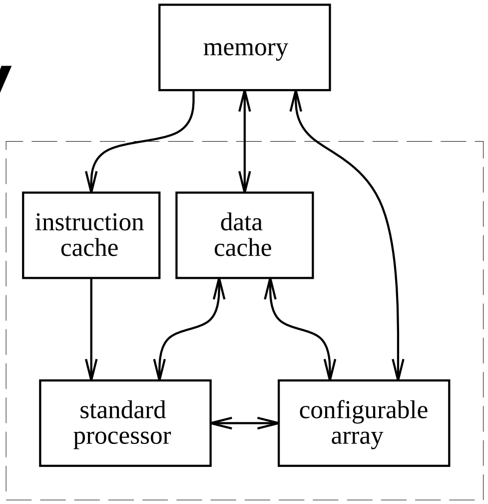
Previous Work

- PRISM - Brown - discrete: 68K + XC4K Computer, March 1993
- PRISC - Harvard - integrated, tiny array
- Cypris Crypto Processor - Lookheed Martin - programmable PLA
- OneChip - U of Toronto
- RISC+FPGA case study - CS252, Fall 1995

<http://www.cs.berkeley.edu/projects/brass/reproc.html>

GARP – a first pass implementation

- MIPS-II core (from T0)
- Co-processor interface to gate-array
- $w = 2$
- Datapath oriented logic rows
- 24×32×2 ≈ 1500 bit processors
- Local configuration cache ($\approx c = 4$)
- Automatic reload of array instructions
- Development by: John Hauser `<jhauser@cs>`



`<http://www.cs.berkeley.edu/projects/brass/garp.html>`

Research

- Expand understanding of where to exploit reconfigurable architectures
- Tools
  - Automate (parts of) the discovery and selection process
  - Transformation/reformulation for array
  - Array mapping
- Better Array Architectures
- Strategies for managing reconfigurable resources

Review

Review

- Array Processing
- Issue: Instruction resources
- Exploit application structure
  to reduce instruction requirements
- Classify architectures by instruction organization
- Implications of architecture/application mismatches
- Area model $\rightarrow$ efficiency
- Reconfigurable architectures
  - high computational density
  - for regular tasks
  - How many tasks fit into area?
- BRASS: Exploring hybrid – best of both worlds?

More?

Reconfigurable Architectures
for
General-Purpose Computing

`<http://www.ai.mit.edu/people/andre/phd.html>`

BRASS

`<http://www.cs.berkeley.edu/projects/brass/>`

Reinventing Computing

`<http://www.ai.mit.edu/projects/transit/`
`rc_home_page.html>`