USC

Reconfigurable Computing: Opportunities and Challenges

Viktor K. Prasanna

XXX Department of Electrical Engineering

University of Southern California

USC Viterbi School of Engineering

Outline

- Reconfigurable Computing
- Theoretical Foundations
- Network Intrusion Detection
- High Performance Embedded Signal Processing
- FPGA-based Designs for BLAS
- Conclusion

Acknowledgements

- Joint work with my students:

  Ling Zhuo, Zack Baker(LANL), Jingzhao Ou(Xilinx),

  Ron Scrofano

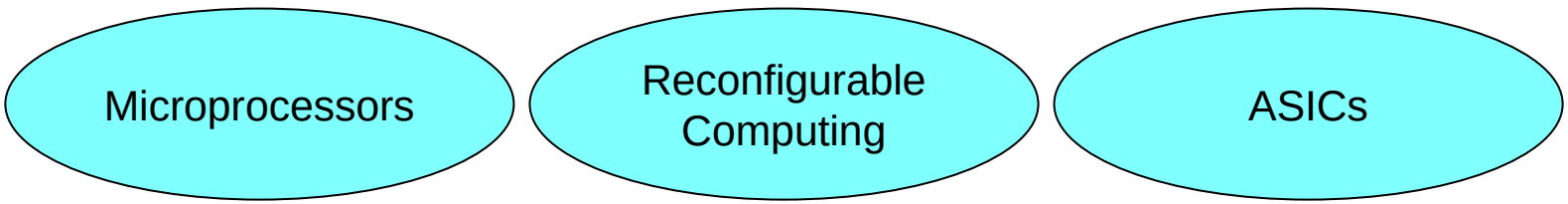
  Gerald R. Morris(US Army ERDC)

- US National Science Foundation (NSF)
- US Defense Advanced Research Projects Agency (DARPA)
- Los Alamos National Lab (LANL)

Ceng.usc.edu/~prasanna

Reconfigurable Computing

- Goal: software flexibility with hardware performance
  - Microprocessors can execute any software, but performance can be slow
  - ASICs can execute only one application, but quickly
  - *Reconfigurable computing seeks to bridge this gap*
    - Reconfiguration allows same hardware to execute multiple applications
    - Executing application in hardware leads to higher performance than in software



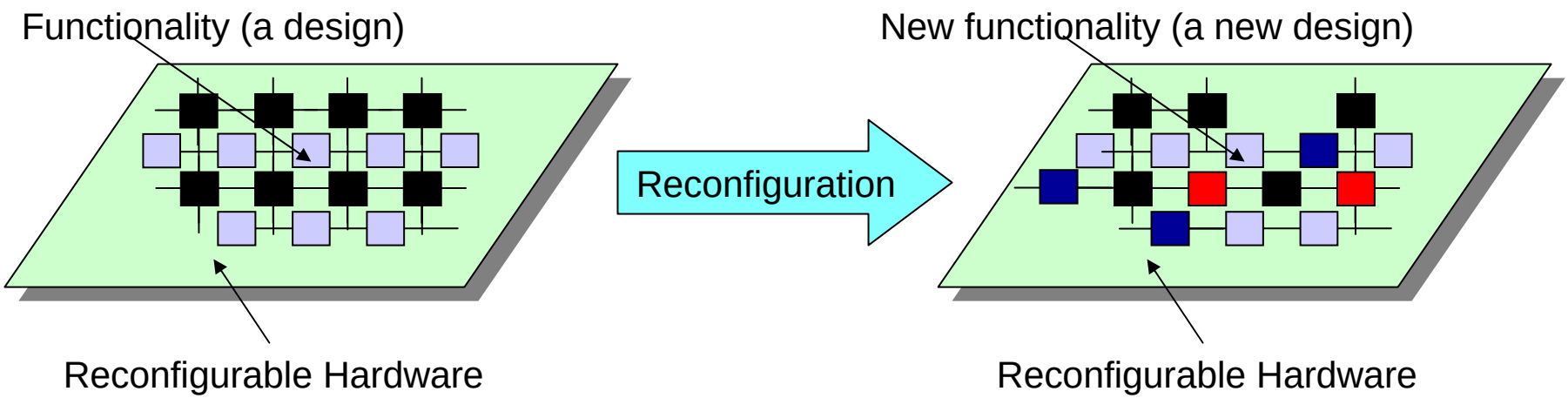
Highest Flexibility
Performance?

High Flexibility
High Performance

Highest Performance
Lowest Flexibility

Reconfigurable Computing (2)

- Configuration
  - "Programming" the hardware
  - Implements desired functionality in hardware
    - For example, if addition is necessary, adder made in hardware
- Reconfiguration
  - Changing the configuration of the device
  - Typical overhead: ~ microseconds to ms
  - Two types: static and dynamic

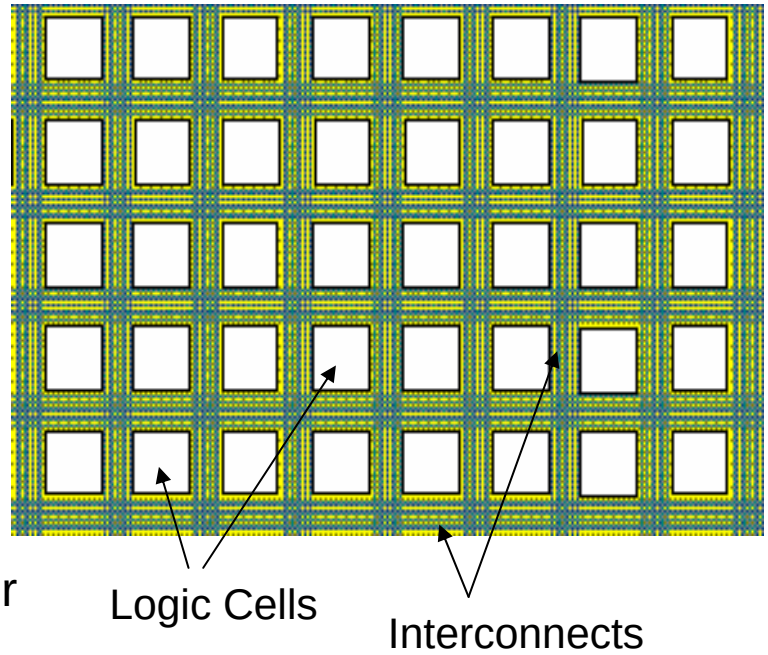


Types of Reconfiguration

- Static reconfiguration
  - Configure the device once for the application
  - Do not reconfigure until the application is finished or do not reconfigure at all
    - Hardware still flexible in the design phase
  - Application: high performance computing that needs hardware performance without the high cost of designing an ASIC
- Dynamic/run-time reconfiguration
  - Hardware is reconfigured while the application is executing.
  - Partial reconfiguration
    - Part of the reconfigurable hardware is reconfigured while the rest stays the same and continues to execute
  - Duration between reconfigurations varies
    - Long duration: e.g. router updating tables and/or protocols, cell phone switching protocols
    - Short duration: e.g. regular expression matching, DSP application switching between stages

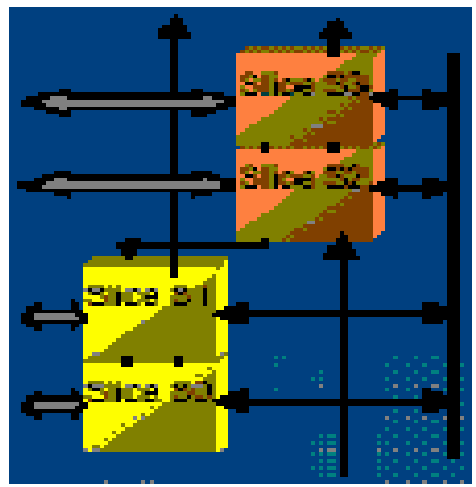
FPGA Basics

- FPGA consists of
  - Matrix of programmable logic cells
    - Implement any logic function
      - AND, OR, NOT, etc
    - Groups of cells make up higher level structures
      - Adders, multipliers, state logic, etc.
  - Programmable interconnects
    - Connect the logic cells to one another
  - Embedded features
    - ASICs within the FPGA fabric for specific functions
      - Embedded multipliers, on-chip memory, microprocessors
- FPGAs are SRAM-based
  - Configure device by writing to configuration memory



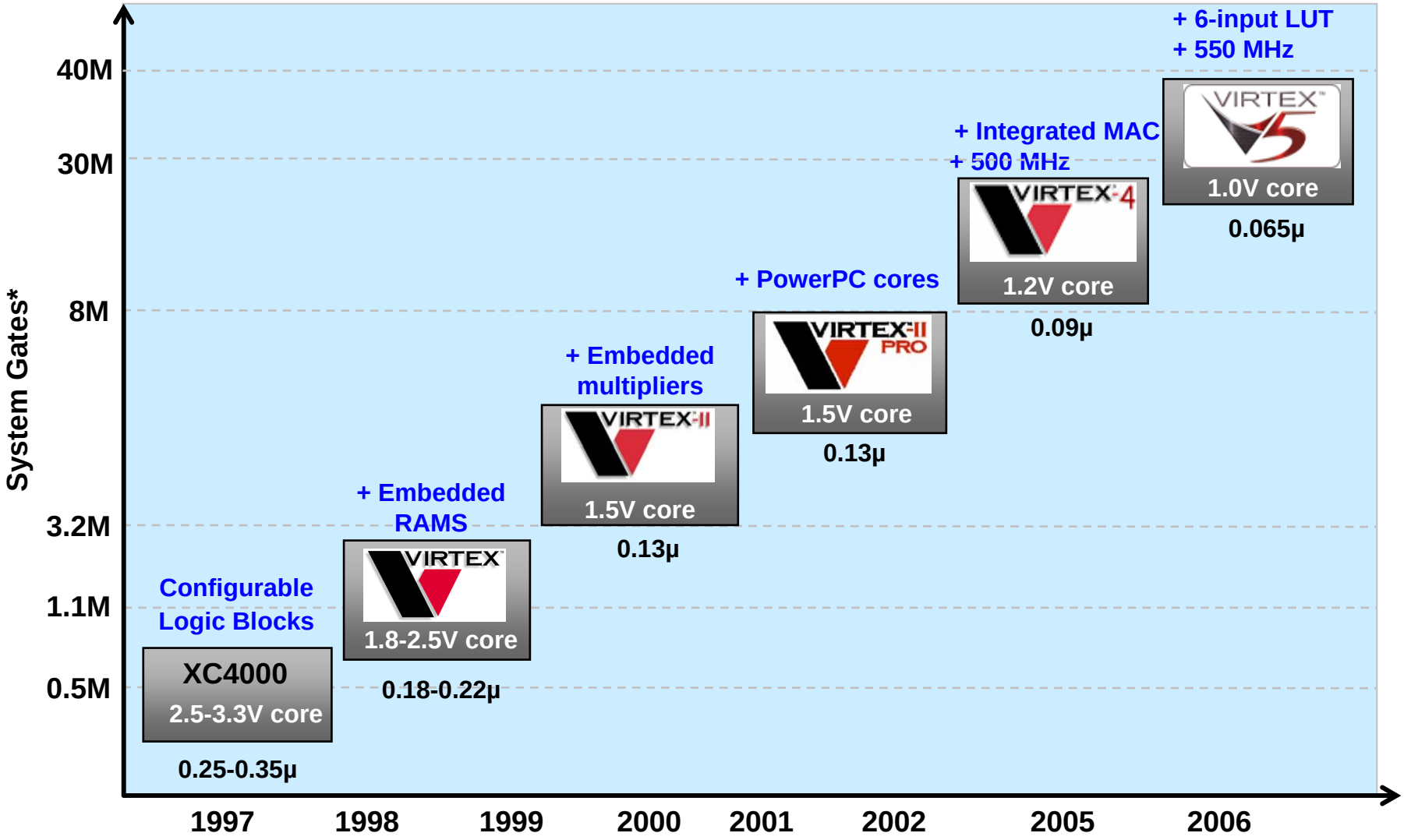
Xilinx Virtex-II Pro

- Xilinx terminology
  - Four-input Look-Up Table (LUT): programmable block implementing four-input logic functions
  - *Slice*: 2 LUTs, flip-flops, multiplexers, arithmetic logic, carry logic, and dedicated internal routing
  - *Routing*: the interconnection between logic resources giving them the desired functionality
- Features
  - Up to 44,096 slices in one chip
  - Up to 7792 Kbits of embedded RAM
  - Up to 444 embedded multipliers
  - Up to 1200 User I/O pins



Virtex-II Pro CLB

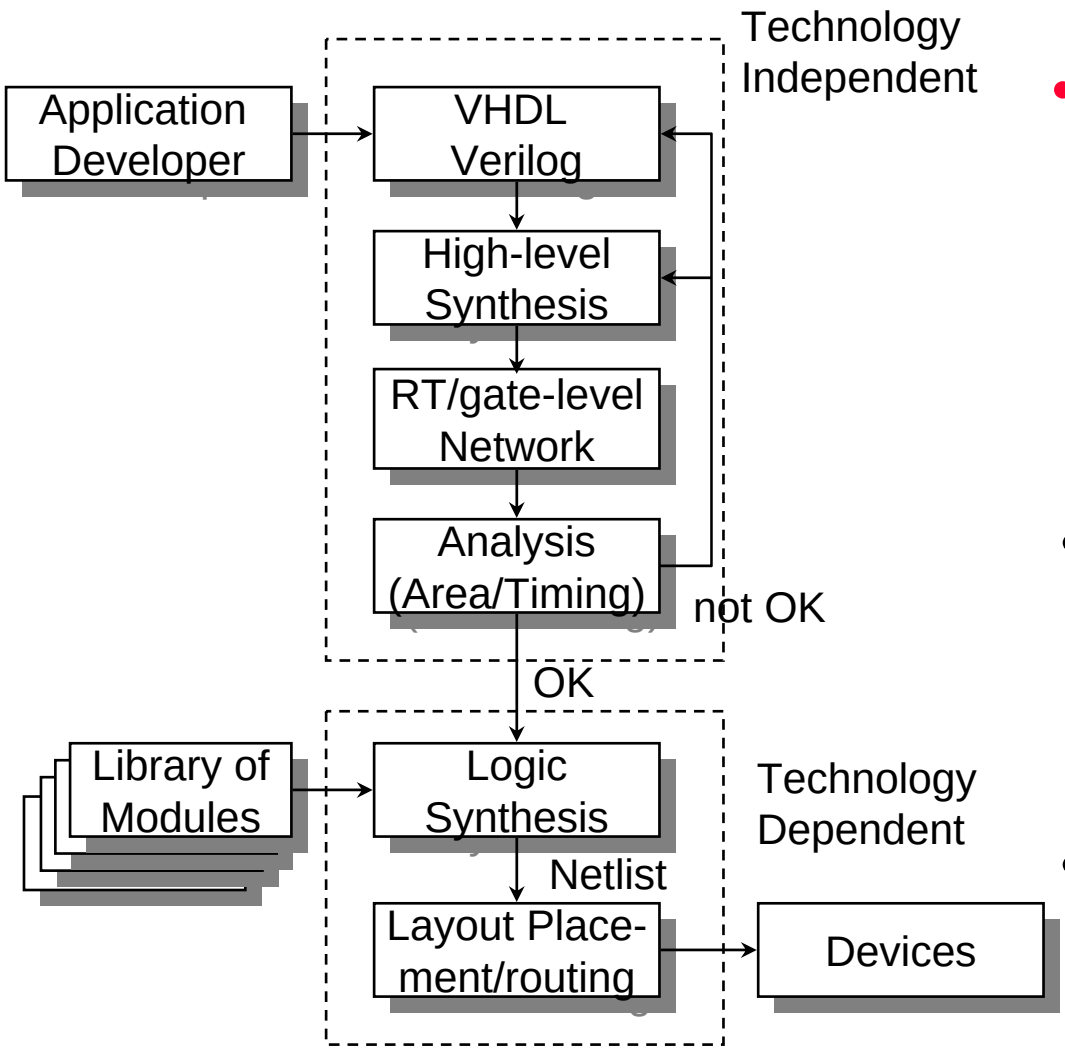
Device Trends



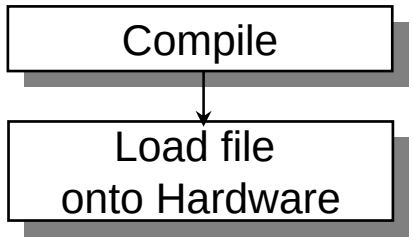
Outline

- Reconfigurable Computing
- Theoretical Foundations
- Network Intrusion Detection
- High Performance Embedded Signal Processing
- FPGA-based Designs for BLAS
- Conclusion

Traditional Approach to Design of Configurable Computing Solutions

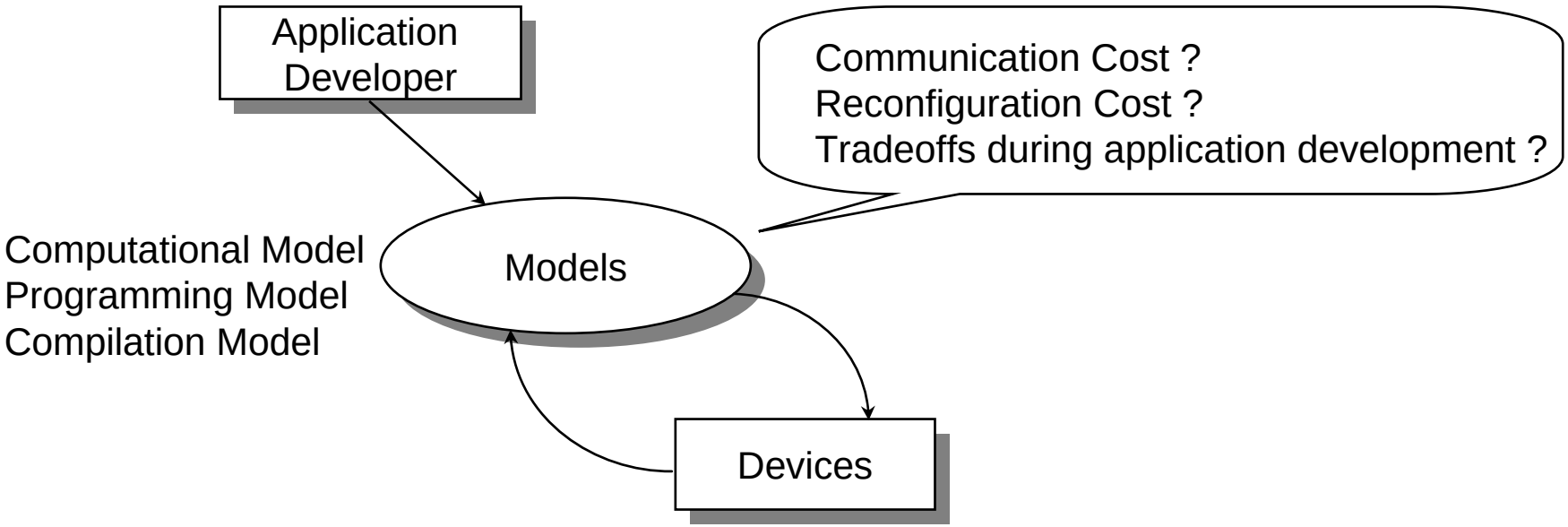


- Static configuration

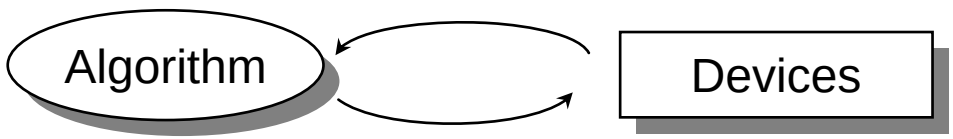


- Application developer does not “see” the architecture/device features
- Logic synthesis Vs. Algorithm synthesis

Alternate Approach

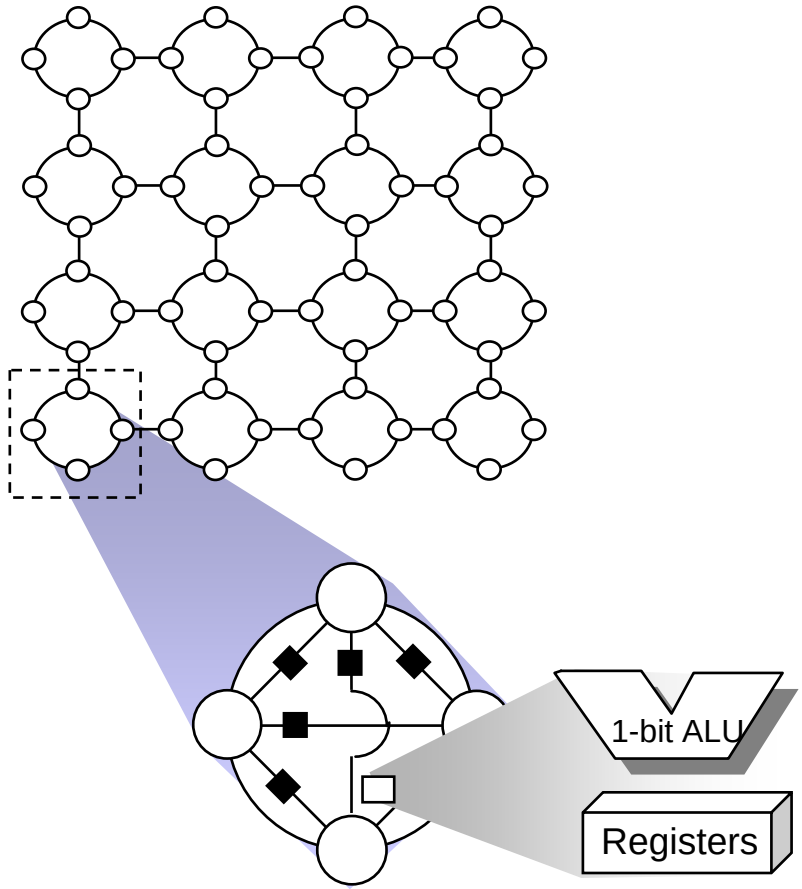


- Application developer "sees" an abstraction of the device features in the algorithm design phase



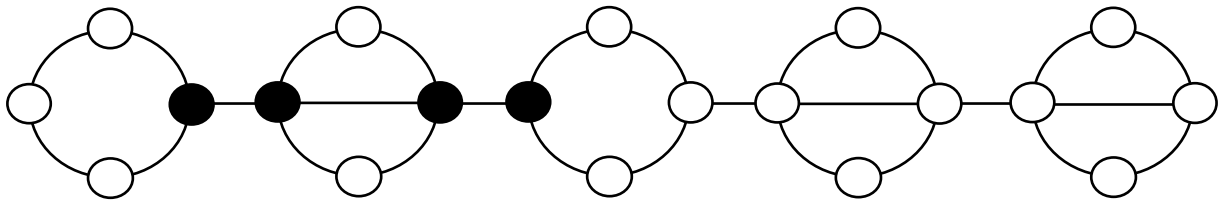
Reconfigurable Mesh Model (RMesh)

- A model for understanding dynamic configuration
  - NxN mesh of processing elements
  - Processing Element
    - Configurable logic
    - Configurable switches
- Synchronous Model
- Communication Cost
  - Constant delay
  - Log delay
- Abstract Model
- MIT Conference in Advanced Research in VLSI, 1989



Power of Dynamic Reconfiguration

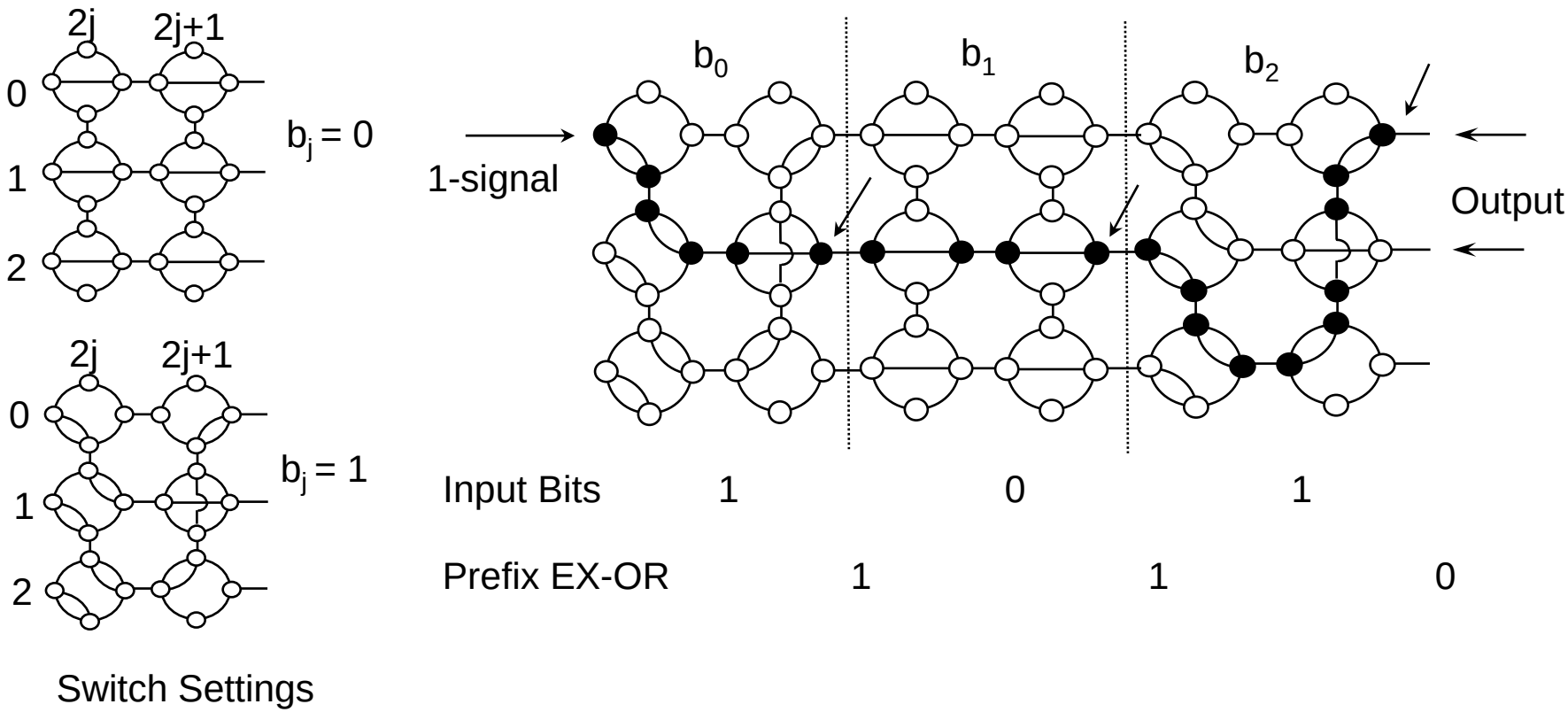
- Addition of two *n*-bit numbers for *n* = 5



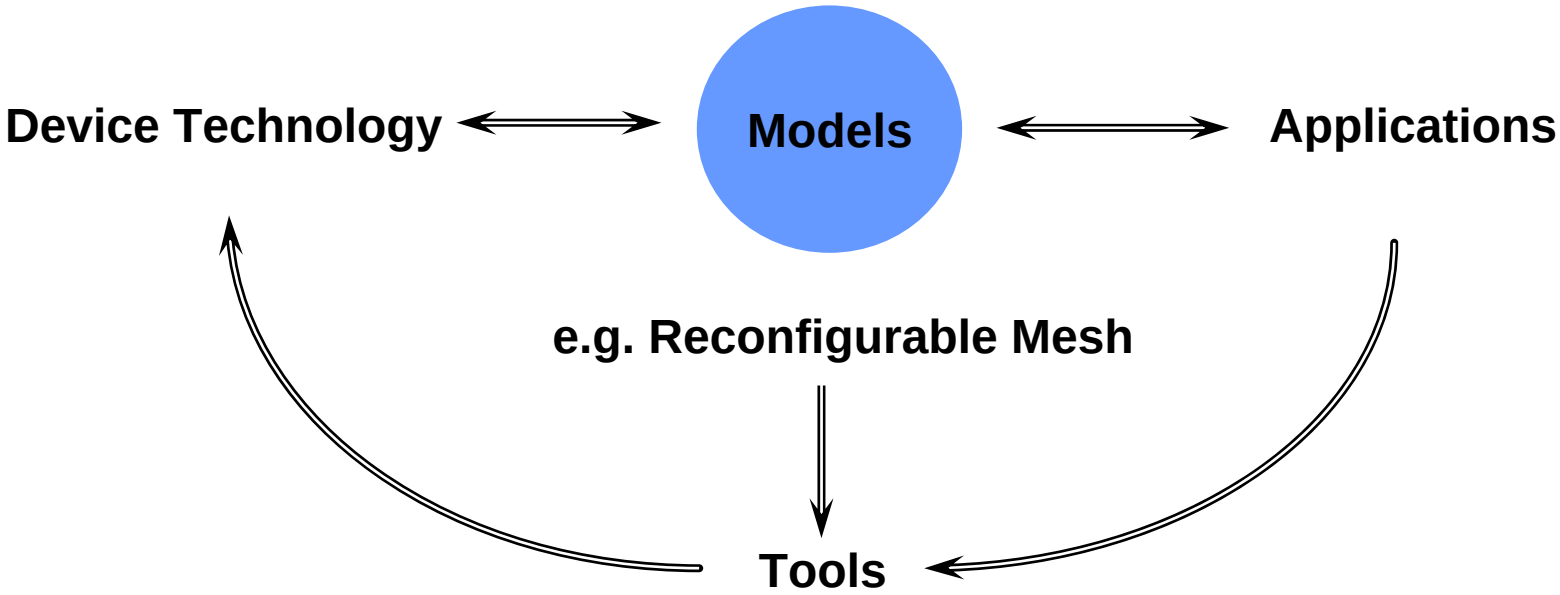
		i = 4	3	2	1	0
Input	a_i =	0	1	1	0	0
	b_i =	0	0	1	1	1
Carry	c_i =	1	1	0	0	0
Sum Output	z_i =	1	0	0	1	1

Power of Dynamic Reconfiguration

- EX-OR computation
  - Modulo Arithmetic - Coding theory applications
  - FAST ⇨ Exponential number of gates !



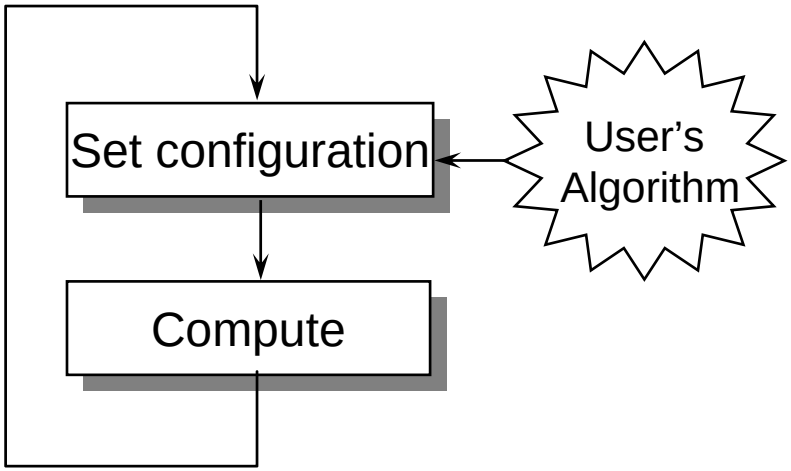
Challenge (1): Models



- Algorithmic configurable computing
- Hidden potential of dynamic reconfiguration

Widely Accepted Models?

Challenge (2): Dynamic Reconfiguration



- Data dependent reconfiguration
- Frequent reconfiguration
- Distributed control
- Cost of configuration
  - Number of bits
  - Reconfiguration time

✓ • Configure logic

? • Configure connections

} at Run time

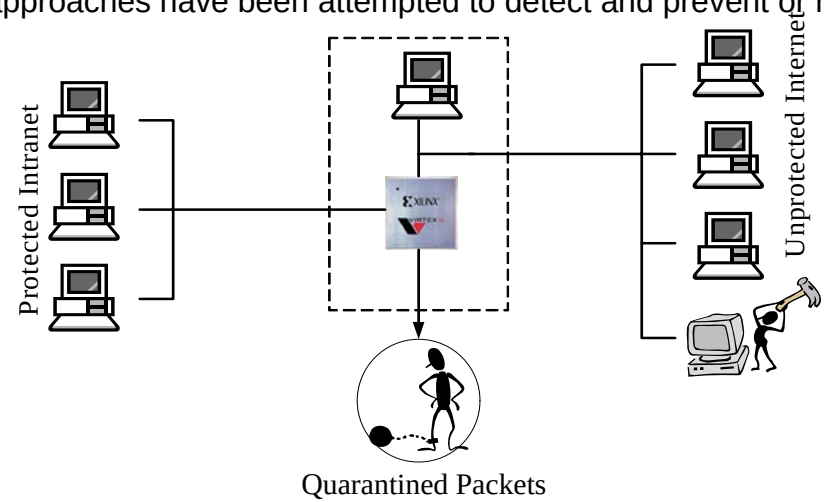
Non Trivial Applications of Dynamic Reconfiguration

Outline

- Reconfigurable Computing
- Theoretical Foundations
- Network Intrusion Detection
- High Performance Embedded Signal Processing
- FPGA-based Designs for BLAS
- Conclusion

Intrusion Detection System (IDS)

- Definition
  - Search the packet contents for various patterns that imply an attack
- Objective
  - Protect an intranet from attack
    - Real-time filtering is thus a key performance metric
    - For large subnets, high data rates must be handled without buffer overruns
- Methodology
  - Detect and mitigate attacks
    - Assume security is provided through a machine that sits on network and can see all traffic
      - Many approaches have been attempted to detect and prevent or record attacks autonomously



String Matching for IDS

- IDS
  - All incoming packets are filtered for specific characteristics or content
    - Snort/Hogwash database
    - Databases have thousands of patterns requiring string matching
  - 10 Gb/s and higher rates desired
    - Computationally intensive!

- String matching
  - String literals often are direct commands
  - Correlated strings can find illicit behavior (buffer overflows)
    - Regular expressions can find more complicated patterns
  - Algorithms
    - Aho-Corasick, Knuth-Morris-Pratt (KMP), Boyer-Moore, etc

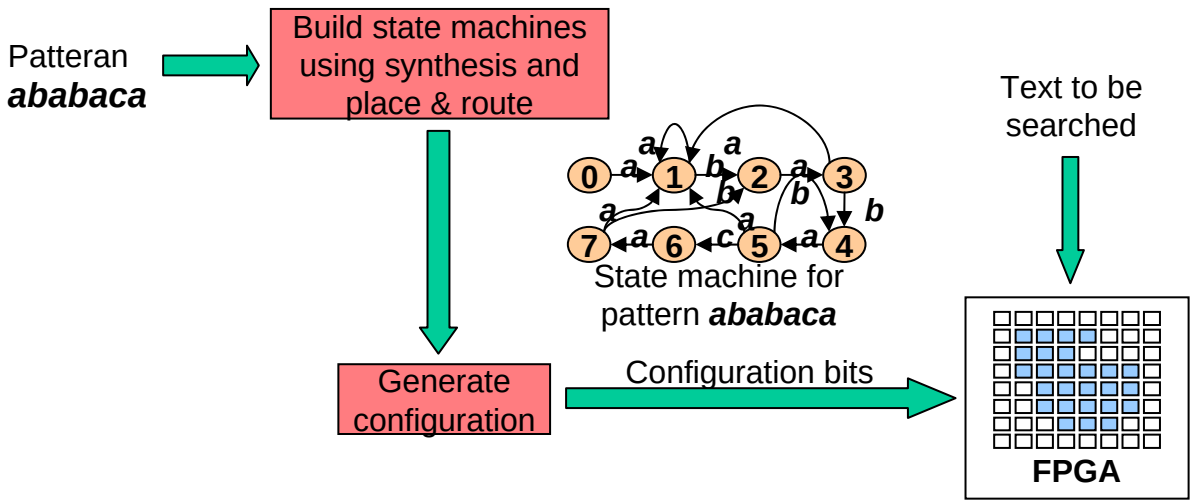
Processors?

Dual 1GHz Pentium III system, using 845 patterns, runs at **50 Mbps**

- FPGA allows fine-grained parallelism and computational reuse
  - Potential for thousands of state machines running in parallel
  - Thousands of parallel comparisons allow for performance advantage over even the best algorithms running on traditional microprocessors
    - A single FPGA (our 4-way design, exact match, 1000 patterns), runs up to **5 Gbps**.

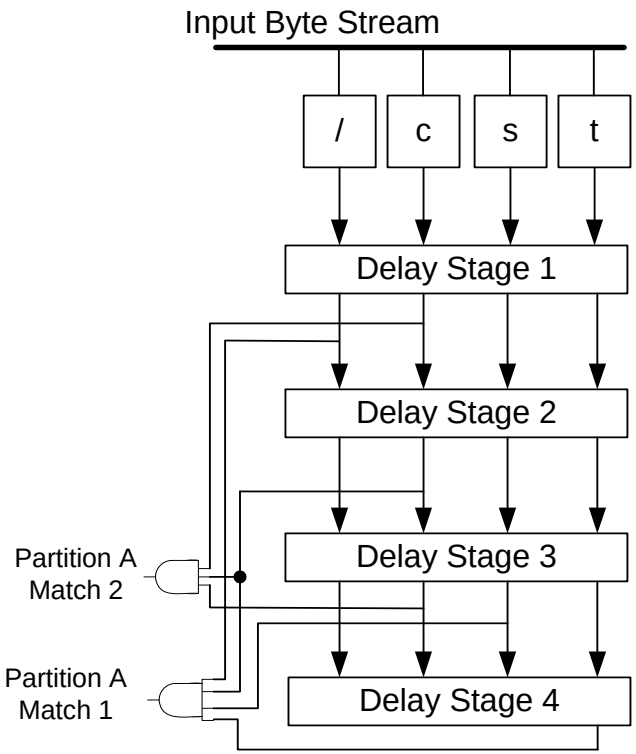
Hardwired Approach for IDS

- Builds complete state machines with HDL code
  - Potential for very high area and time performance
  - State machine: A model composed of states and transitions
    - A state shows current position (information) of the machine
    - A transition indicates a change from one state to another by certain conditions
  - States and transitions are built after synthesis and place & route
    - Very long time is required to change patterns
  - Difficult to update the pattern matching rules on-the-fly!



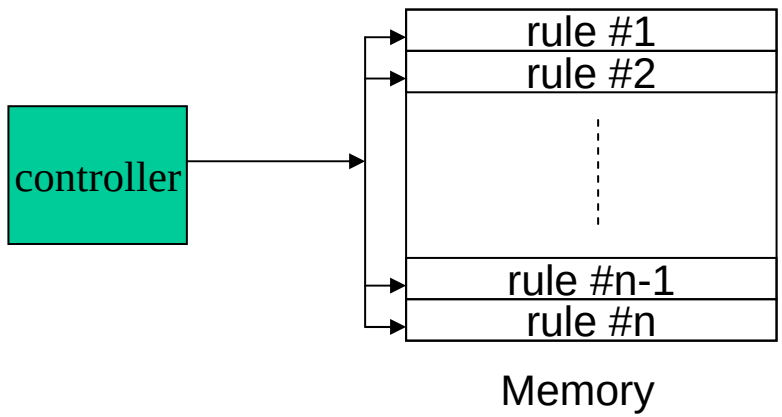
Our Techniques (1)

- Hardwired designs – simplified state machine
  - System-level Design with Pre-decoding
    - Uses pre-compilation phase and reconfiguration of FPGA to compactly represent patterns
  - Database composed of several thousand pattern matching rules
    - Unit view of matching units can be inefficient
      - Unnecessary work and hardware
    - Process entire database to remove redundancy
  - Redundancy extraction techniques are novel to USC



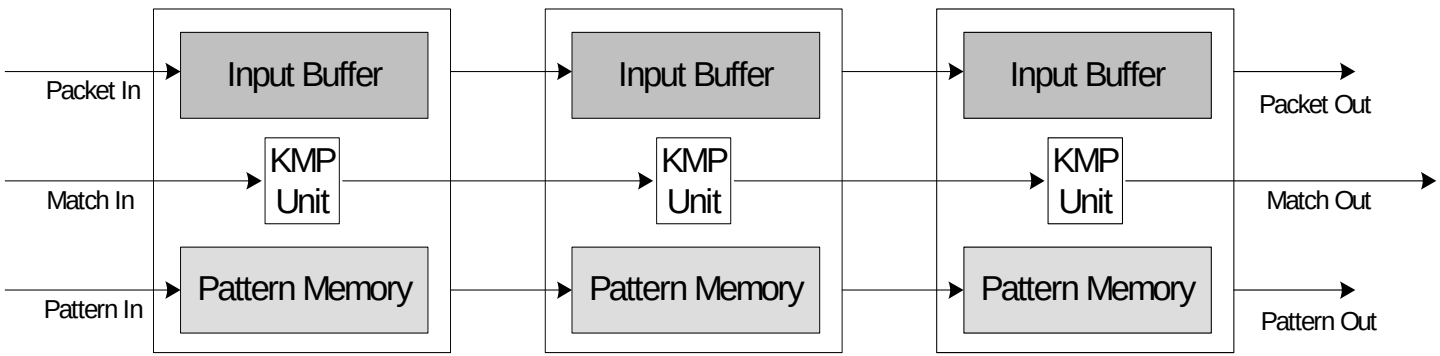
Memory-based Approach for IDS

- Easy and fast update of the rule database
  - Programming for the controller and memory structures are generated as needed
  - State machines are generated externally by our software tools (C/C++)
    - Advantage: No synthesis and place & route time is required for the generation of state machines
    - These generated state machines are loaded into FPGA Memories (Block RAM etc.)
  - Our work is based on an FPGA implementations of Aho-Corasick and Knuth-Morris-Pratt techniques



Our Techniques (2)

- Knuth-Morris-Pratt (KMP) Architecture
  - Advantages
    - With our modification, allows configuration in linear array, a powerful strategy to minimize wire length and fanout
    - Increase in pattern size causes a proportional increase in area only in look up tables and buffer
    - Memory-based (ASIC or FPGA implementation)
      - Allows for on-the-fly reconfiguration



Some Results

• Define a performance metric, and we have an architecture optimized for that metric

Design	Throughput	16 Char Unit Size	Performance/Unit Area
USC KMP (unpipelined)	1.8 Gb/s	92	19.6
USC KMP (pipelined)	2.4 Gb/s/2	120/2	20
USC Unary	2.07 Gb/s	7.3	283
USC Unary (4 byte)	6.1 Gb/s	22.3	271
USC Unary (8 byte)	10.3 Gb/s	32	322
USC Unary (Prefilter)	6.4 Gb/s	9.4	682
USC Unary (Tree)	2.00 Gb/s	6.6	303
Los Alamos	2.2 Gb/s	243	9.1
UCLA	2.88 Gb/s	160	18
U/Crete	10.8 Gb/s	269	40.1

• Throughput, unit size per pattern (in logic cells; one slice is two logic cells), and performance (in Mb/s/cell). Throughput is assumed to be constant over variations in pattern size. Our experiments are on V2Pro 100 –6 using ISE 5.2 with Synplify 7.2

Some Results

High area efficiency,
On-the-fly reconfiguration

• Define a performance metric, and we have an architecture optimized for that metric

Design	Throughput	16 Char Unit Size	Performance/Unit Area
USC KMP (unpipelined)	1.8 Gb/s	92	19.6
USC KMP (pipelined)	2.4 Gb/s/2	120/2	20
USC Unary	2.07 Gb/s	7.3	283
USC Unary (4 byte)	6.1 Gb/s	22.3	271
USC Unary (8 byte)	10.3 Gb/s	32	322
USC Unary (Prefilter)	6.4 Gb/s	9.4	682
USC Unary (Tree)	2.00 Gb/s	6.6	303
Los Alamos	2.2 Gb/s	243	9.1
UCLA	2.88 Gb/s	160	18
U/Crete	10.8 Gb/s	269	40.1

• Throughput, unit size per pattern (in logic cells; one slice is two logic cells), and performance (in Mb/s/cell). Throughput is assumed to be constant over variations in pattern size. Our experiments are on V2Pro 100 –6 using ISE 5.2 with Synplify 7.2

Some Results

Pre-design prefix grouping allows higher area efficiency

• Define a performance metric, and we have an architecture optimized for that metric

Design	Throughput	16 Char Unit Size	Performance/Unit Area
USC KMP (unpipelined)	1.8 Gb/s	92	19.6
USC KMP (pipelined)	2.4 Gb/s/2	120/2	20
USC Unary	2.07 Gb/s	7.3	283
USC Unary (4 byte)	6.1 Gb/s	22.3	271
USC Unary (8 byte)	10.3 Gb/s	32	322
USC Unary (Prefilter)	6.4 Gb/s	9.4	682
USC Unary (Tree)	2.00 Gb/s	6.6	303
Los Alamos	2.2 Gb/s	243	9.1
UCLA	2.88 Gb/s	160	18
U/Crete	10.8 Gb/s	269	40.1

• Throughput, unit size per pattern (in logic cells; one slice is two logic cells), and performance (in Mb/s/cell). Throughput is assumed to be constant over variations in pattern size. Our experiments are on V2Pro 100 –6 using ISE 5.2 with Synplify 7.2

Challenge (3): Beyond String Matching

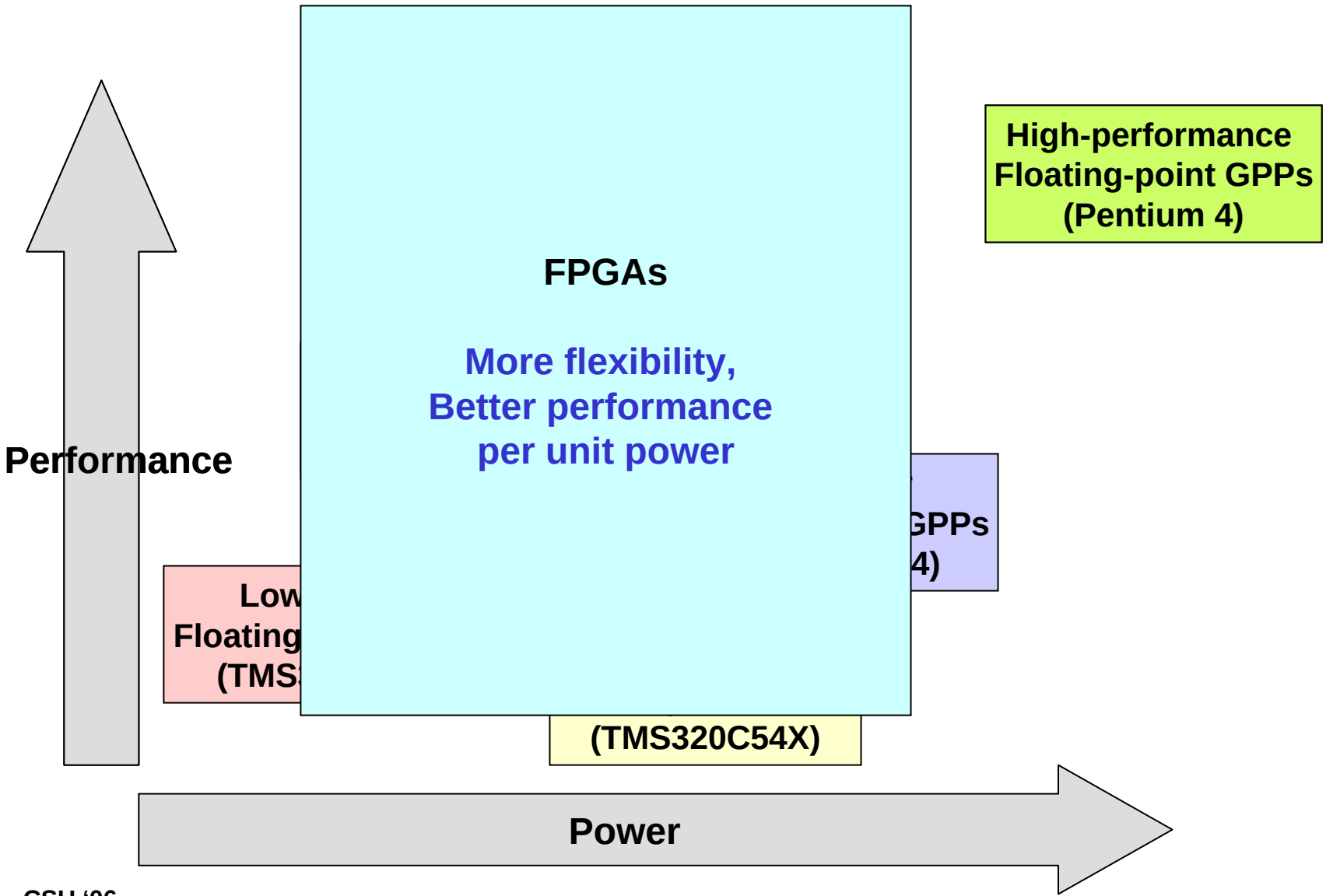
Regular Expressions

Correlated Attacks

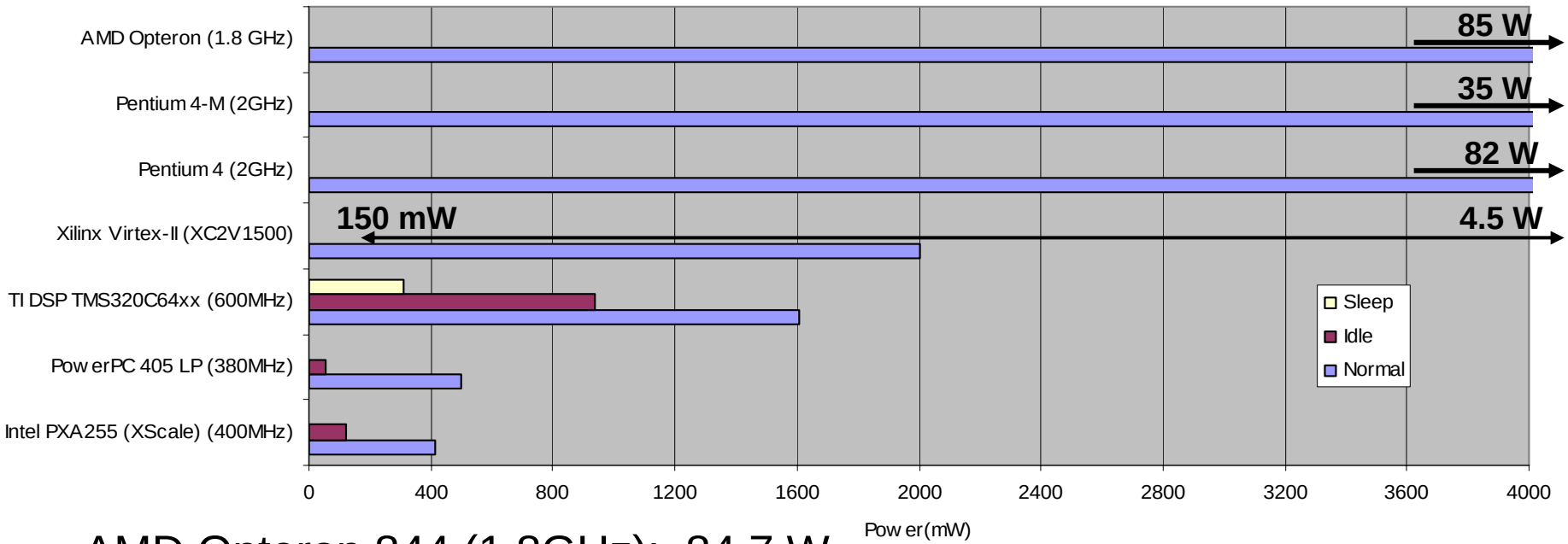
Outline

- Reconfigurable Computing
- Theoretical Foundations
- Network Intrusion Detection
- High Performance Embedded Signal Processing
- FPGA-based Designs for BLAS
- Conclusion

Floating-Point Device Options for High Performance Embedded Signal Processing

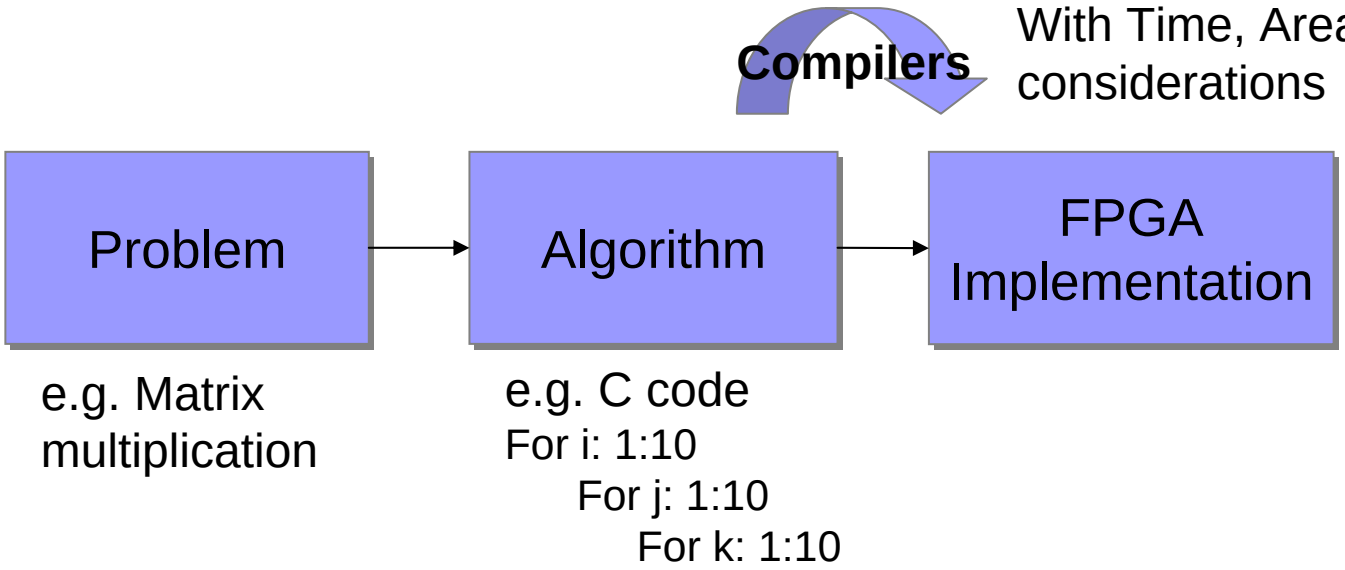


Power Dissipation of Various Devices



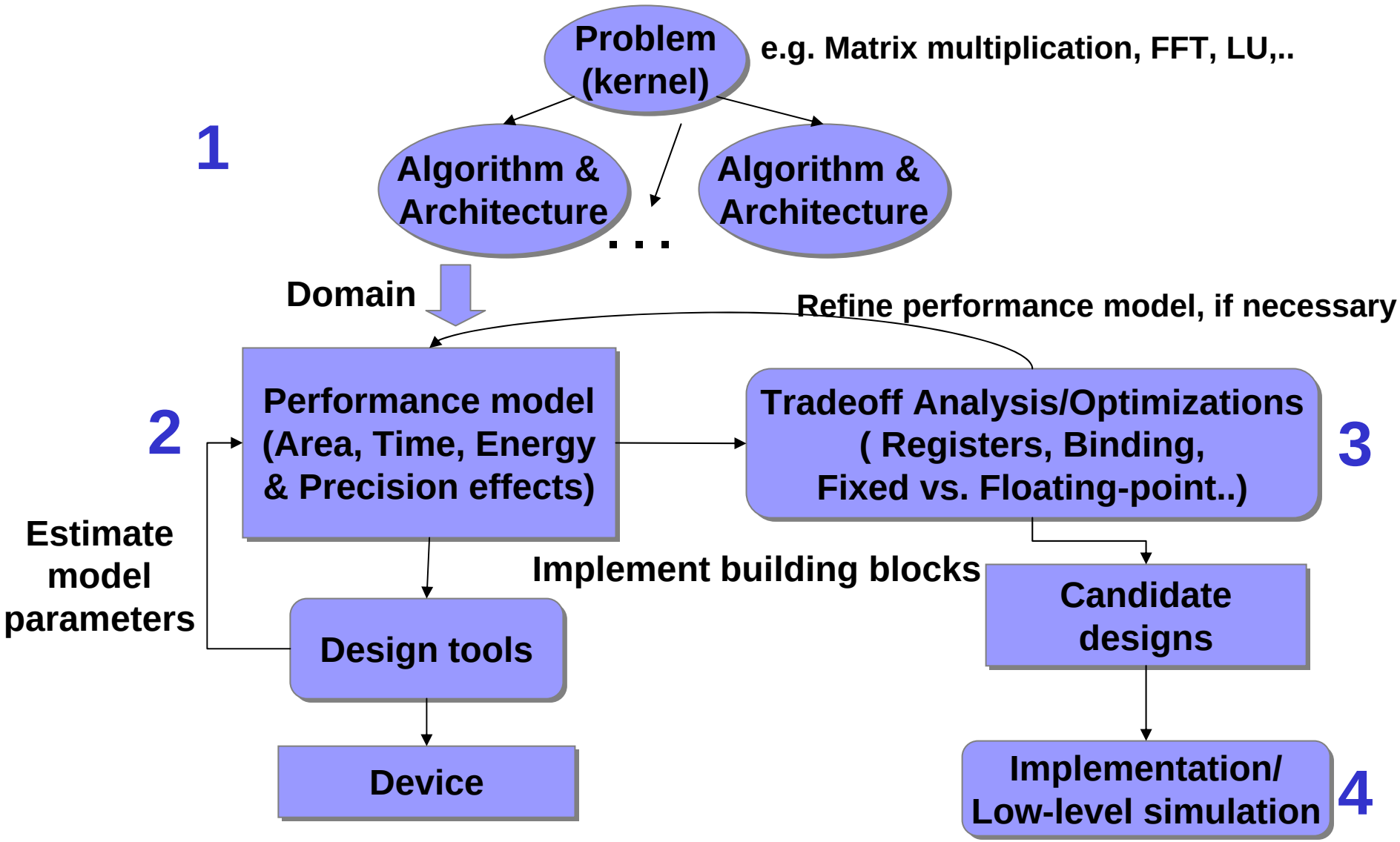
- AMD Opteron 844 (1.8GHz): 84.7 W
- Pentium 4 (3.06GHz): 81.8W, Pentium 4-M (2.4GHz): 35W
- TI DSP TMS32064xx (600MHz): 1.6W, four 16-bit MACs
- Xilinx Virtex-II (XC2V1500, 150MHz): 150mW-4.5W, 7680 slices, 48 18x18 embedded multipliers, 864 Kbit Block RAM
- Xilinx Virtex-II (XC2V8000, 150MHz): 525mW-20W, 46,592 slices, 168 18x18 embedded multipliers, 3 Mbit Block RAM

Problem Definition and Challenges



- Our focus
  - Design an **algorithm** for implementation on FPGAs
  - Minimize energy dissipation
- Large number of trade-offs
  - Memory
  - Interconnect
  - Amount of parallelism
  - . . .
- Energy performance models ?

The Approach: Overview



Example: Matrix Multiplication

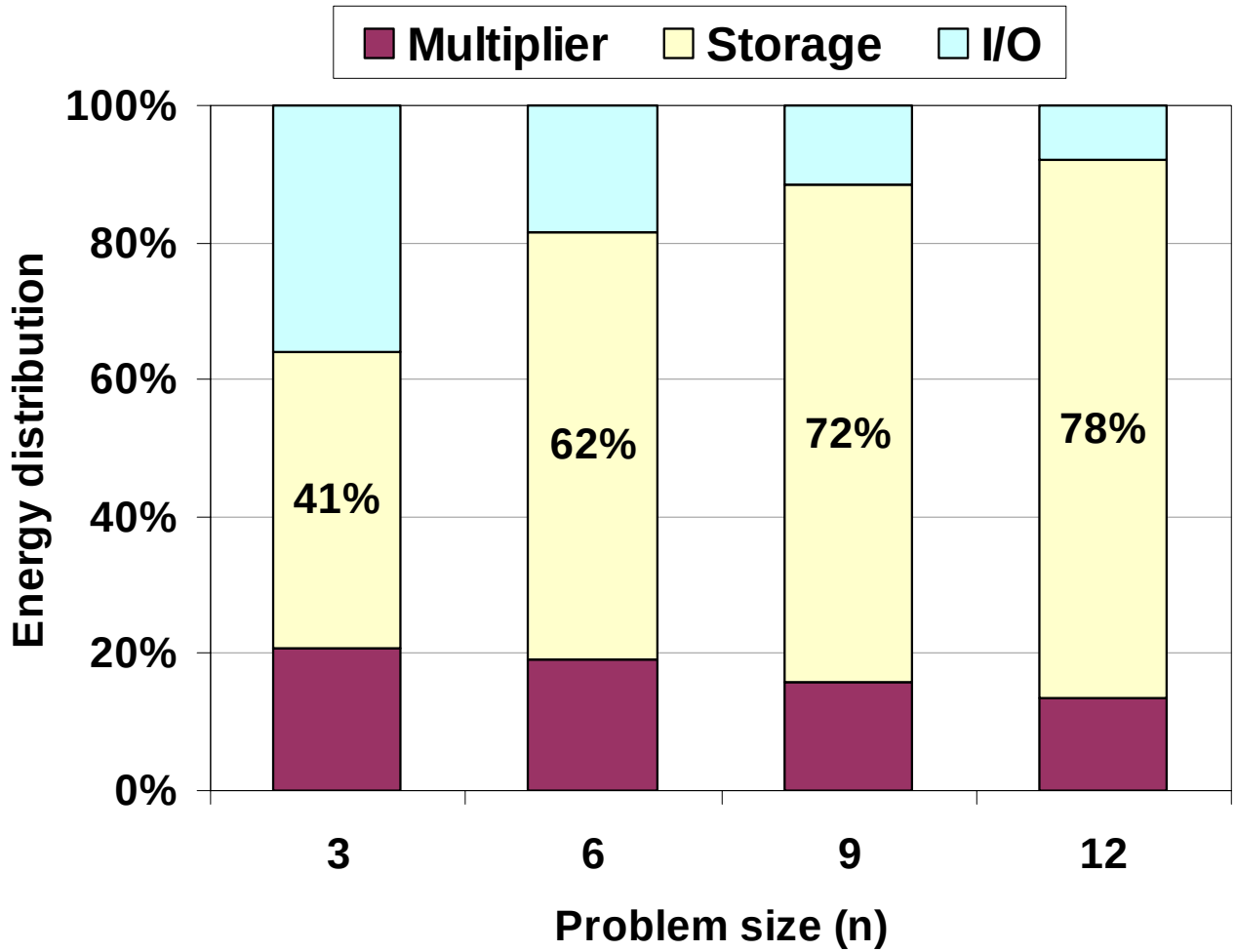
3x3 Matrix Multiplication

Design	Energy (nJ)	Latency (cycles)	Area (slices)
Xilinx	25.0	45	180
Design 1	24.5	15	487

- Xilinx:
  - Slows down registers, saves power
  - Block multiplier, power-efficient
  - Low level optimizations
- Design 1 (based on known Systolic Design):
  - Systolic architecture
  - Data in registers shift in every cycle (Energy?)
  - Speedup with more area, no savings in energy

Opportunity for Energy Savings

- Lot of energy dissipation in registers (storage)



Performance Comparison

- Target FPGA: XC2V1500, speed grade –5
- Xilinx designs based on 3x3 matrix multiplication (150MHz)
  - Block matrix multiplication is used for large problem sizes

Problem size (n)	Xilinx (150MHz)				Our designs (150MHz)			
	A	T	E	EAT	A	T	E	EAT
3	299	0.18	23.1	1.24	434	0.06	16.0	0.42
6	299	1.44	185.0	79.7	861	0.24	105.9	21.9
12	299	11.52	1479.8	5079.3	1699	0.96	775.0	1264.0
15	299	22.50	2890.3	19444.6	2083	1.50	1509.7	4717.1

A (slice), T (usec), E (nJ), EAT (10^{-12})

Matrix Multiplication (2004)

- **64-bit floating-point matrix multiplication on FPGA using our IEEE-754 compliant FP cores**

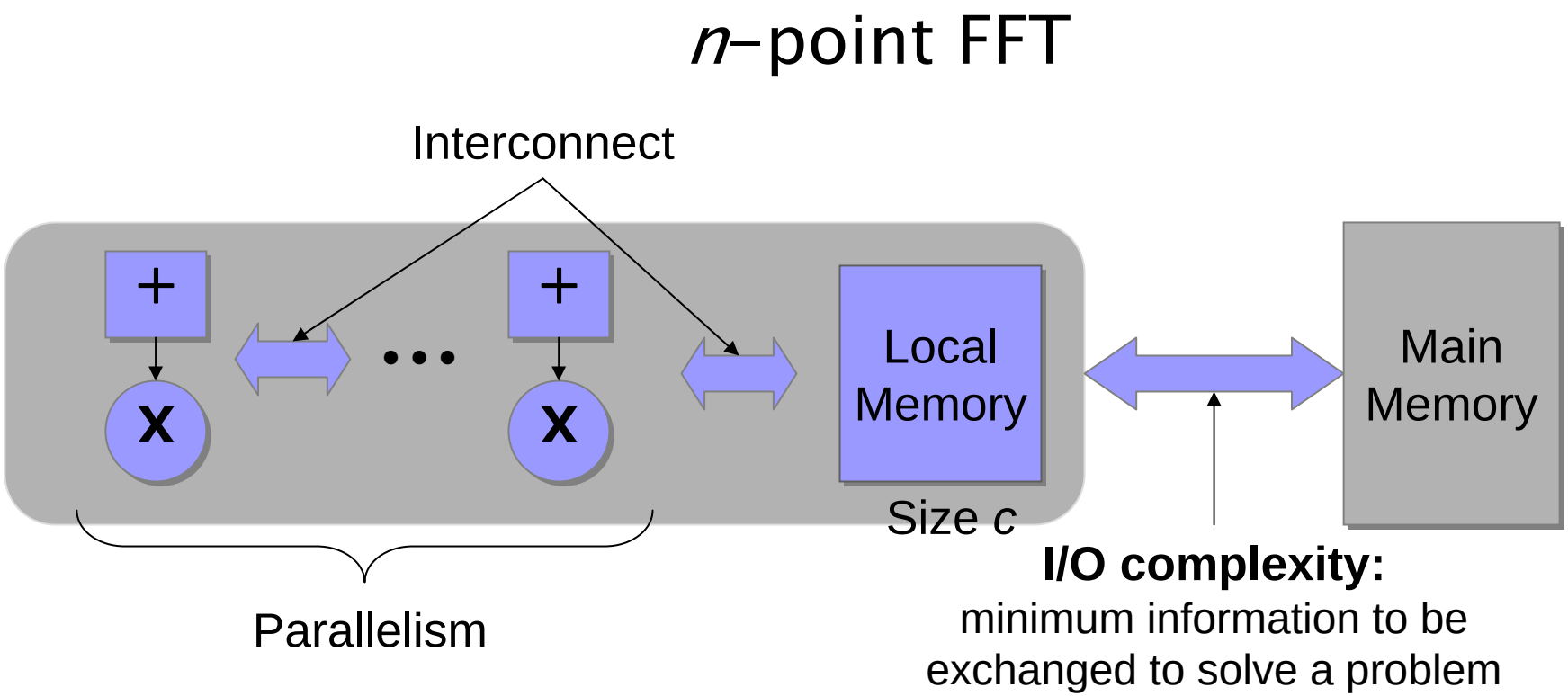
	FPGA XC2VP125-7 170 MHz	Pentium 4 SSE2 1.5 GHz*	AMD Athlon 1 GHz*
GFLOPS	5.7 – 10.5** (sustained)	2.0 (peak)	1.1 (peak)
Power (W)	26	54.7	60
GFLOPS/W	0.20 – 0.40**	0.036	0.018

- **Performance (in GFLOPS): 5x – 9x**
- **Performance/Power (in GFLOPS/W): 12-22x**

*From data sheets

**FP cores: least overhead – most compliant

I/O Complexity of FFT



For *n*-point FFT, I/O complexity = $\Omega(n \log n / \log c)$

Performance Comparison

Percentage increase/decrease in Area, Time, Energy and EAT in comparison with the Xilinx Designs

Problem size(n)	Our Designs Vp	Hp	Binding	Area (inc.)	Time (dec.)	E (dec.)	EAT (dec.)
16	1	2	SRAM	0.86x	1.0x	57%	2.71x
	4	2	SRAM	1.75x	4.0x	58%	5.45x
64	1	3	SRAM	2.10x	3.0x	72%	5.17x
	1	3	BRAM	1.49x	3.0x	78%	9.18x
	4	3	SRAM	5.27x	12.0x	77%	9.70x
	4	3	BRAM	3.89x	12.0x	78%	13.77x
256	1	4	BRAM	1.57x	3.0x	68%	5.94x
	4	4	BRAM	4.32x	12.0x	72%	9.77x
1024	1	5	BRAM	1.76x	3.0x	60%	4.25x
	4	5	BRAM	4.29x	12.0x	73%	10.43x

Challenge (4): Energy Estimation

Energy Performance Models

Accuracy Vs. Speed

Domain Specific Modelling

Hardware/Software Co-design

Outline

- Reconfigurable Computing
- Theoretical Foundations
- Network Intrusion Detection
- High Performance Embedded Signal Processing
- FPGA-based Designs for BLAS
- Conclusion

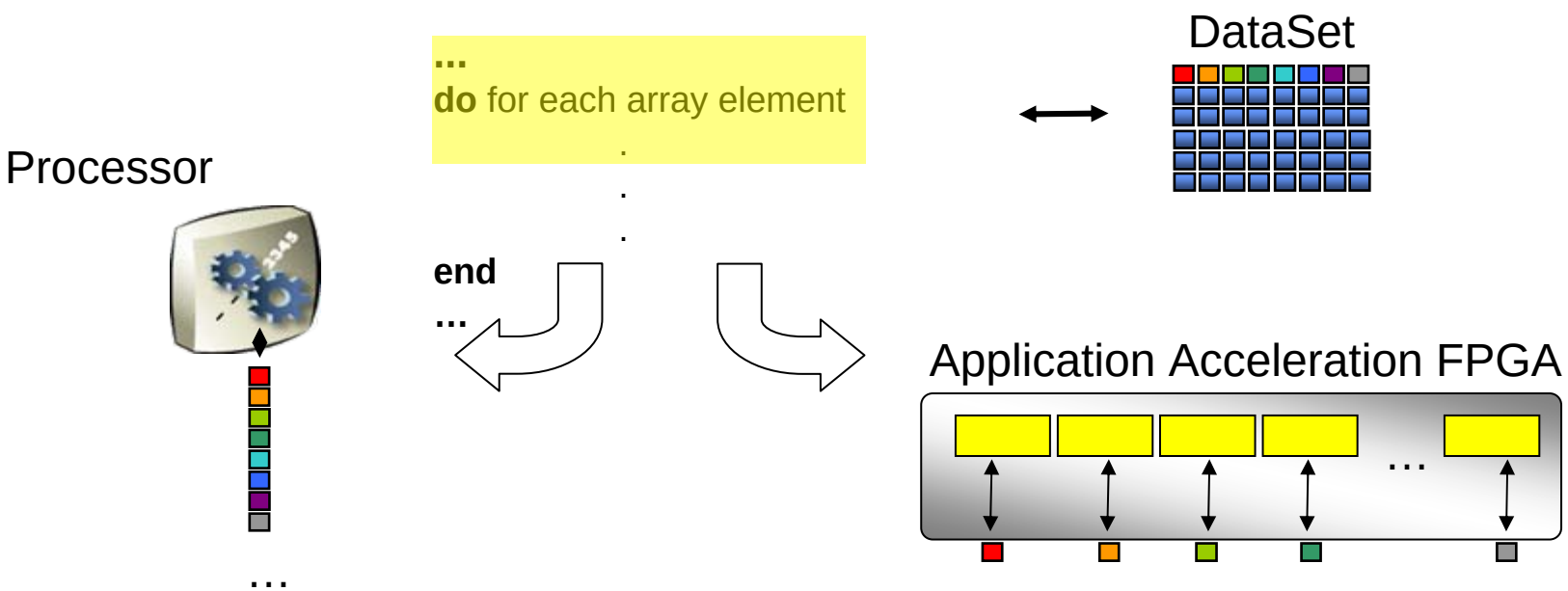
Floating-Point Based Applications on FPGAs

- More opportunities provided by FPGAs
  - Large number of embedded memory blocks
  - Large number of building blocks (multipliers,..)
  - Large number of I/O pins
  - Large amount of off-chip as well as on-chip memory bandwidth
- Current FPGAs have become attractive for floating-point based applications
  - Matrix algebra: dense matrix multiplication
  - Complex scientific applications: molecular dynamics
  - Highly competitive with microprocessors
  - FPGA-accelerated systems:

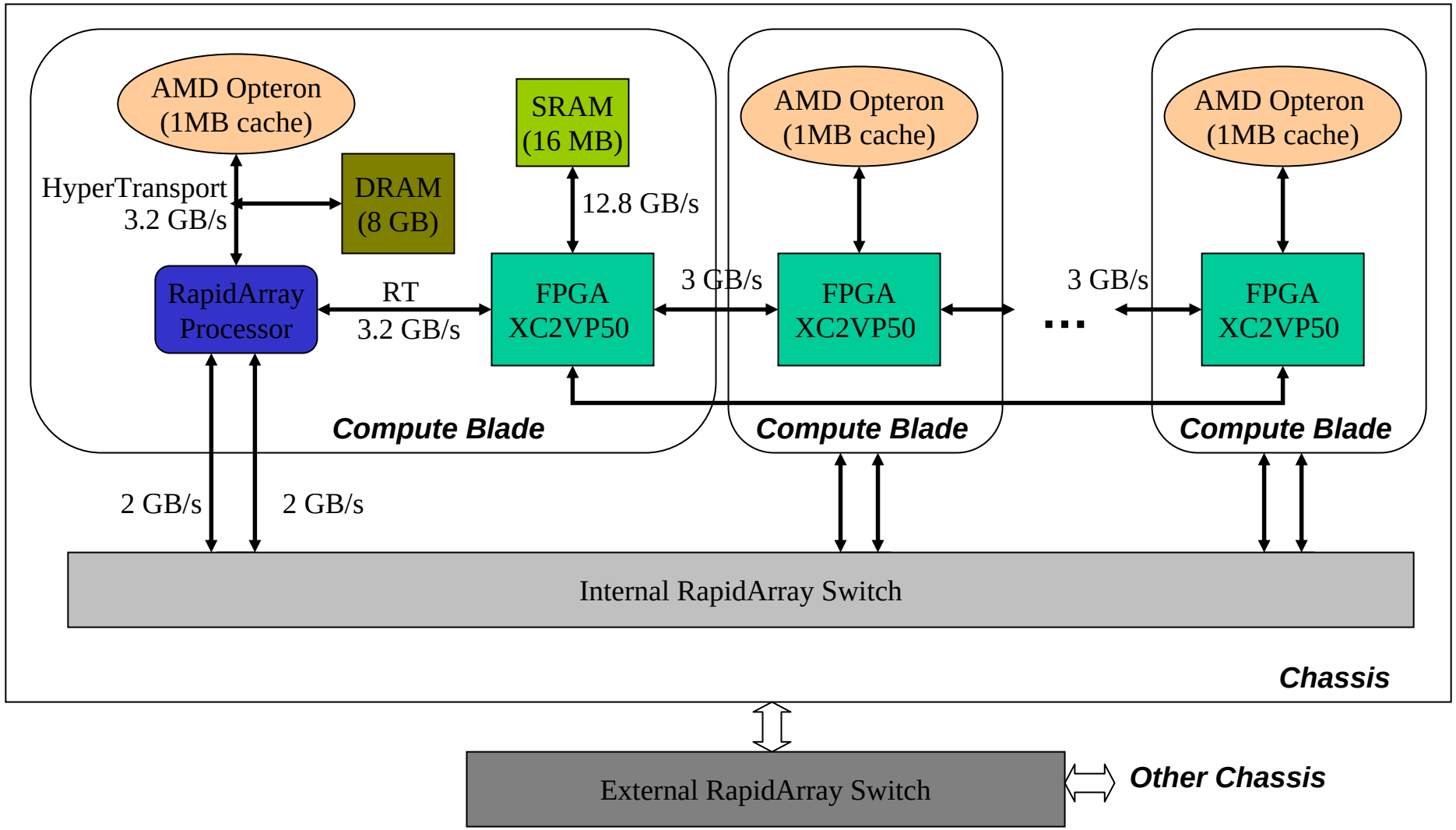
    SRC 6E/7, Cray XD1(XT3), SGI, Star Bridge Hypercomputer,…

Reconfigurable Computing Systems

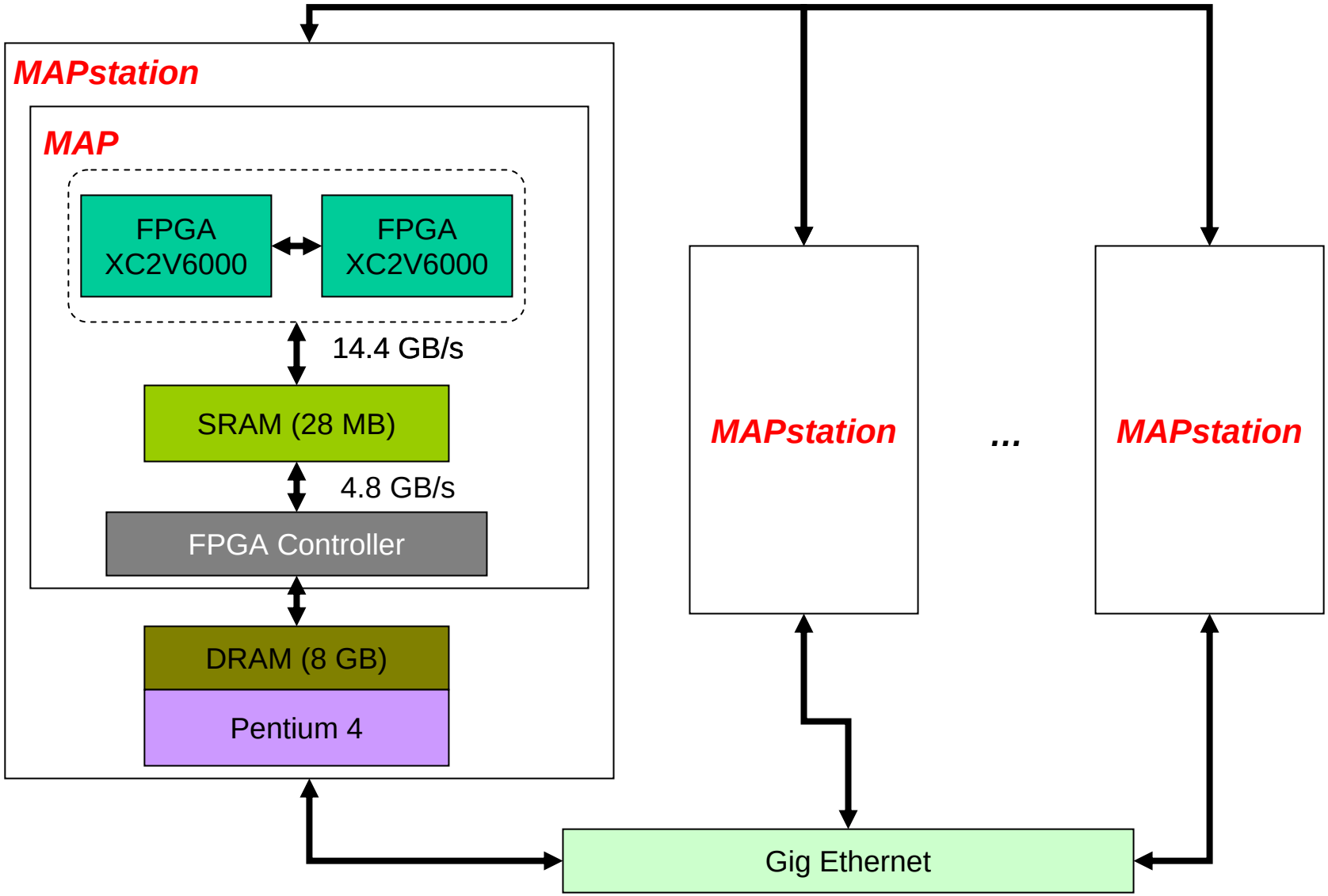
- Combine general-purpose processors, FPGAs, memory and interconnect
- FPGAs act as programmable processors or co-processors
  - Interconnect connects the processors and FPGAs
  - FPGAs have access to multiple levels of memory



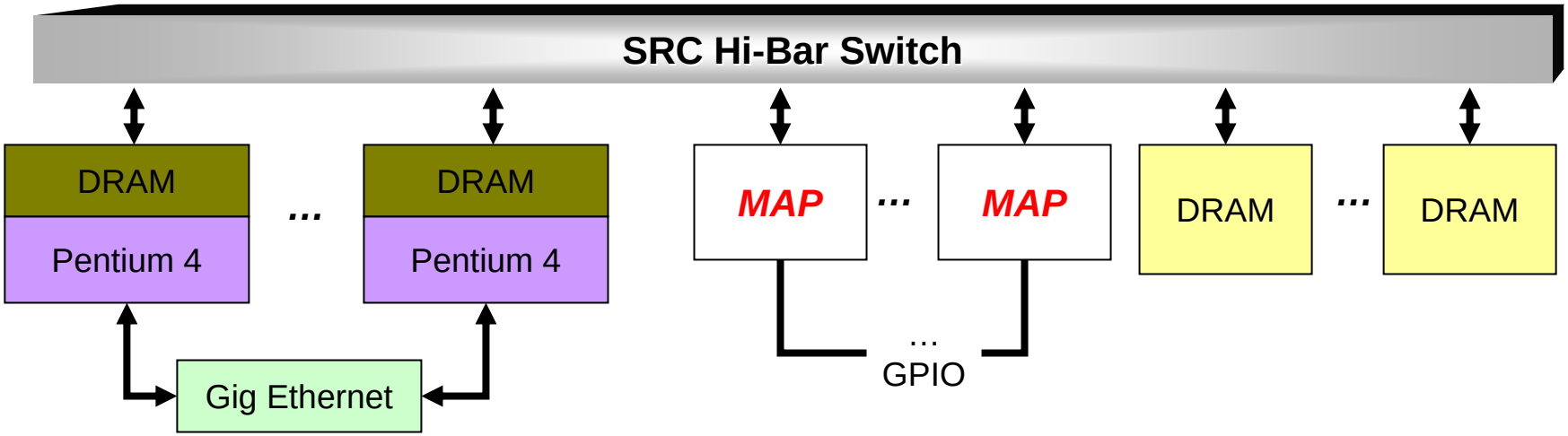
Cray XD1 – Hardware Architecture



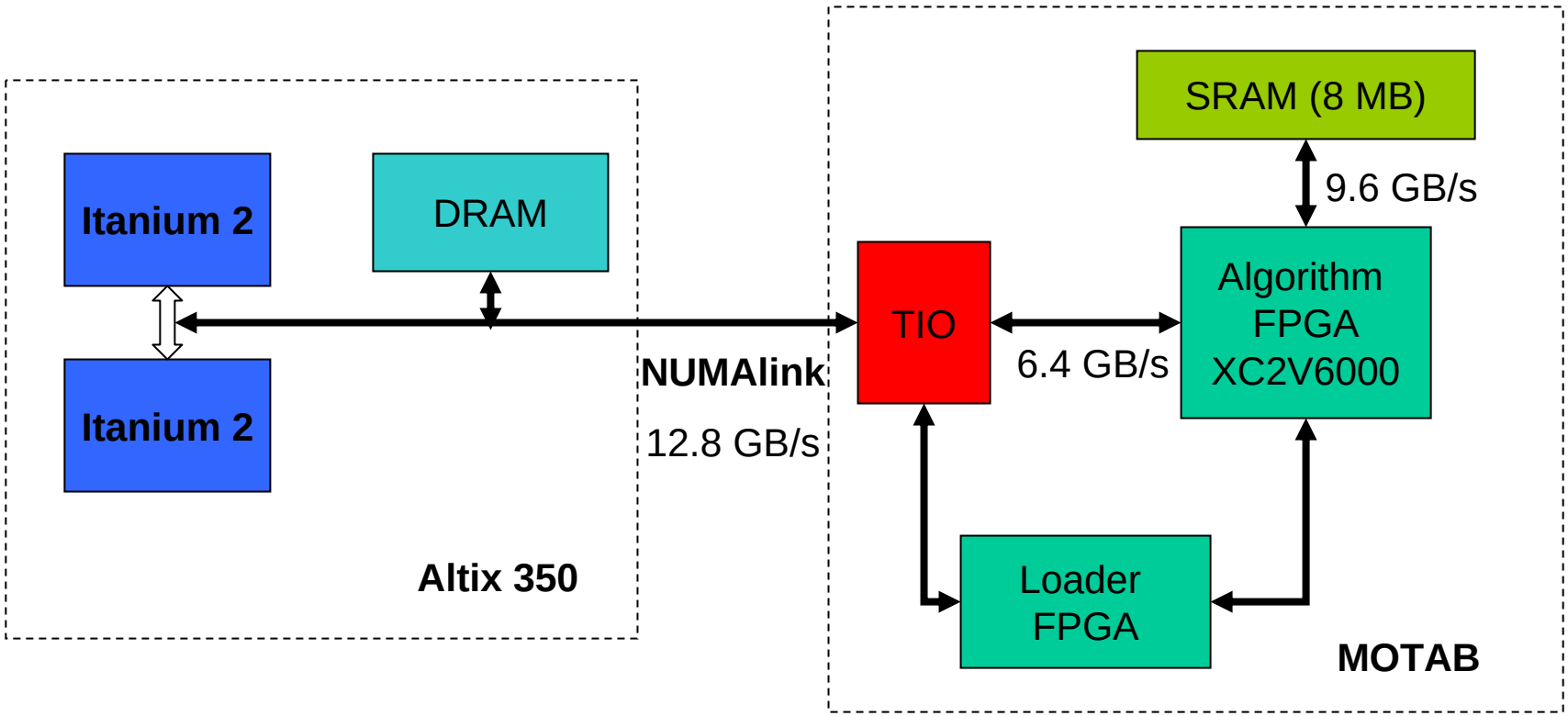
SRC 6E (1)


MAPstation
MAP
FPGA XC2V6000
FPGA XC2V6000
14.4 GB/s
SRAM (28 MB)
4.8 GB/s
FPGA Controller
DRAM (8 GB)
Pentium 4
MAPstation
...
MAPstation
Gig Ethernet

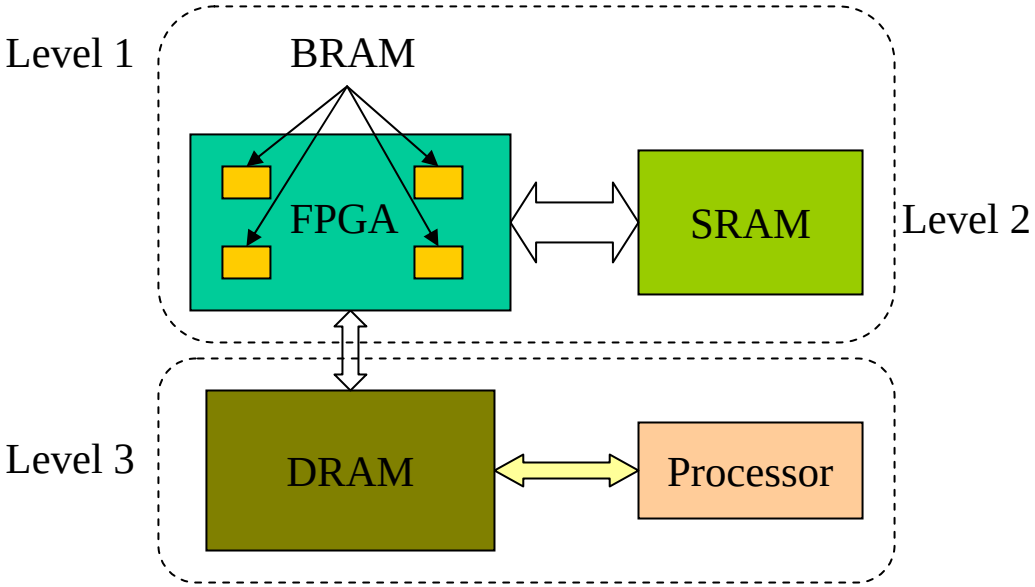
SRC 6E (2)


SRC Hi-Bar Switch
DRAM
Pentium 4
DRAM
Pentium 4
MAP
MAP
DRAM
DRAM
Gig Ethernet
GPIO

SGI RASC Brick



Memory Hierarchy



	SRC		Cray	
Level	Size	Bandwidth	Size	Bandwidth
1	648KB	260 GB/s	522 KB	209 GB/s
2	24 MB	14.4 GB/s	16 MB	12.8 GB/s
3	8 GB	2.8 GB/s	8 GB	3.2 GB/s

BLAS Library

- Basic Linear Algebra Subprograms
  - Level 1: vector-vector operations
  - Level 2: matrix-vector operations
  - Level 3: matrix-matrix operations
- Importance of BLAS
  - BLAS operations used in a wide range of software including LINPACK
  - High performance BLAS library is essential for performance improvement of many scientific applications

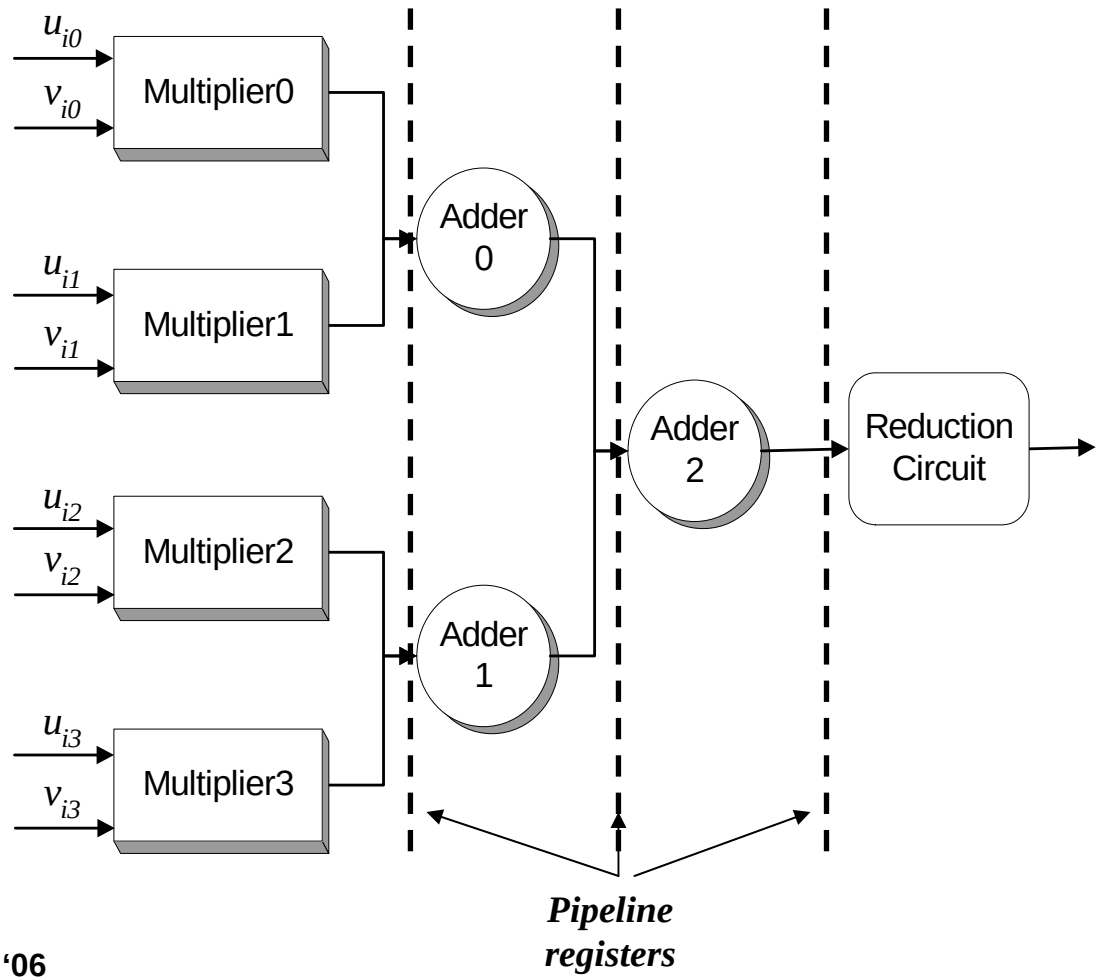
FPGA-Based BLAS

- Suitable for hardware acceleration
  - Loop-oriented: can benefit from spatial parallelism
  - Simple control logic: control structure is relatively small compared with the data path
  - Data-oblivious: algorithm execution does not depend on input data
- Design challenges
  - Various hardware constraints: memory, configurable logic, no. of FPUs, clock rate, bandwidth, no. of I/O pads
  - Various design tradeoffs:
    1. Clock rate vs. area
    2. BRAM, no. of copies, clock rate
    3. Memory vs. logic
    4. Reduce or not to Reduce

Rich set of design tradeoffs See SC '05

Level 1 BLAS

- Dot product $u \times v = \sum_{i=0}^{n-1} u_i v_i$



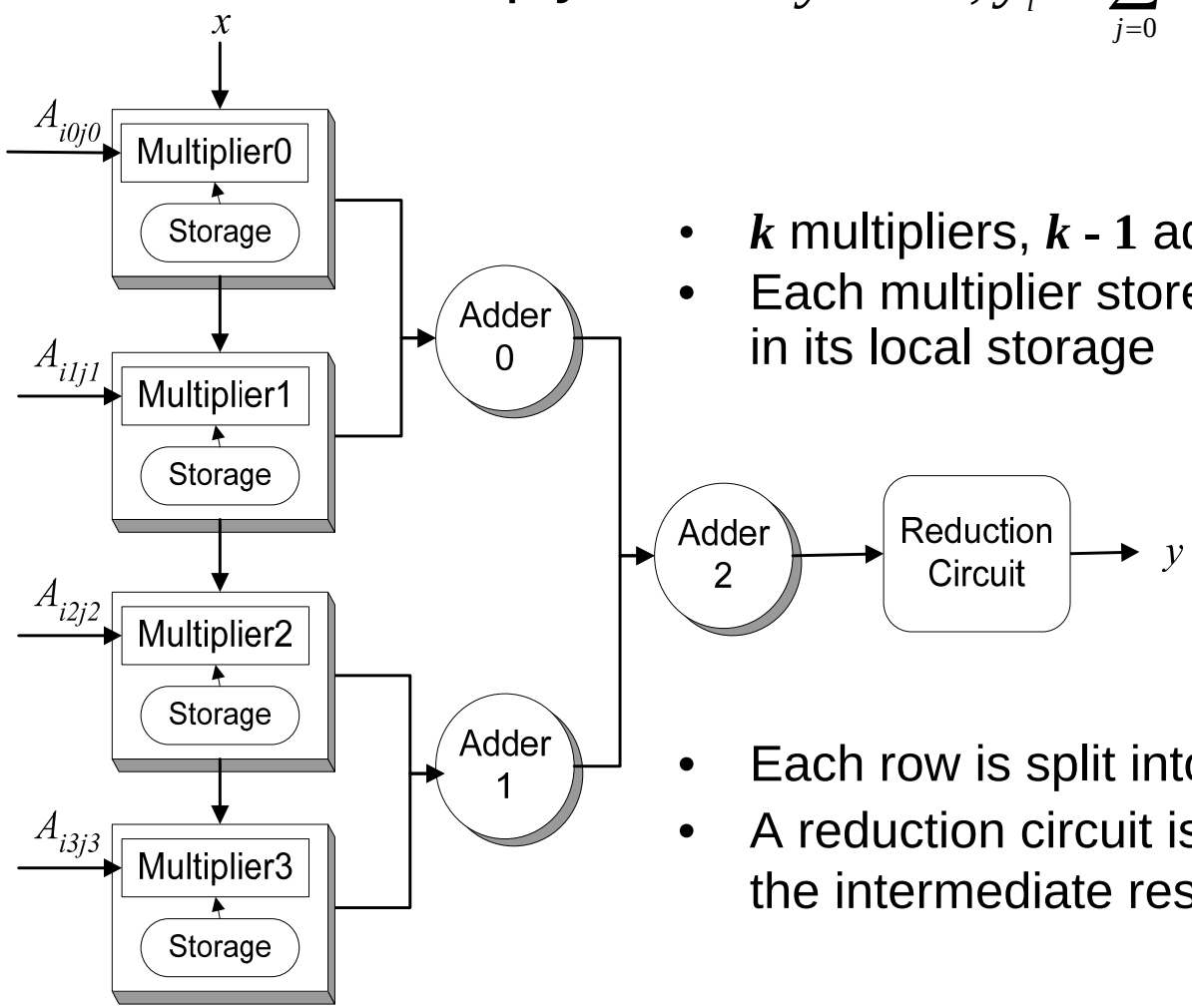
- Floating-point adders and multipliers are pipelined
- Additions, multiplications and I/O operations are overlapped

- k multipliers, k -1 adders
- By varying k, the architecture can match available memory bandwidth

Level 2 BLAS

- Matrix-vector multiply

$$y = Ax, y_i = \sum_{j=0}^{n-1} a_{ij} x_j (0 \le i \le n-1)$$



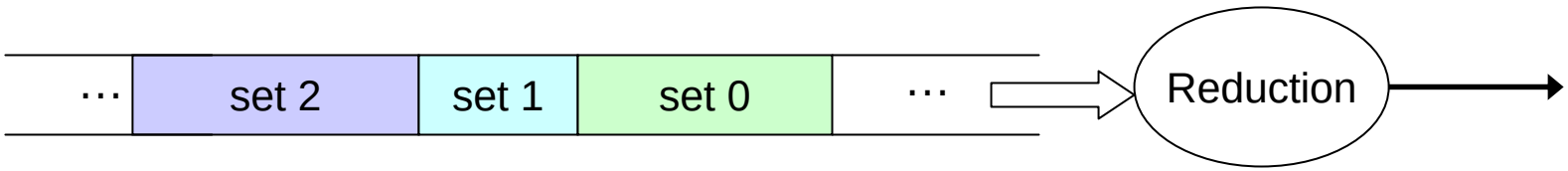
- k multipliers, k - 1 adders
- Each multiplier stores blocks of vector x in its local storage

- Each row is split into subrows of length k
- A reduction circuit is used to accumulate the intermediate results of the subrows

Reduction Circuit

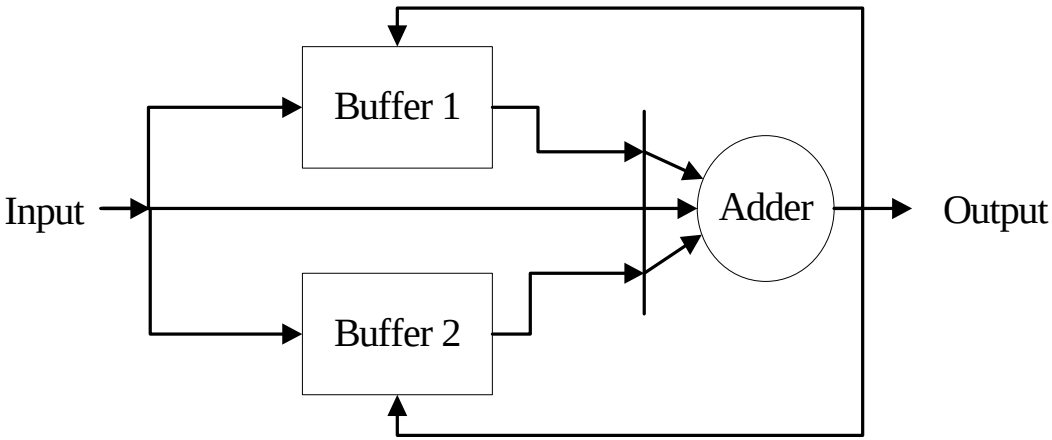
- Problem definition

  Reduce p input sets, set i contains s_i floating-point values. The inputs are delivered sequentially



- Design challenges
  - Reduction using pipelined floating-point adders may result in data hazards
  - Throughput: accept one new input in each clock cycle without stalling
  - Area: number of adders, buffer size

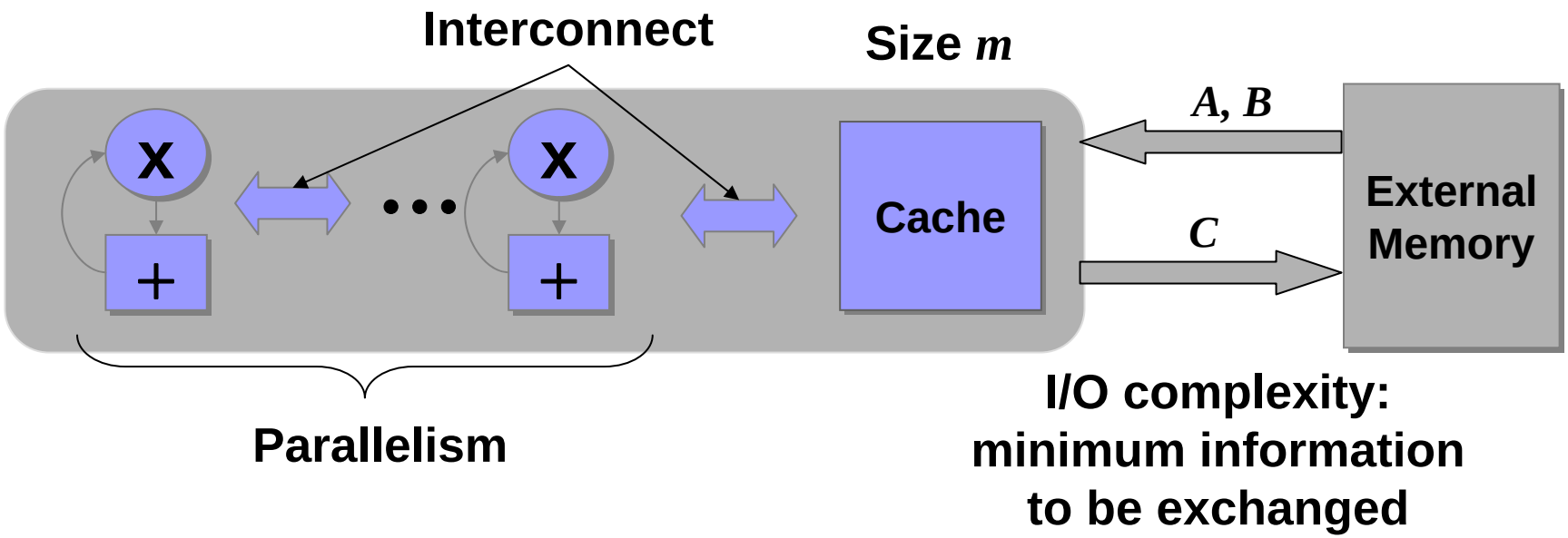
Our Solution



- Only one adder is needed
- Two buffers of size α^2 (α : no. of pipeline stages in the adder)
- No data hazards
- No data overflow in the buffer
- Use of the adder is conflict-free
- Reduces $\boldsymbol{p}$ input sets in at most $\sum_{i=0}^{p-1} s_i + 2\alpha^2$ clock cycles

Level 3 BLAS

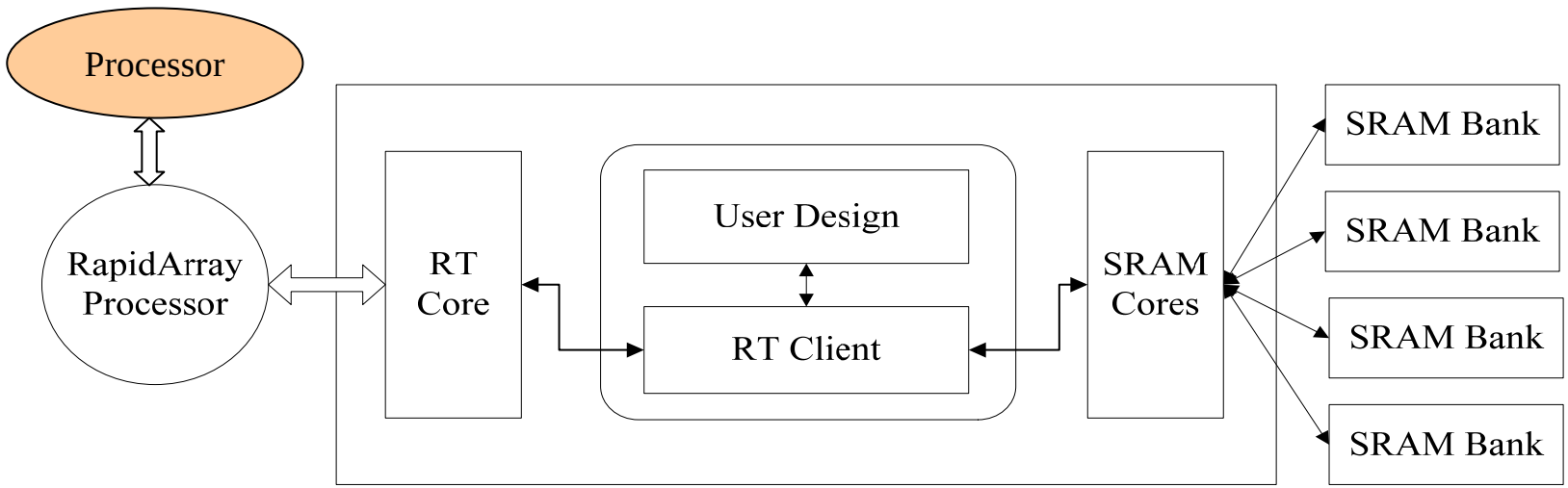
- Matrix multiply: $C = AB$
  - A, B, C: $n \times n$ dense matrix



Theorem (Hong and Kung): For $n \times n$ matrix multiplication
I/O complexity = $\Omega(n^3/\sqrt{m})$, $1 \leq m \leq n^2$

Implementation (1)

- Setup
  - Tools: Xilinx ISE 7.1i and Modelsim 5.7
  - Target device: Virtex-II Pro XC2VP50 (in Cray XD1)
- Our own floating-point adders/multipliers
  - Double-precision (64-bit)
  - Various levels of compliance with IEEE-754 standard



Implementation (2)

- AMD Opteron Processors
  - 2.6 GHz, 64 KB L1 cache, 1 MB L2 cache
- Software
  - GCC 3.3.3
  - MPICH 1.2.6
- Performance metrics
  - Area: number of slices (cluster of logic cells)
  - Achievable clock speed
  - Memory bandwidth
  - Sustained MFLOPS/GFLOPS
- No Engineering Optimizations

Using One Node in XD1

- Performance of actual implementations
  - Level 2 BLAS: 80% of the latency is spent in moving data from DRAM to SRAM
  - Level 3 BLAS: overlap most I/O with computation

BLAS	Level 2	Level 3
No. of Multipliers/PEs, ***k***	4	8
Area/% of total device area	13772/58%	21029/89%
Clock Speed (MHz)	164	130
SRAM Bandwidth	5.9 GB/s	2.1 GB/s
DRAM Bandwidth	1.3 GB/s	24.3 MB/s
Sustained Performance	262 MFLOPS	2.06 GFLOPS

Performance Comparison

- Level 2 BLAS
  - Determined by memory bandwidth, $\cong 2bw$ FLOPS
  - Our design: 80% of the peak performance
- Level 3 BLAS
  - Peak perf.= Maximum no. of FPUs/chip **X** maximum clock speed of FPUs ***(Assumption: no memory or routing constraints)***

    4.4 GFLOPS for XC2VP50
  - Our design: 46.7% of the peak performance
- Performance of general-purpose processors for matrix multiply*:
  - A 3.2 GHz Xeon with 1 MB L3 cache: 5.5 GFLOPS
  - A 3.0 GHz P4 with 512 KB L2 cache: 5.0 GFLOPS

**** Performance Benchmarks for Intel Math Kernel Library*, White Paper**

Using Multiple Nodes in XD1

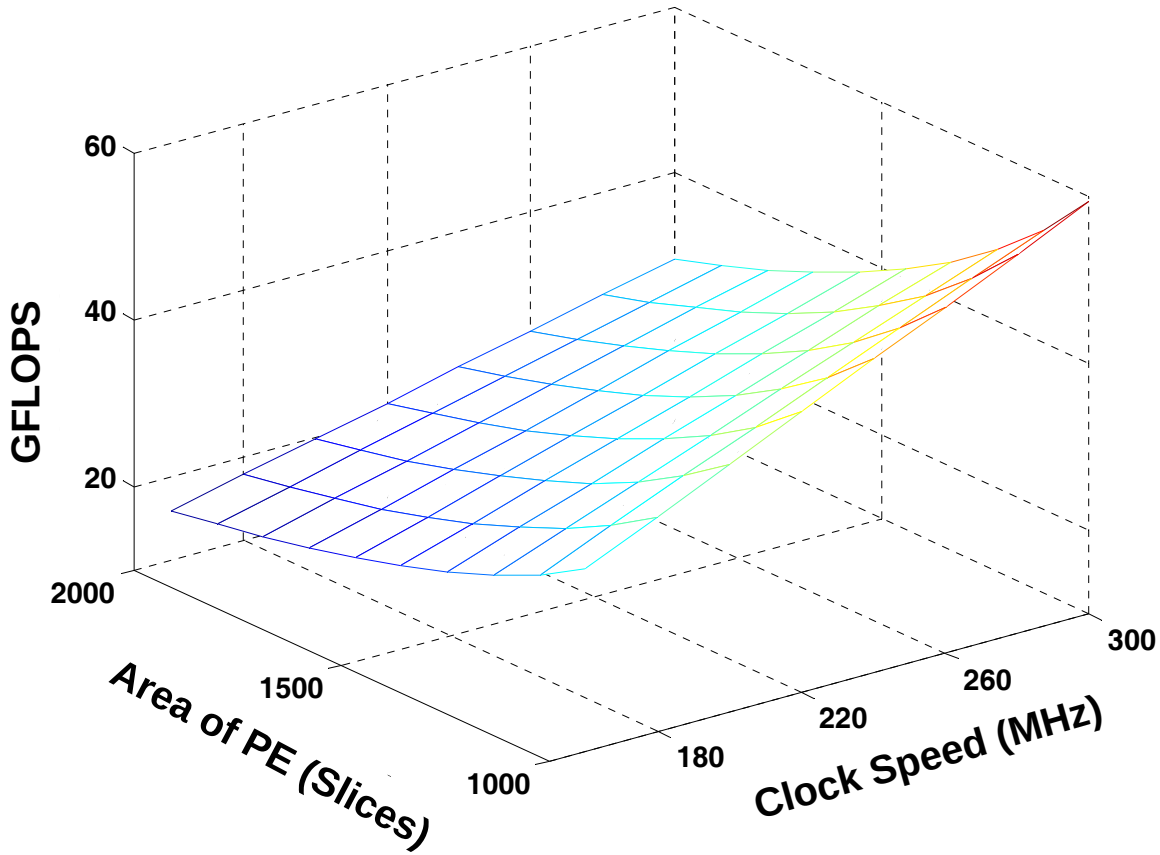
- Level 2 BLAS
  - Partition the matrix into multiple column blocks
  - MPI communication time is negligible compared with I/O time and computation time
  - Performance constrained by ***DRAM memory bandwidth***
  - Performance: ***238*** MFLOPS/FPGA
- Level 3 BLAS
  - MPI communication time and I/O time can be overlapped with the computation time
  - Constrained by ***computational power of FPGA***
  - Using nodes in one chassis: ***2.05*** GFLOPS/FPGA
  - Using nodes across two chassis: ***2.03*** GFLOPS/FPGA
  - Total Performance of a chassis: ***12.3*** GFLOPS

Performance Prediction (1)

- Many opportunities for performance improvement
- Using Improved floating-point units
  - Design of FPUs has no effect on the architecture
  - Improved FPUs can be easily plugged in
- Using larger, faster devices
  - Performance of our design scales with the hardware resources
  - Higher clock speed results in higher performance
  - Techniques are not device-specific
- Bandwidth is fixed at current value

Performance Prediction (2)

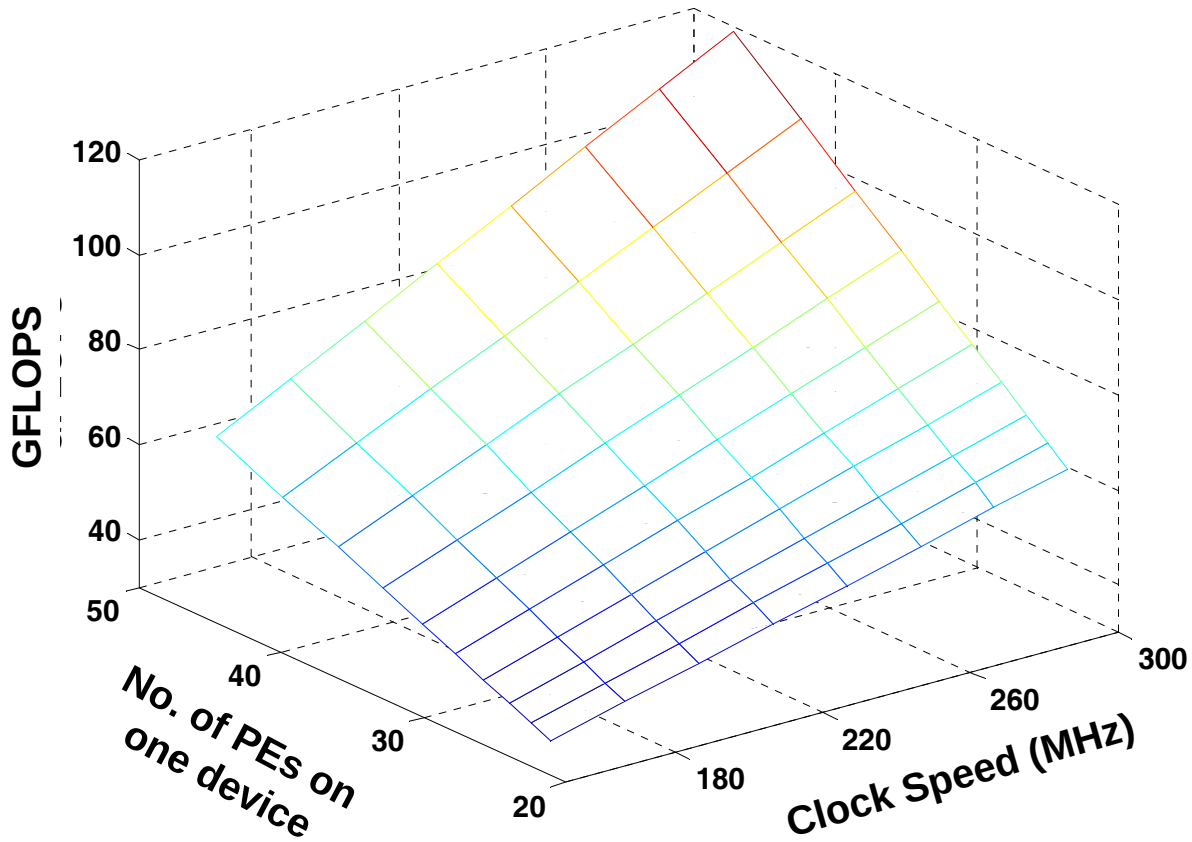
- Improved FPUs are used (in current architecture) in XD1



Projected Sustained Performance of a Chassis, for 64-bit Matrix Multiply

Performance Prediction (3)

- Larger, faster devices are used in XD1



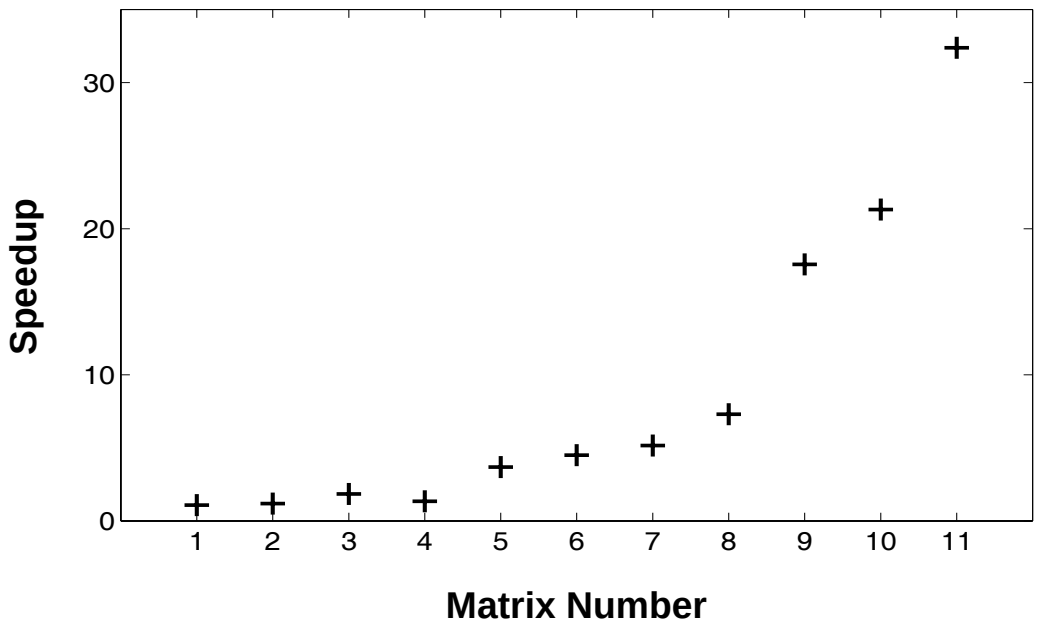
Projected Sustained Performance of a Chassis, for 64-bit Matrix Multiply

Sparse Matrix Computations

- Intel Itanium 2 system
  - Peak memory bandwidth of 8.5 GB/s
  - Vector x for each test matrix fits in L2 cache
  - SPARSITY developed by UC Berkeley
- Our design
  - $k = 4$
  - 50% of the device area
  - Memory bandwidth of 8 GB/s
  - Achieves higher speedup for more irregular matrices

ACM FPGA 2005

Speedup over a 900 MHz Intel Itanium 2



Challenge (5): System Level Issues

System Level Model (memory access?)

Programming Models/ Tools

Reconfigurable Computing: A decade of Progress

- Devices
- FPGA Vs ASIC
- FPGA Design Flow
- Application Accelerators

- Dynamic Reconfiguration ?
- Abstractions ?
- Commodity ?
- (Sustained) Niche Area ?

Thank you!

Questions

ceng.usc.edu/~prasanna