

Research Infrastructures for Accelerator-Centric Architectures

David Brooks, Brandon Reagen, Yakun Sophia Shao,
Kevin Skadron, Liang Wang



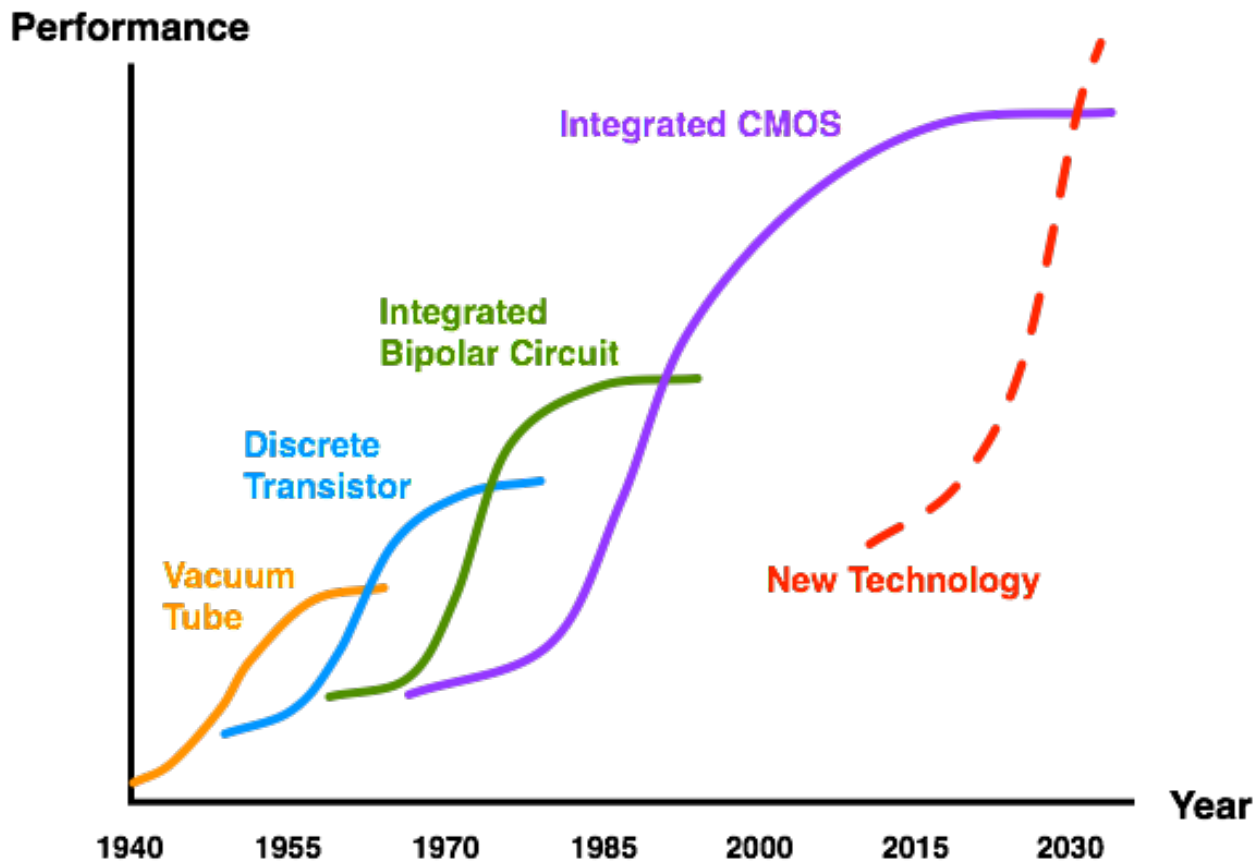
HARVARD
UNIVERSITY



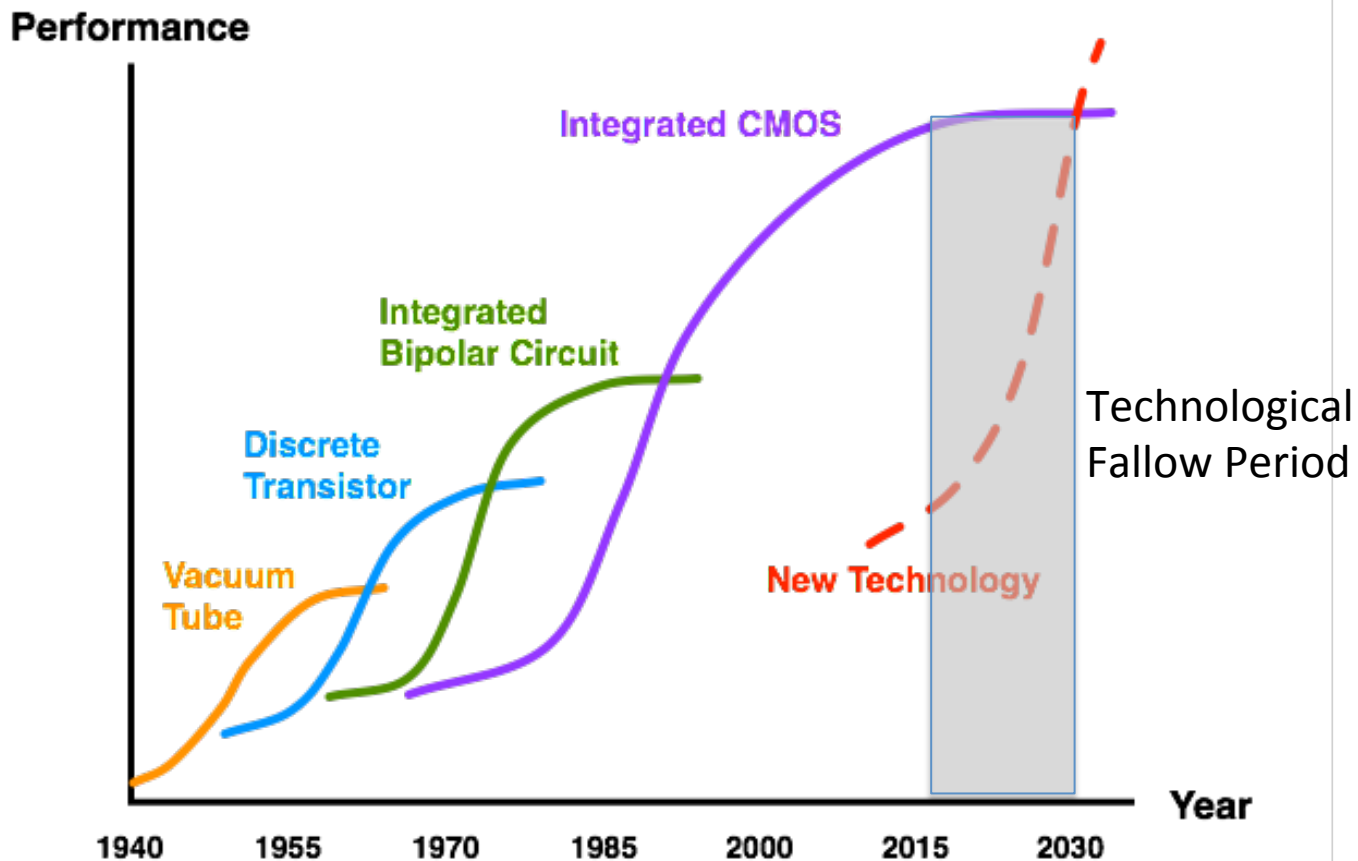
Tutorial Outline

Time	Topic
8:30 am – 9:00 am	Introduction
9:00 am – 10:00 am	Pre-RTL Simulation Framework: Aladdin
10:00 am – 10:30 am	Break
10:30 am – 11:30 am	CAD & Benchmarks: HLS & MachSuite
11:30 am – 12:00 pm	Aladdin Validation and Case Studies
12:00 pm – 1:30 pm	Lunch
1:30 pm – 2:15 pm	Analytical Modeling: Lumos
2:15 pm – 2:45 pm	Workload Characterization Tool: WIICA
2:45 pm – 3:00 pm	Hands-on Exercise: Set up
3:00 pm – 3:30 pm	Break
3:30 pm – 5:00 pm	Hands-on Exercise

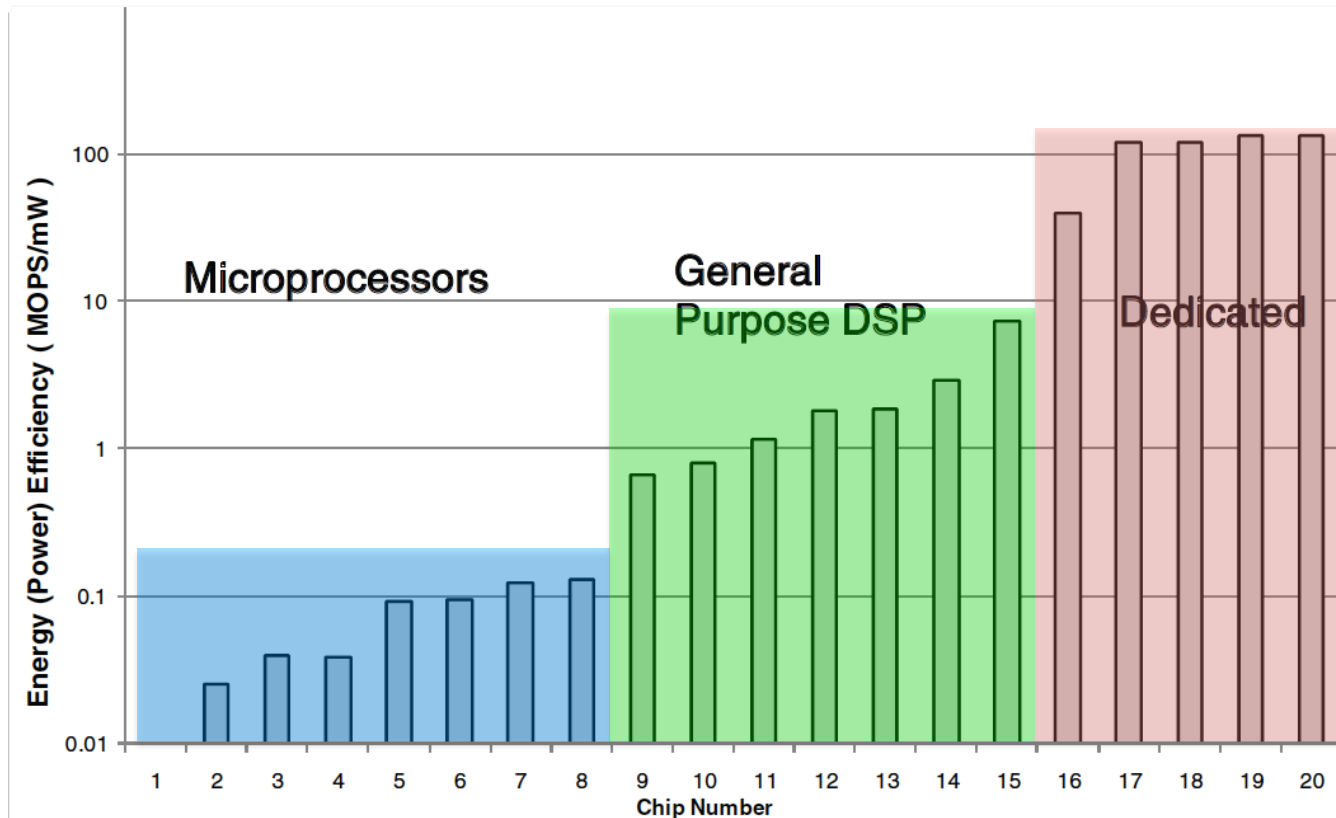
CMOS Technology Scaling



CMOS Technology Scaling



Potential for Specialized Architectures



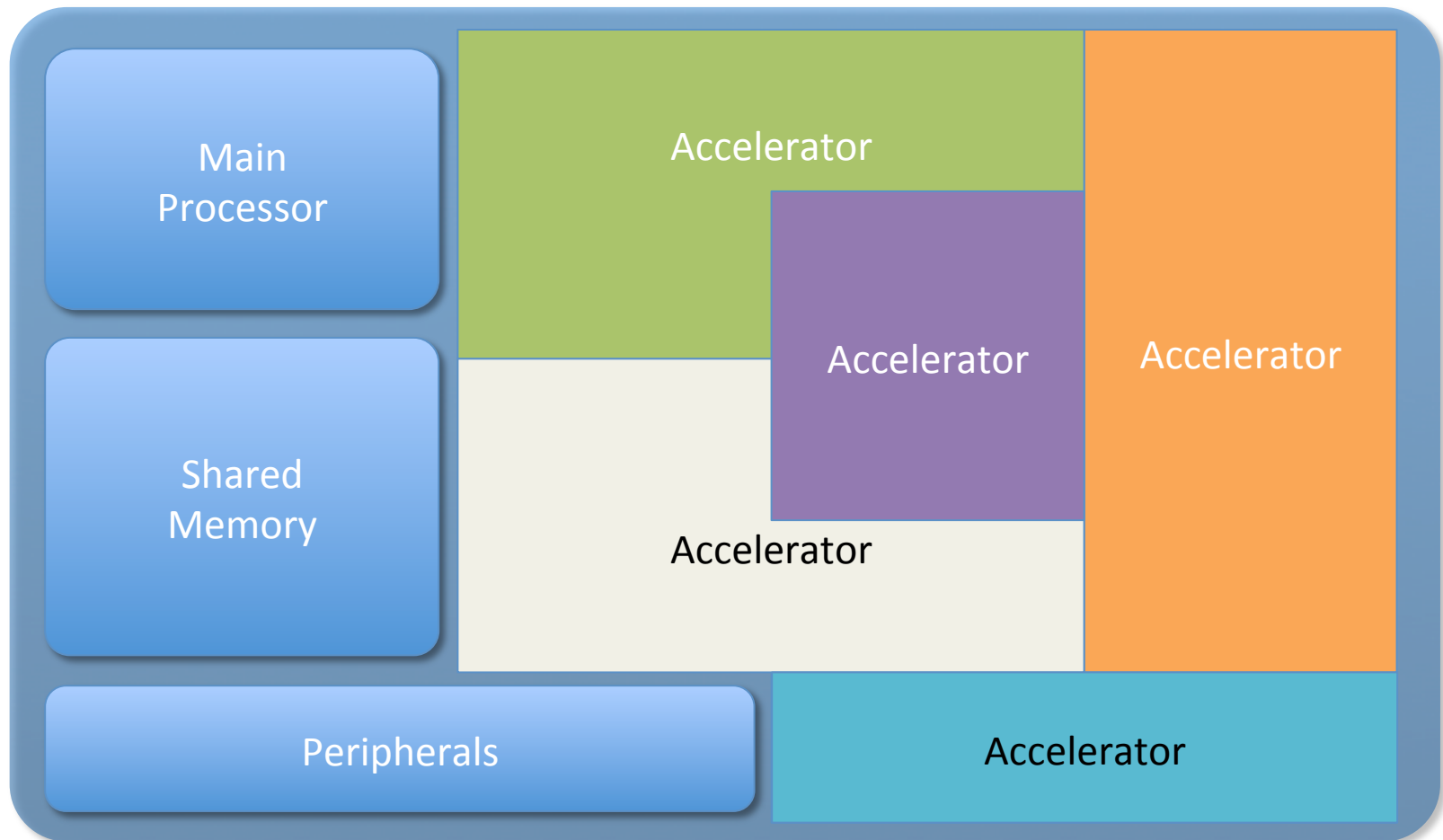
16	Encryption
17	Hearing Aid
18	FIR for disk read
19	MPEG Encoder
20	802.11 Baseband

[Brodersen and Meng, 2002]

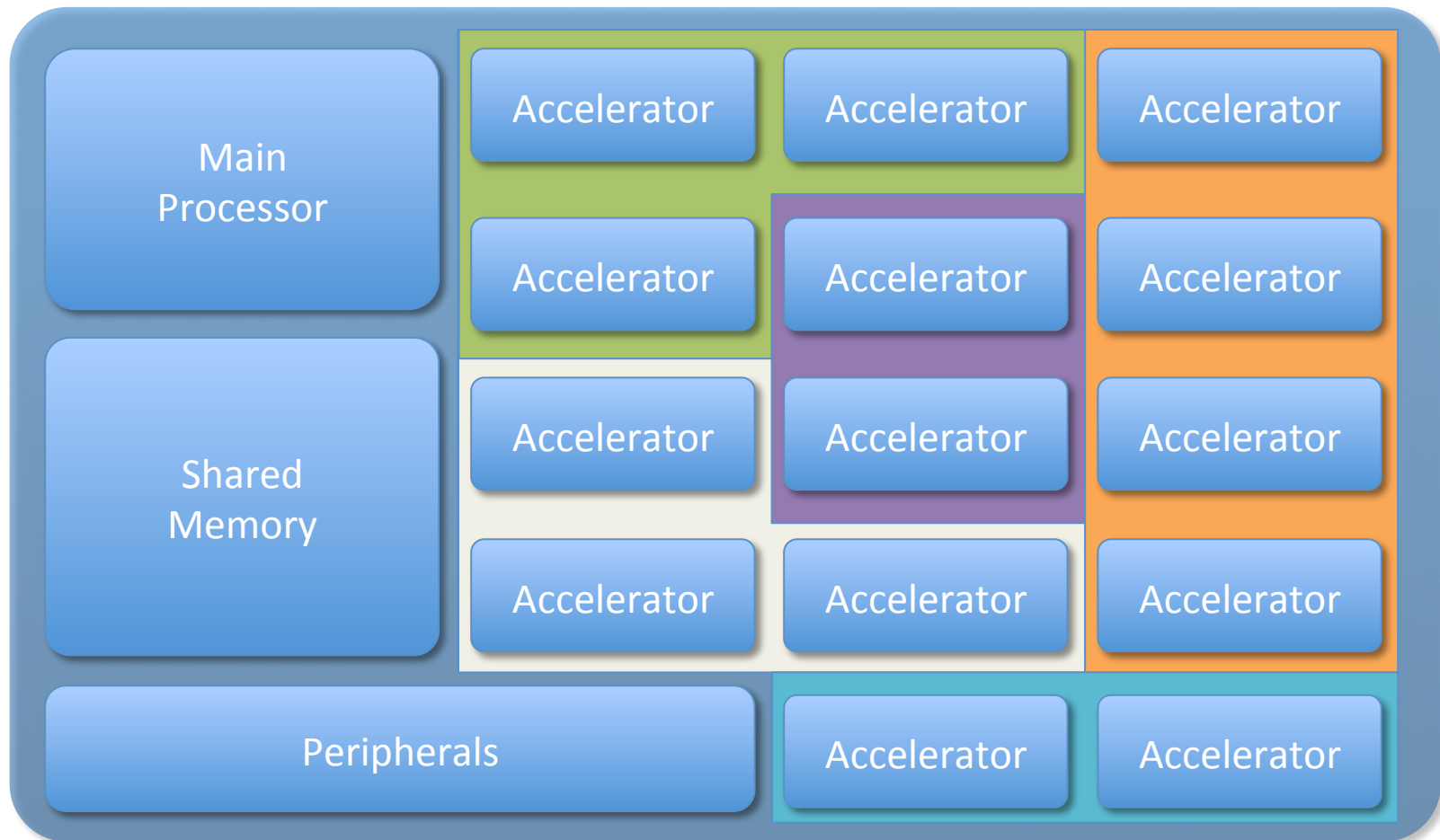
Challenges in Accelerators

- Flexibility
 - Fixed-function accelerators are only designed for the target applications.

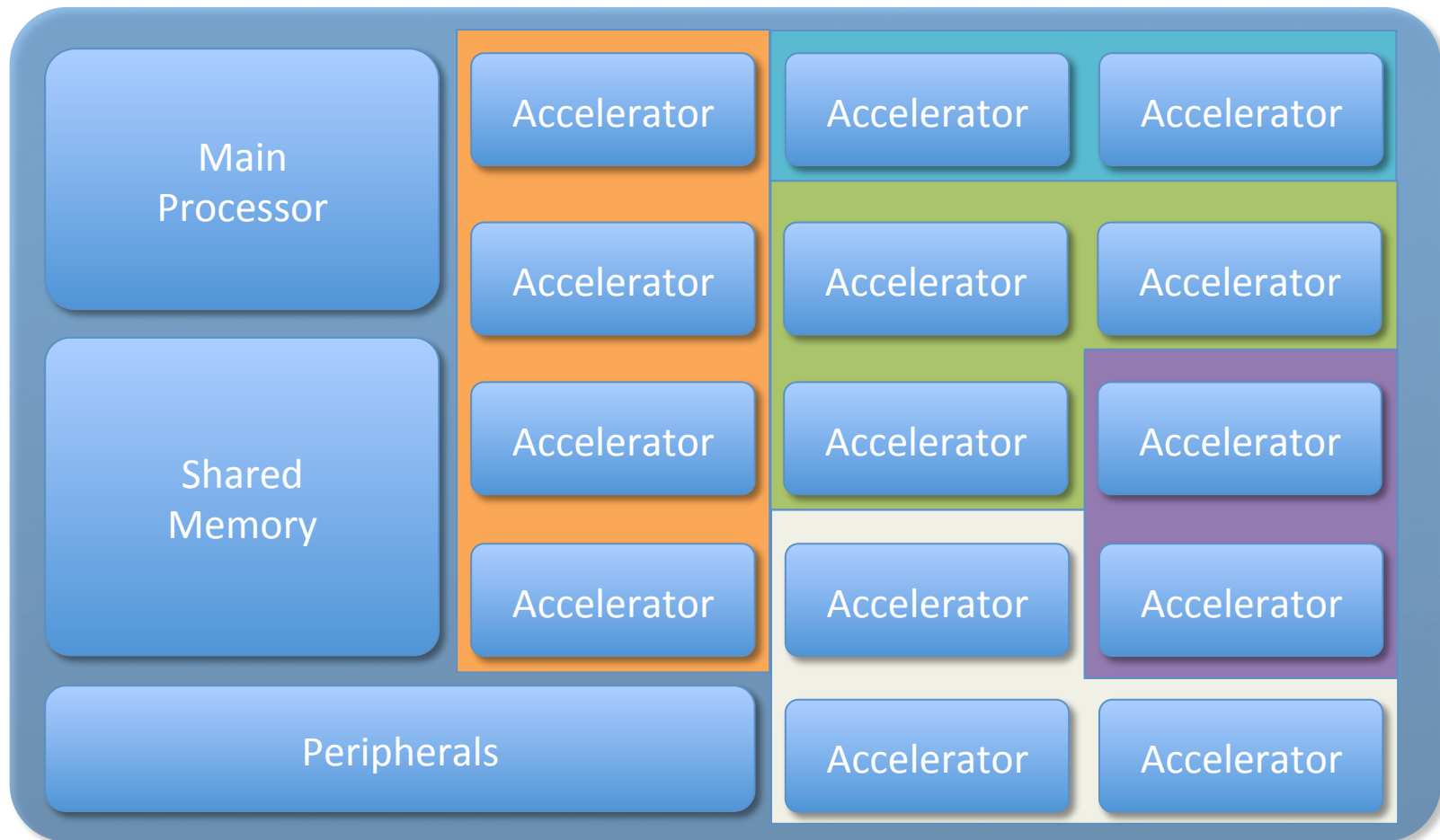
Decomposing accelerator-based systems for flexibility



Decomposing accelerator-based systems for flexibility



Decomposing accelerator-based systems for flexibility

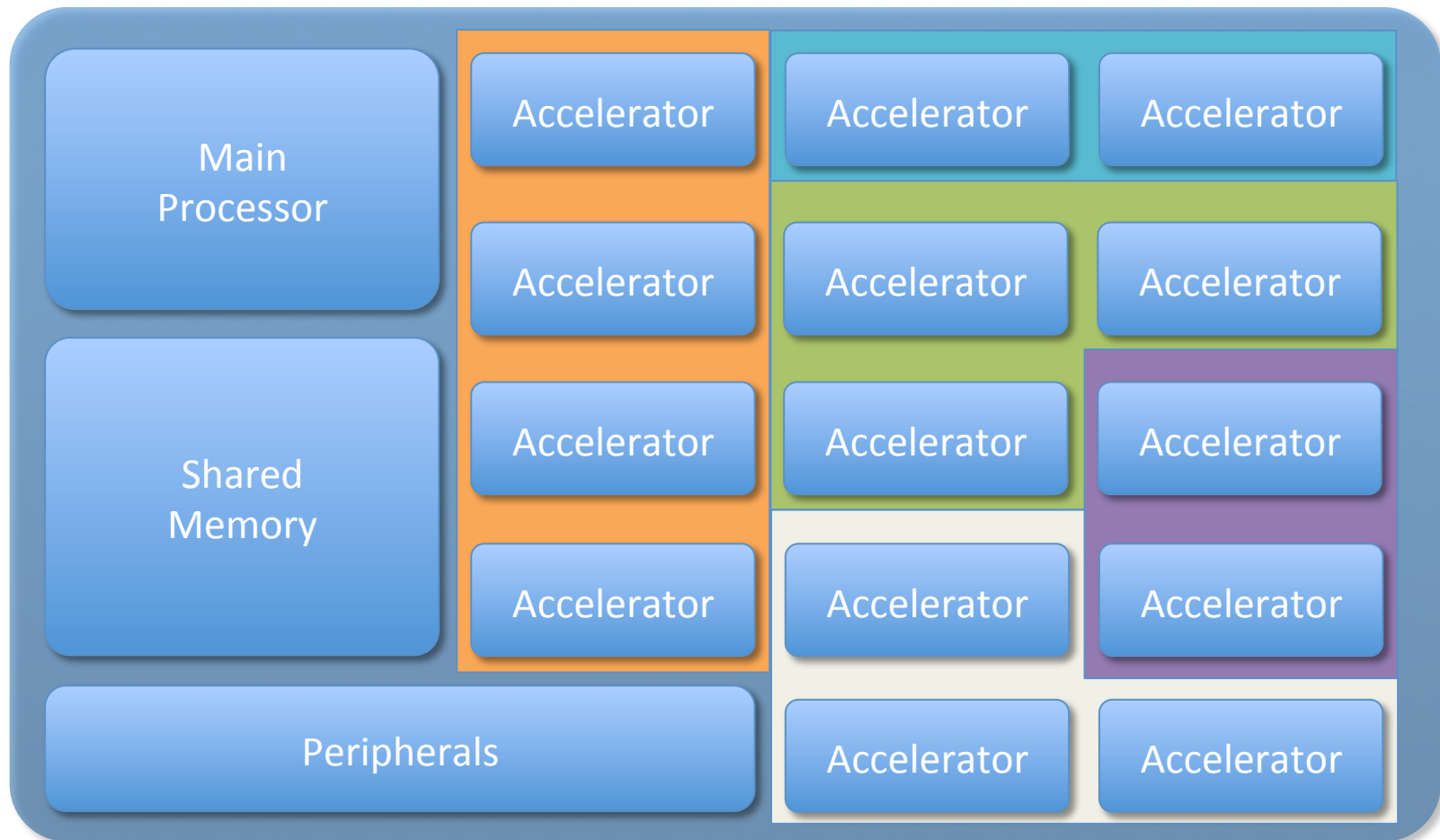


Composable Accelerators Provide Application Flexibility

Challenges in Accelerators

- Flexibility
 - Fixed-function accelerators are only designed for the target applications.
- Design Cost
 - Hand-written RTL implementation is inherently tedious and time-consuming.

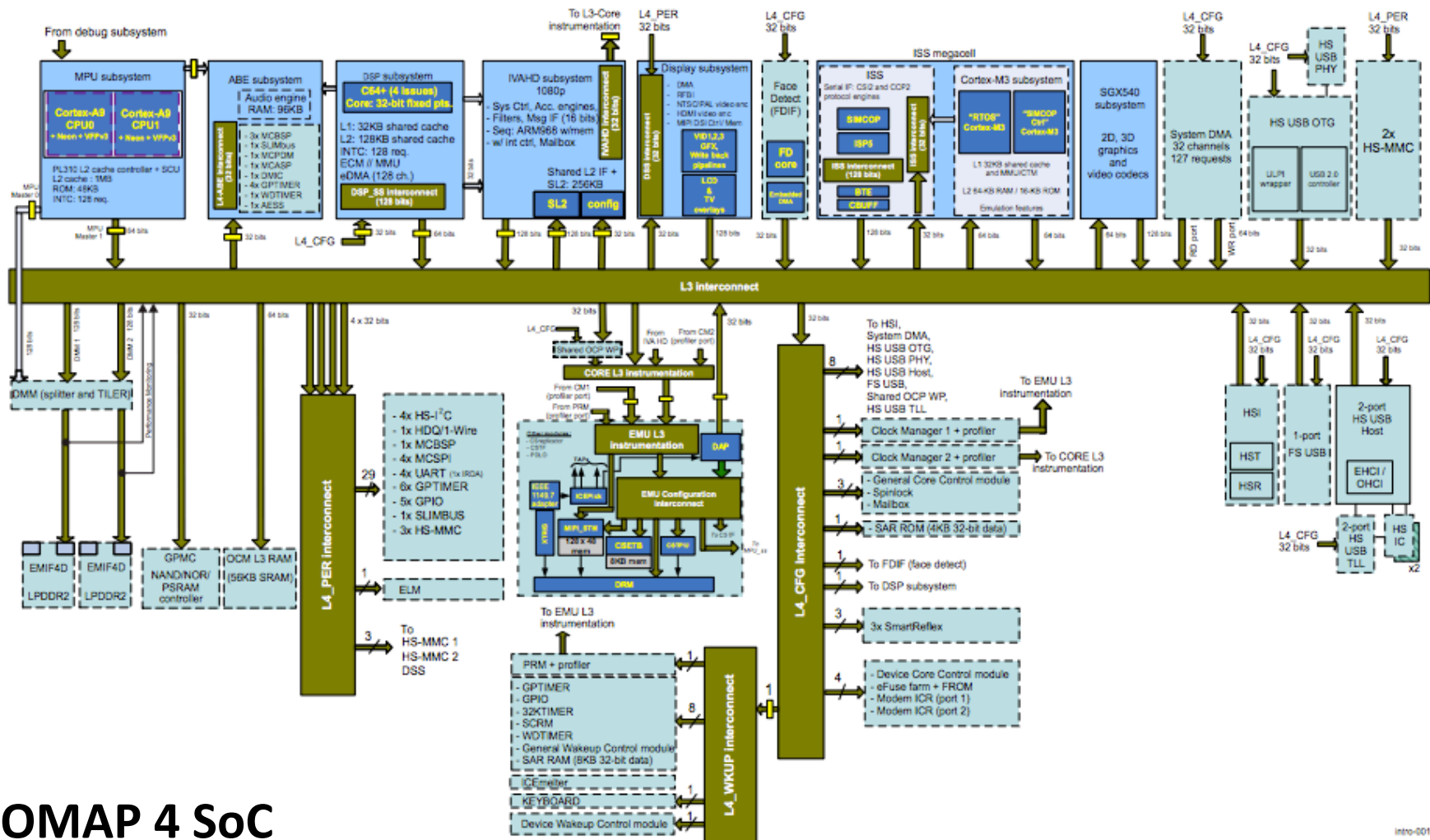
Flexibility requires a large number of accelerators.



Challenges in Accelerators

- Flexibility
 - Fixed-function accelerators are only designed for the target applications.
- Design Cost
 - Hand-written RTL implementation is inherently tedious and time-consuming.
- Programmability
 - Today's accelerators are explicitly managed by programmers.

Today's SoC



OMAP 4 SoC



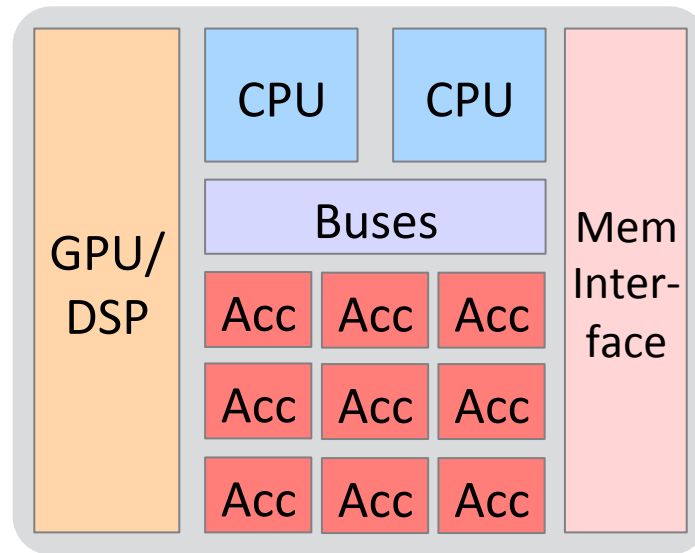
HARVARD
UNIVERSITY



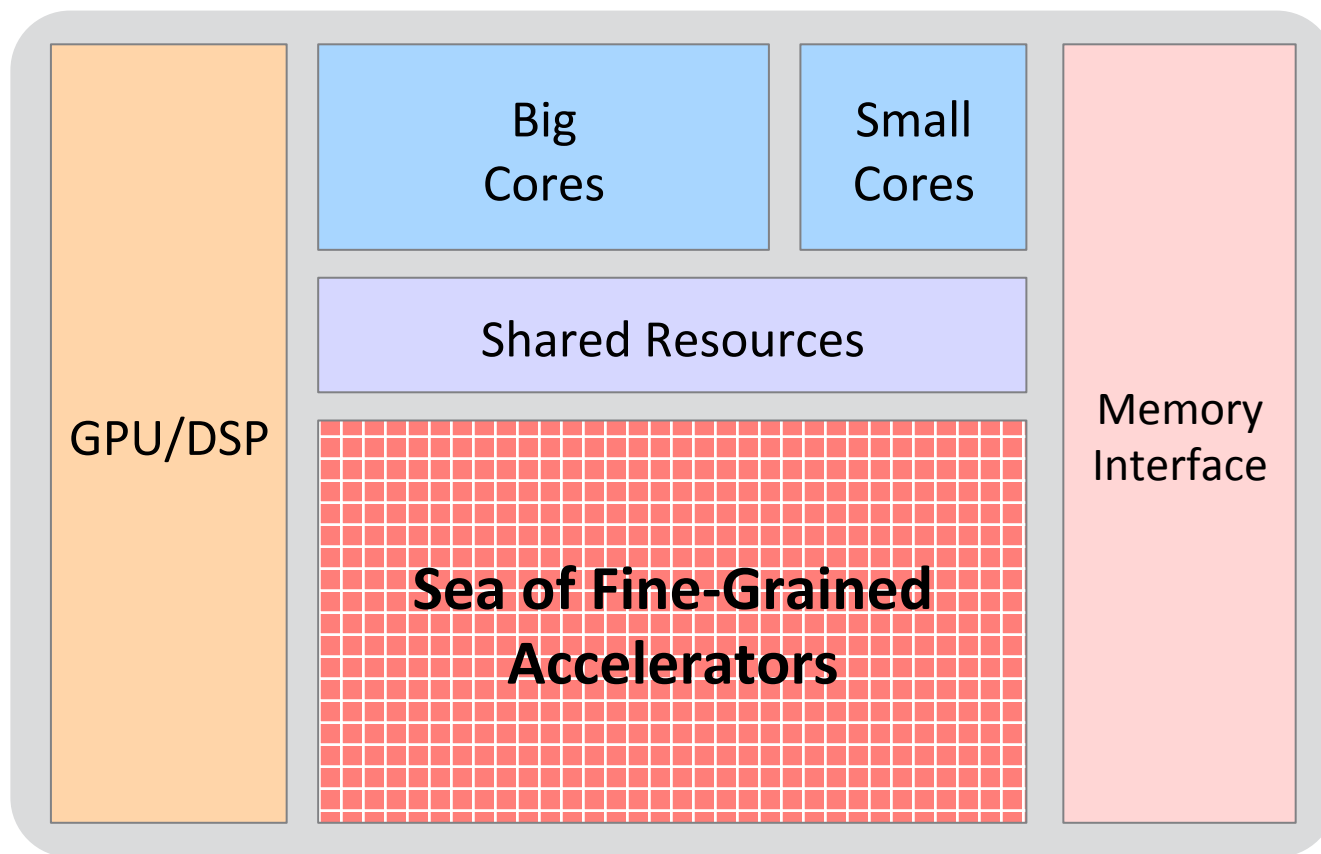
UNIVERSITY
of VIRGINIA

OMAP 4 SoC

Today's SoC



Future Accelerator-Centric Architectures



How to decompose an application to accelerators?

How to rapidly design lots of accelerators?

How to design and manage the shared resources?

↑ Flexibility

↓ Design Cost

↑ Programmability

Tutorial Outline

Time	Topic
8:30 am – 9:00 am	Introduction
9:00 am – 10:00 am	Pre-RTL Simulation Framework: Aladdin
10:00 am – 10:30 am	Break
10:30 am – 11:30 am	CAD & Benchmarks: HLS & MachSuite
11:30 am – 12:00 pm	Aladdin Validation and Case Studies
12:00 pm – 1:30 pm	Lunch
1:30 pm – 2:15 pm	Analytical Modeling: Lumos
2:15 pm – 2:45 pm	Workload Characterization Tool: WIICA
2:45 pm – 3:00 pm	Hands-on Exercise: Set up
3:00 pm – 3:30 pm	Break
3:30 pm – 5:00 pm	Hands-on Exercise



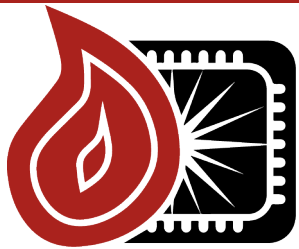


A Pre-RTL, Power-Performance Accelerator Simulator

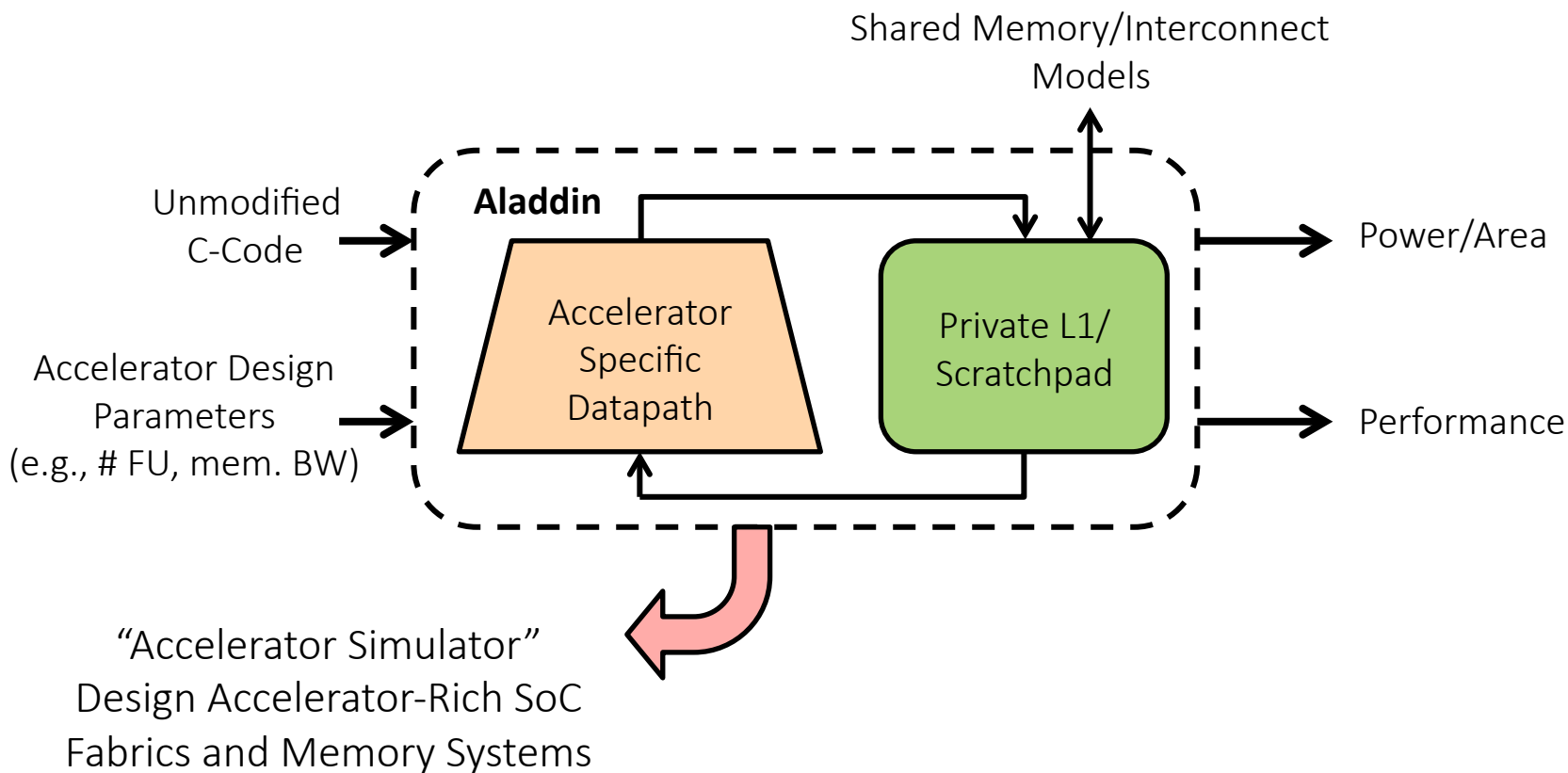
Enabling Large Design Space Exploration of Customized Architectures

Yakun Sophia Shao, Brandon Reagen,
Gu-Yeon Wei, David Brooks
Harvard University

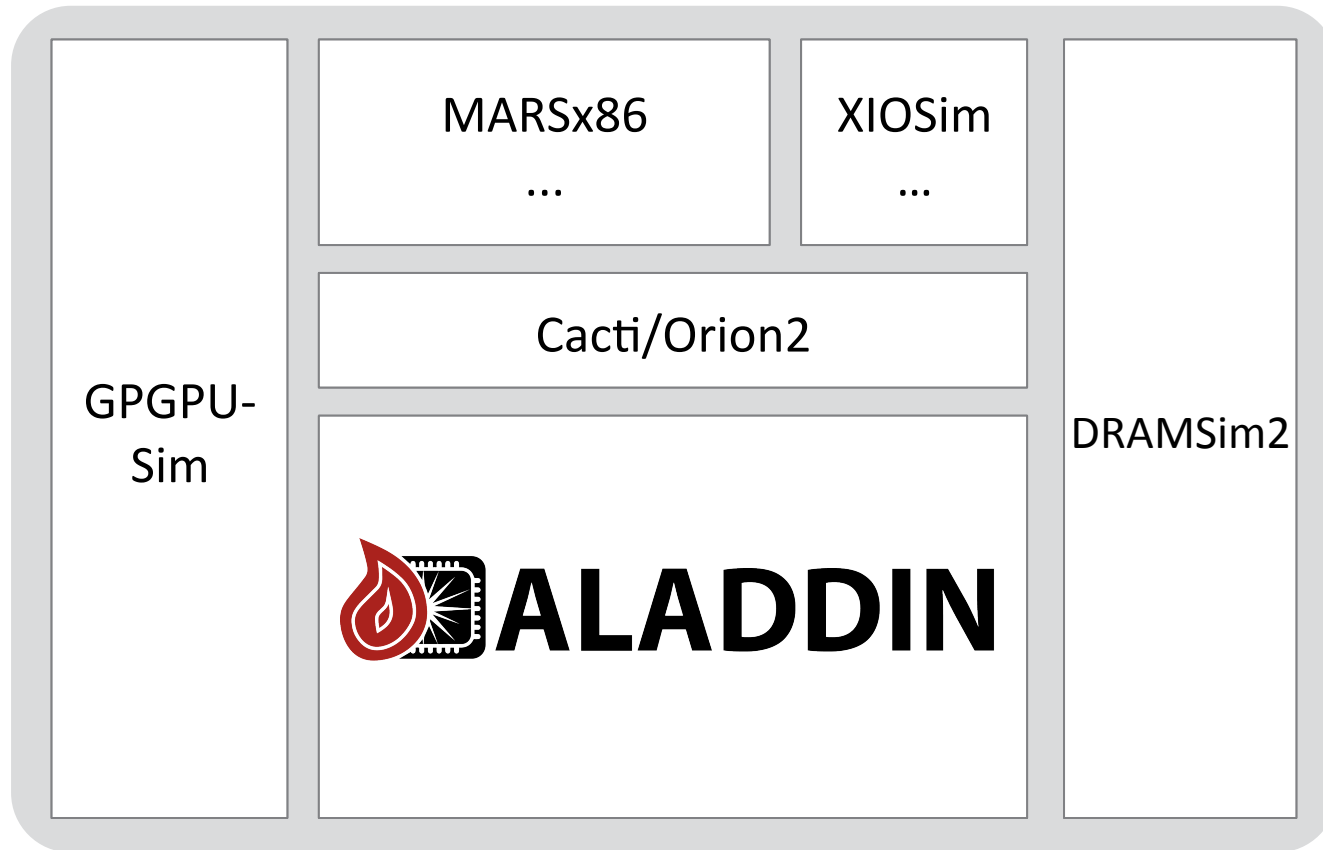


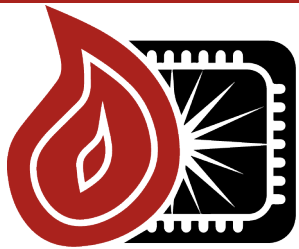


Aladdin: A pre-RTL, Power-Performance Accelerator Simulator

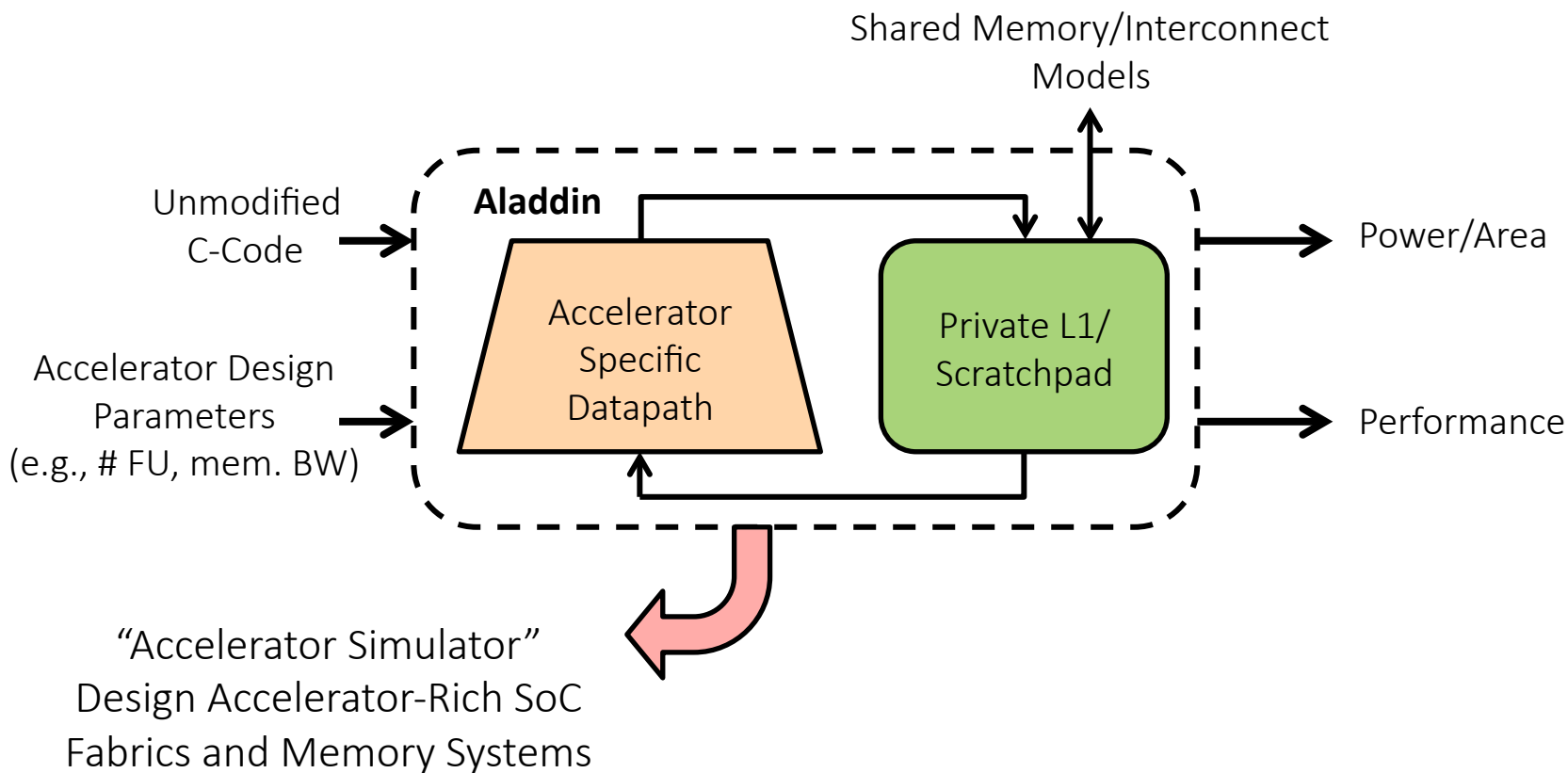


Aladdin enables pre-RTL simulation of accelerators with the rest of the SoC.



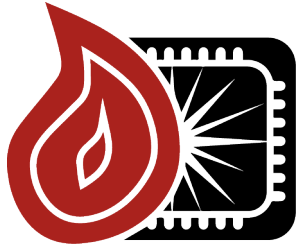


Aladdin: A pre-RTL, Power-Performance Accelerator Simulator

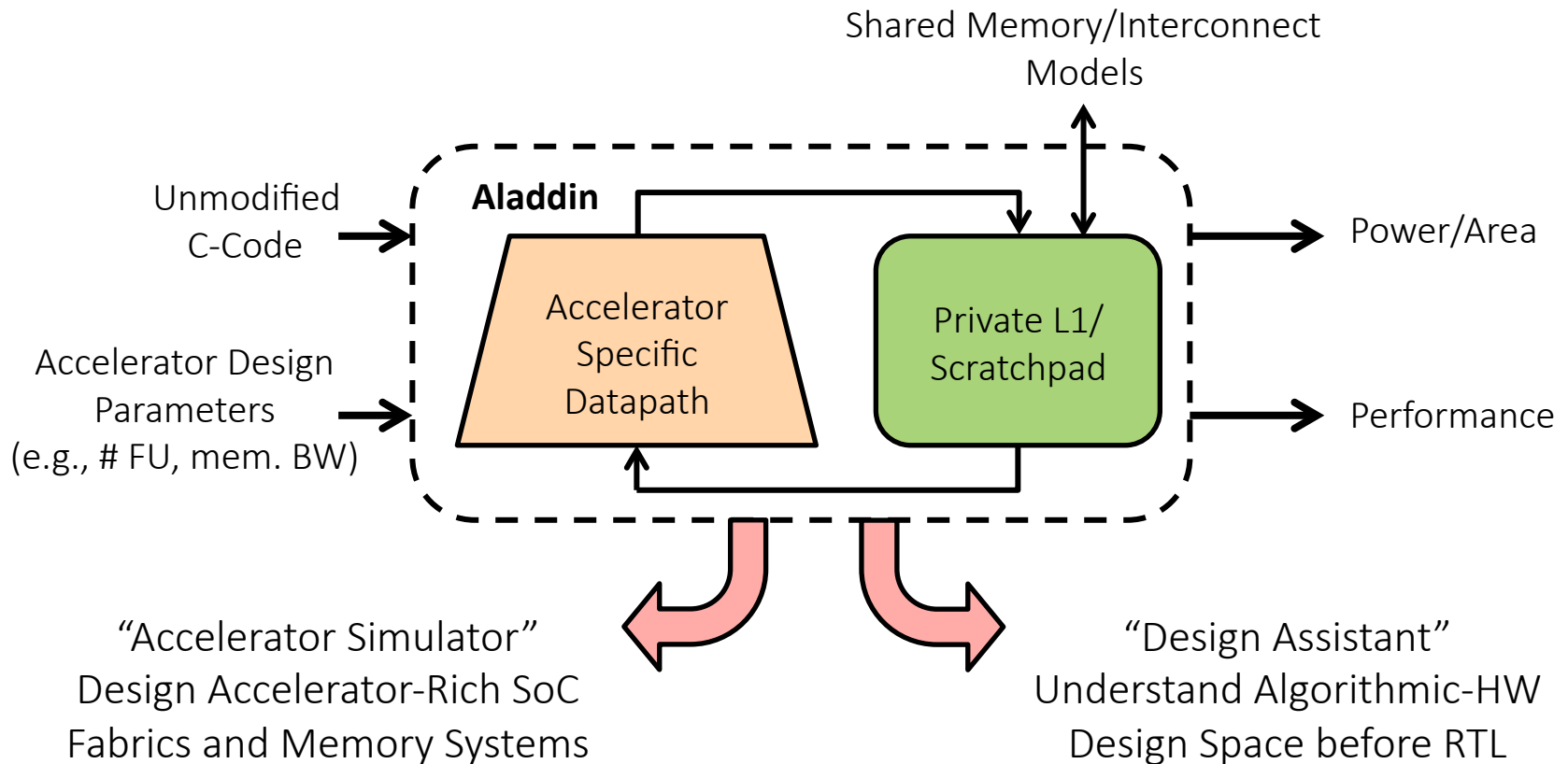


↑ **Flexibility**

↑ **Programmability**



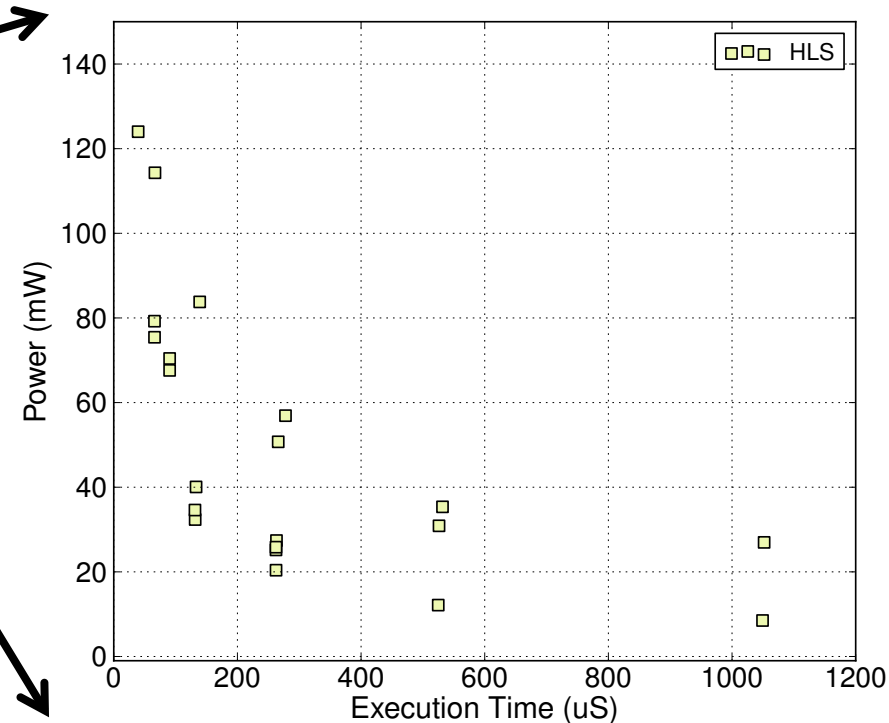
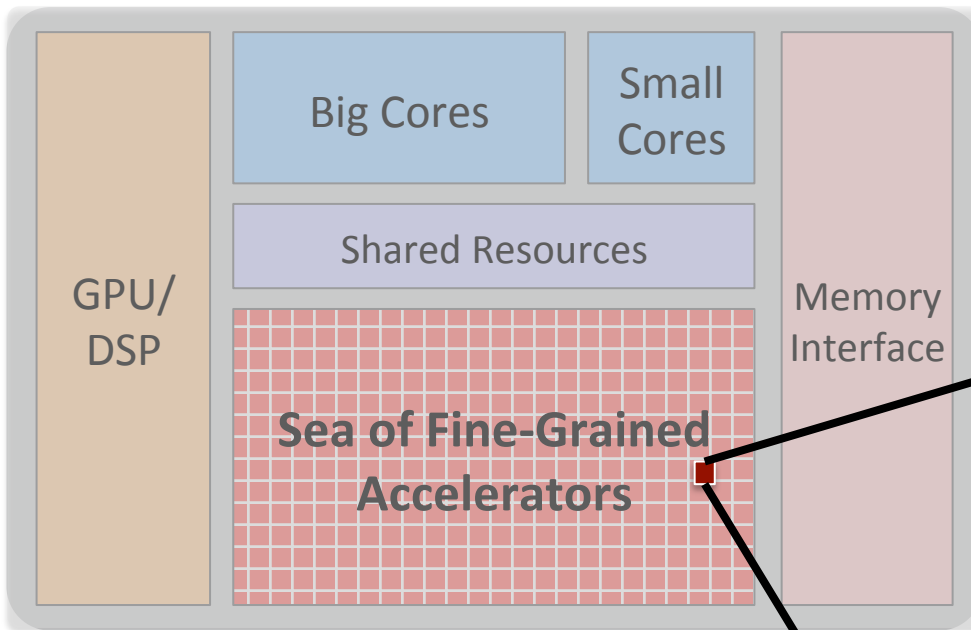
Aladdin: A pre-RTL, Power-Performance Accelerator Simulator



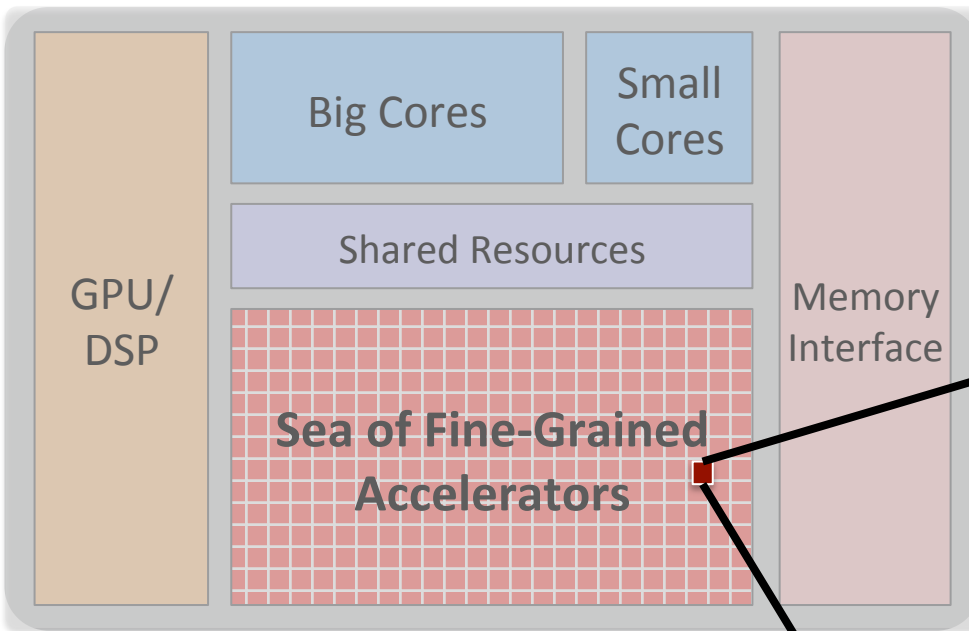
↑ **Flexibility**
↑ **Programmability**

↓ **Design Cost**

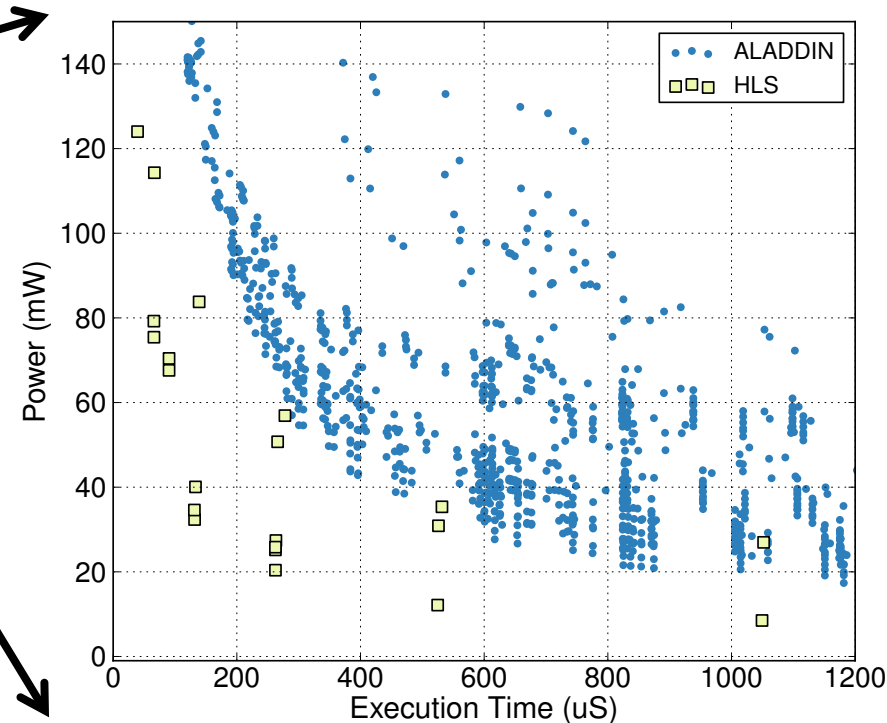
Future Accelerator-Centric Architecture



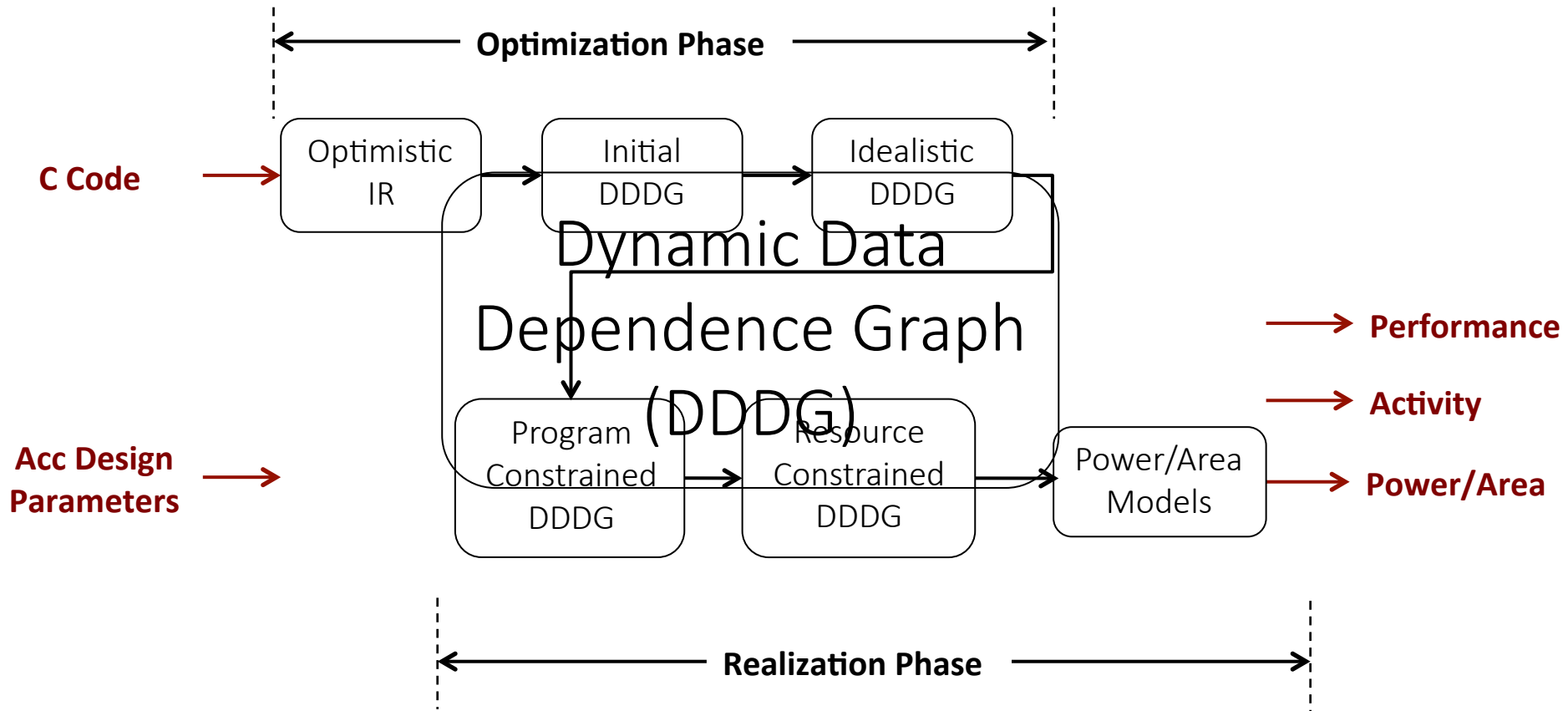
Future Accelerator-Centric Architecture



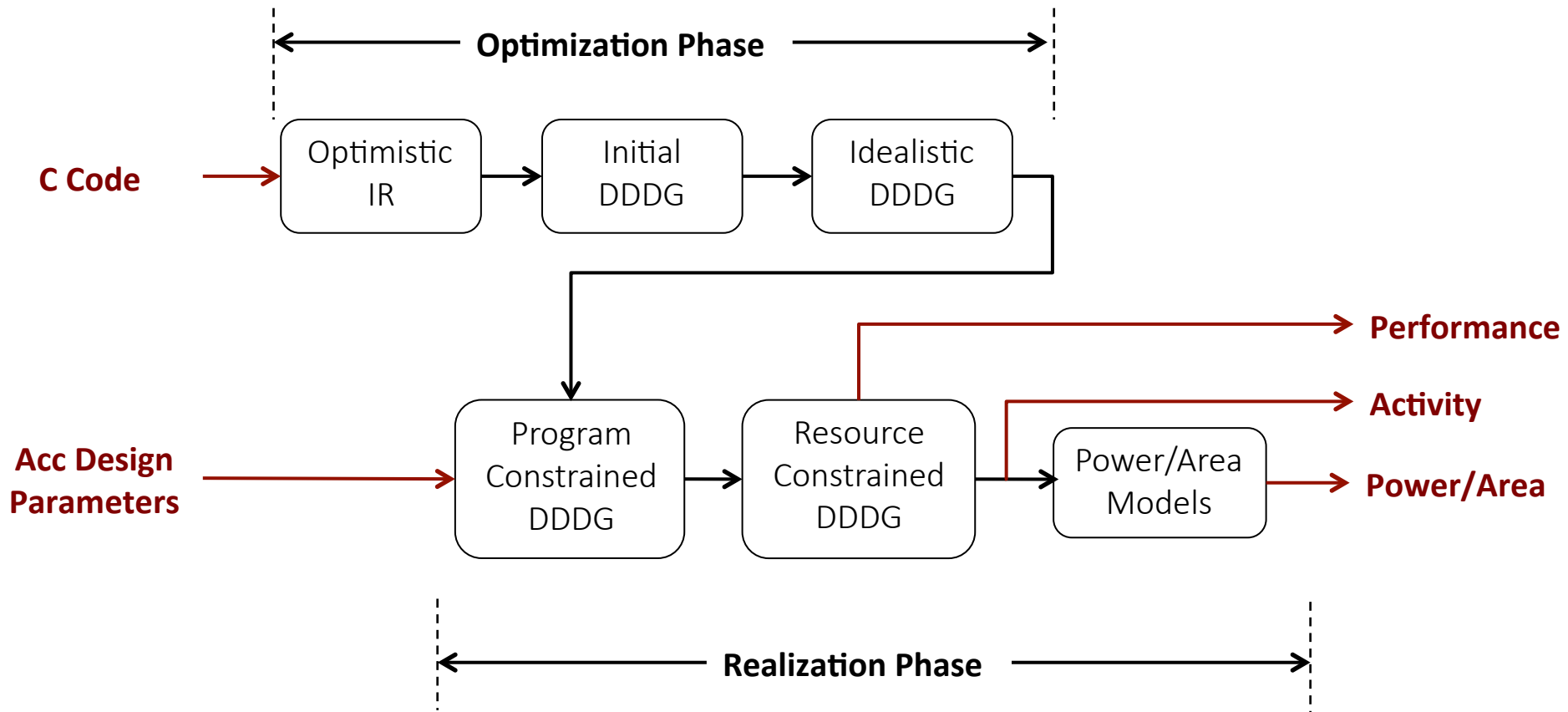
Aladdin can **rapidly** evaluate **large** design space of accelerator-centric architectures.



Aladdin Overview



Aladdin Overview



Aladdin is NOT

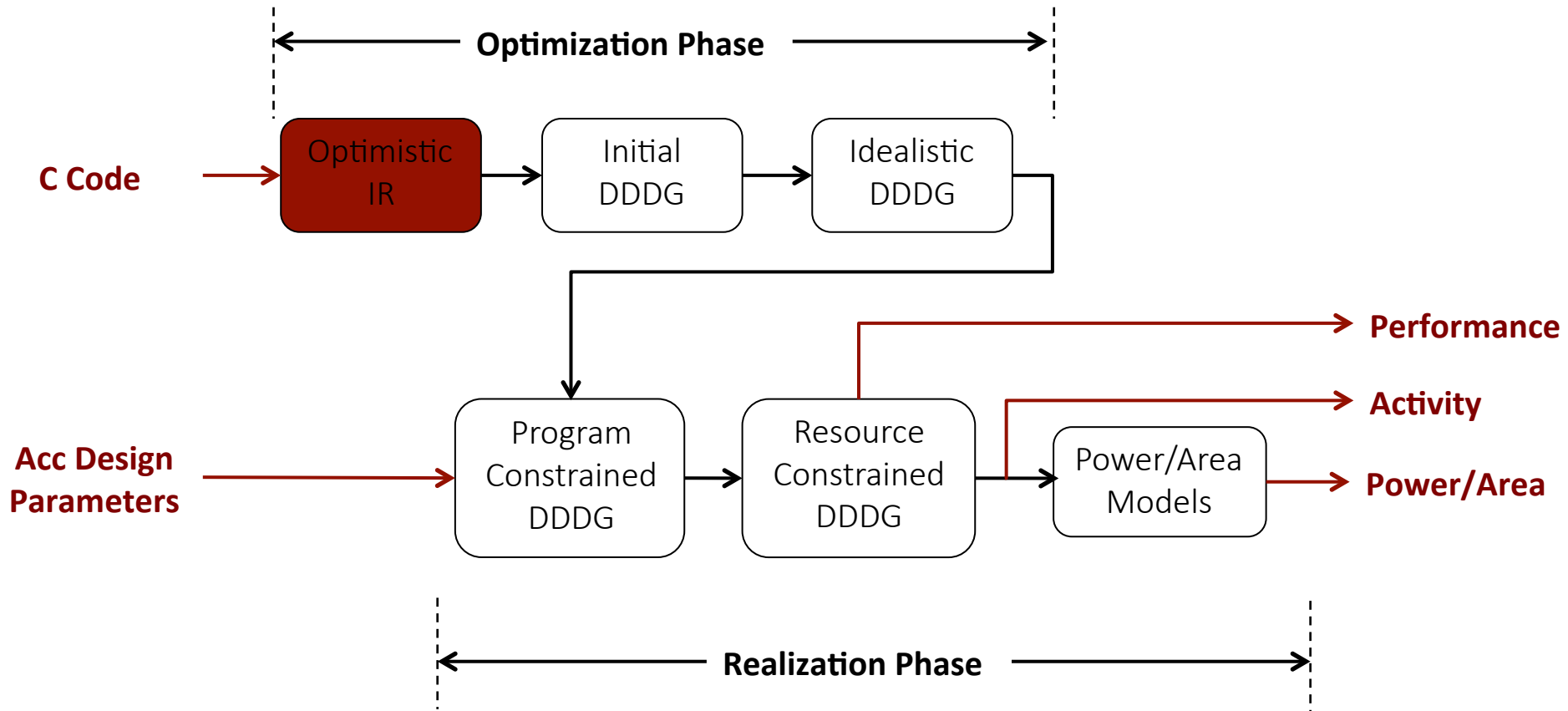
- An HLS flow:
 - No RTL is generated.
 - High-level *estimates* of power and performance;
 - Aladdin uses fully dynamic analysis to expose algorithmic parallelism for unmodified HLL codes;
- Limit of ILP study:
 - “optimistic but realistic” DDDG is constructed to model accelerators.

From C to Design Space

C Code:

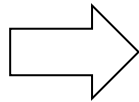
```
for(i=0; i<N; ++i)  
    c[i] = a[i] + b[i];
```

Aladdin Overview



IR Dynamic Trace

C Code:
for(i=0; i<N; ++i)
 c[i] = a[i] + b[i];



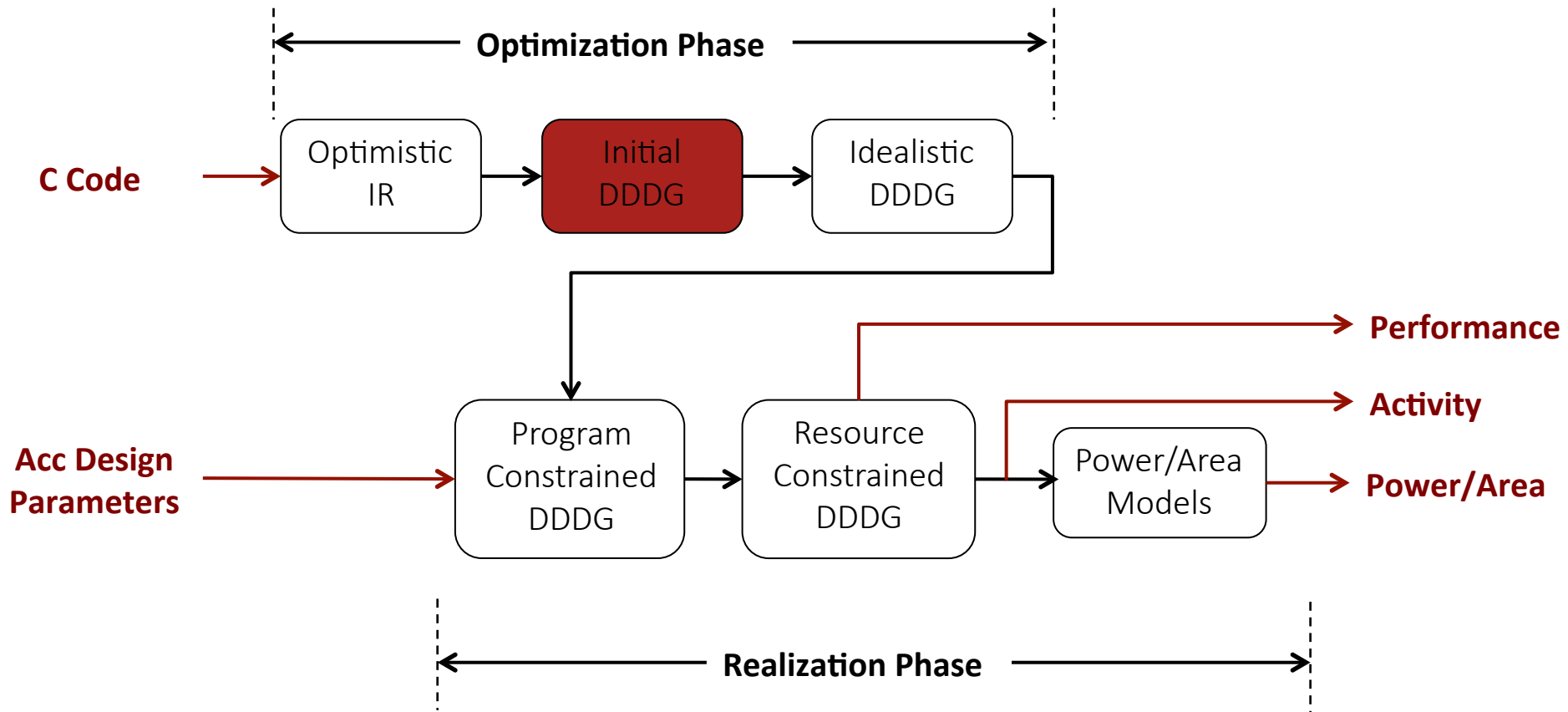
0. r0=0 //i = 0
1. r4=load (r0 + r1) //load a[i]
2. r5=load (r0 + r2) //load b[i]
3. r6=r4 + r5
4. store(r0 + r3, r6) //store c[i]
5. r0=r0 + 1 //++i
6. r4=load(r0 + r1) //load a[i]
7. r5=load(r0 + r2) //load b[i]
8. r6=r4 + r5
9. store(r0 + r3, r6) //store c[i]
10. r0 = r0 + 1 //++i
...

Optimistic IR

- ILDJIT
- High-level IR:
 - machine-, ISA-, and system-library-independent
- Features:
 - Unlimited Registers
 - 80 Opcodes: add, mul, sin, sqrt
 - Only load/store access memory

Shao, et al., ISA-Independent Workload Characterization and Implications for Specialized Architecture, ISPASS, 2013

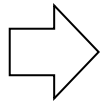
Aladdin Overview



From C to Design Space

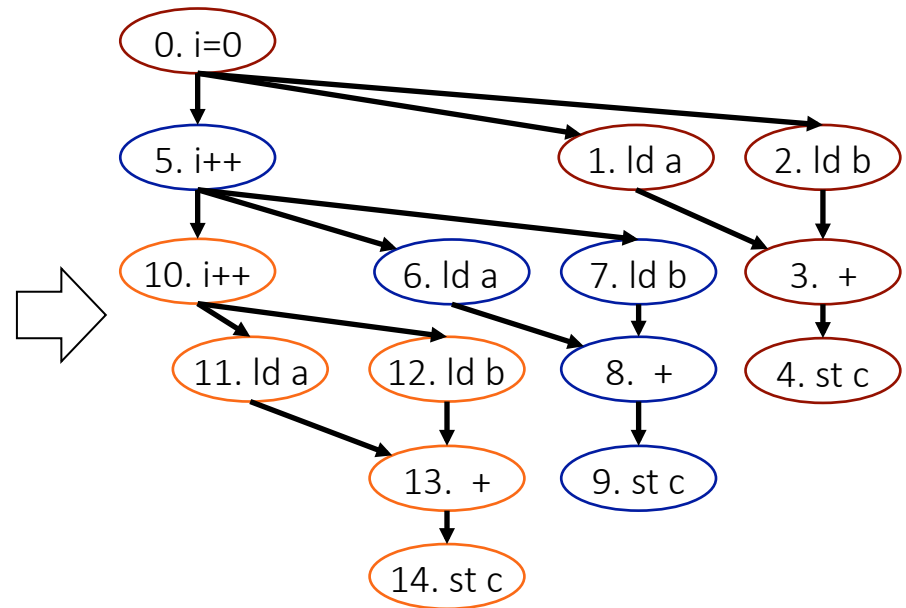
Initial DDDG

C Code:
for(i=0; i<N; ++i)
 c[i] = a[i] + b[i];

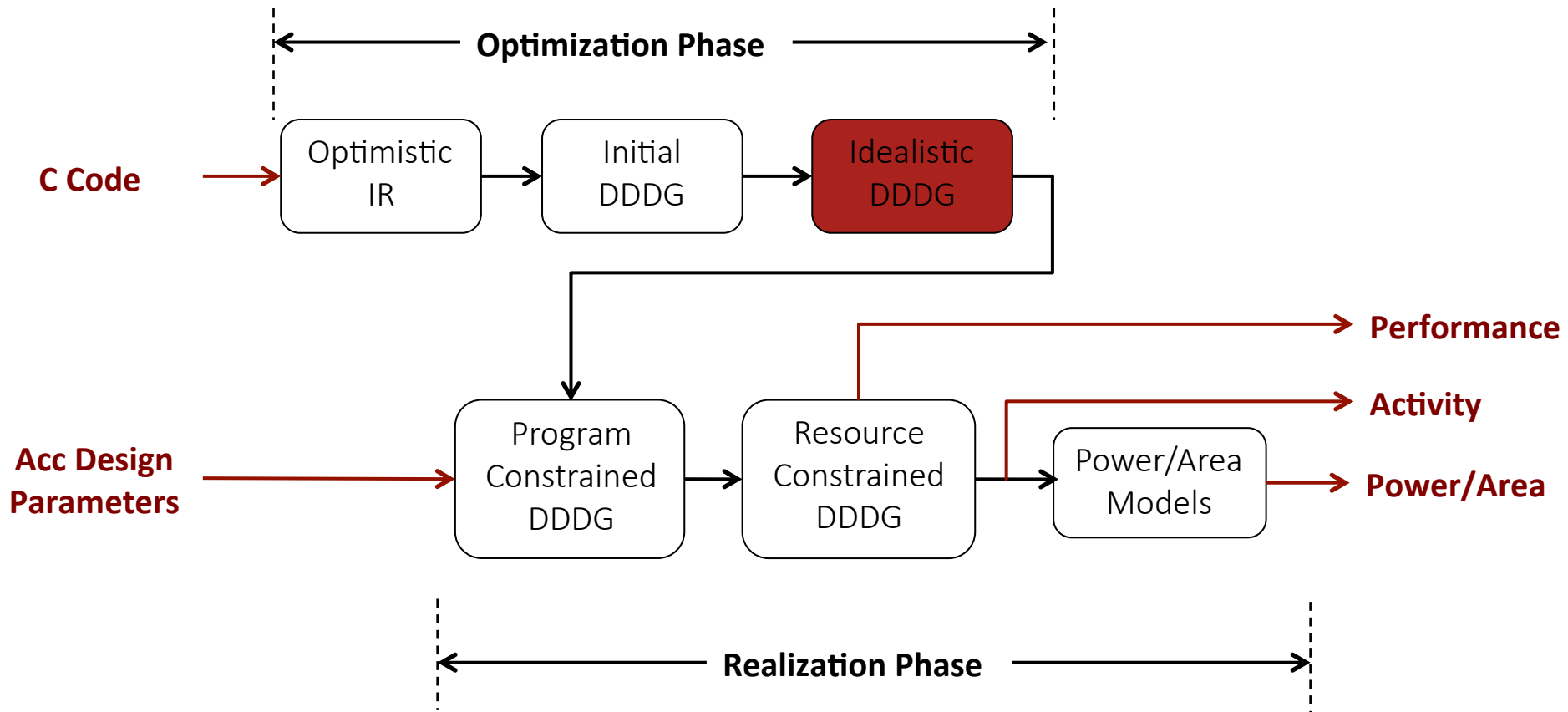


IR Trace:

```
0. r0=0 //i = 0
1. r4=load (r0 + r1) //load a[i]
2. r5=load (r0 + r2) //load b[i]
3. r6=r4 + r5
4. store(r0 + r3, r6) //store c[i]
5. r0=r0 + 1 //++i
6. r4=load(r0 + r1) //load a[i]
7. r5=load(r0 + r2) //load b[i]
8. r6=r4 + r5
9. store(r0 + r3, r6) //store c[i]
10.r0 = r0 + 1 //++i
...
```

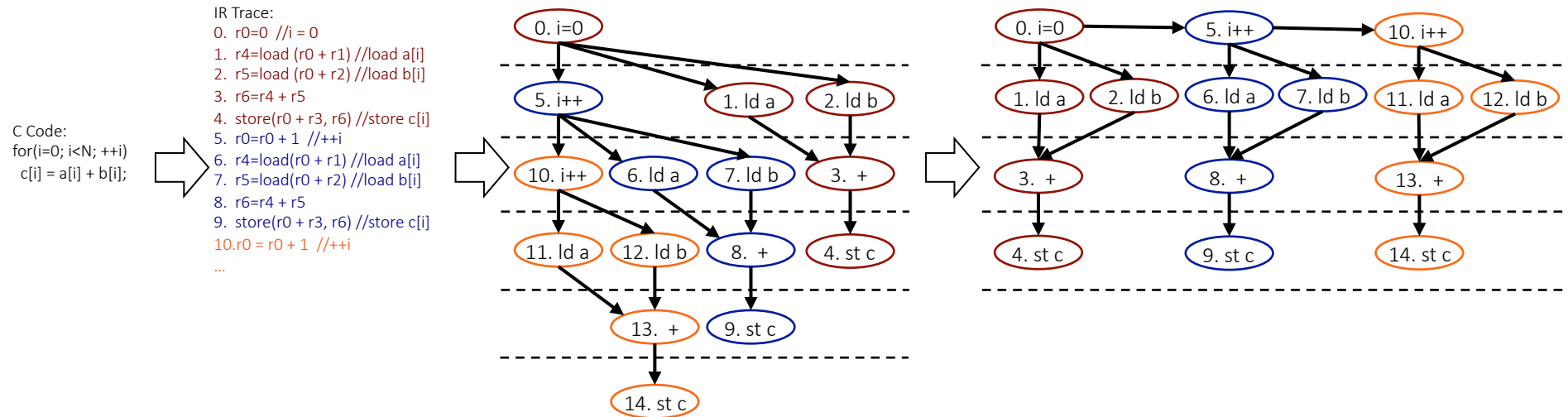


Aladdin Overview



From C to Design Space

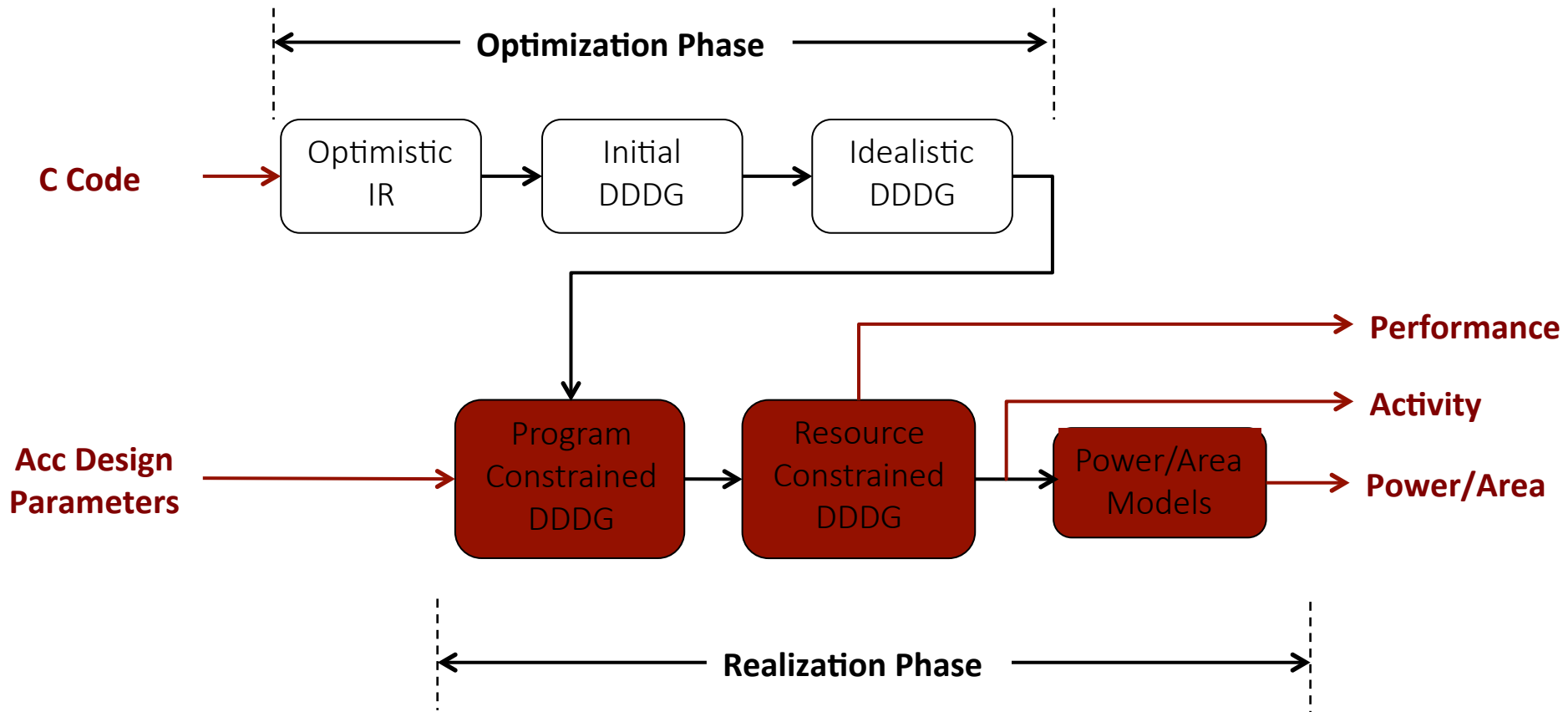
Idealistic DDDG



Idealistic DDDG

- Include application-specific customization strategies.
- Node-Level:
 - Bit-width Analysis
 - Strength Reduction
 - Tree-height Reduction
- Loop-Level:
 - Remove dependences between loop index variables
- Memory Optimization:
 - Memory-to-Register Conversion
 - Store-Load Forwarding
 - Store Buffer
- Extensible
 - e.g. Model CAM accelerator by matching nodes in DDDG

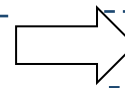
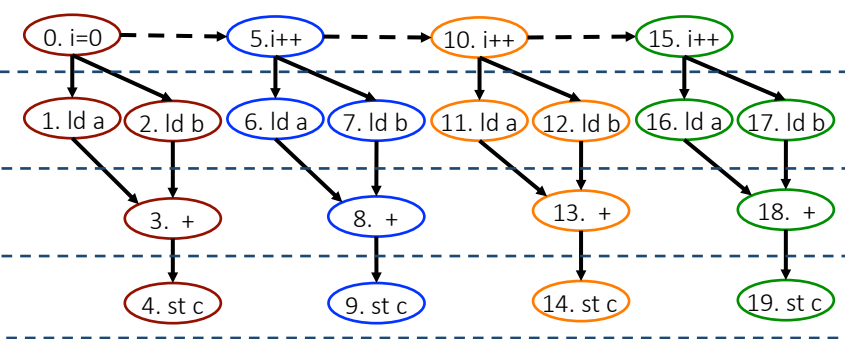
Aladdin Overview



From C to Design Space

One Design

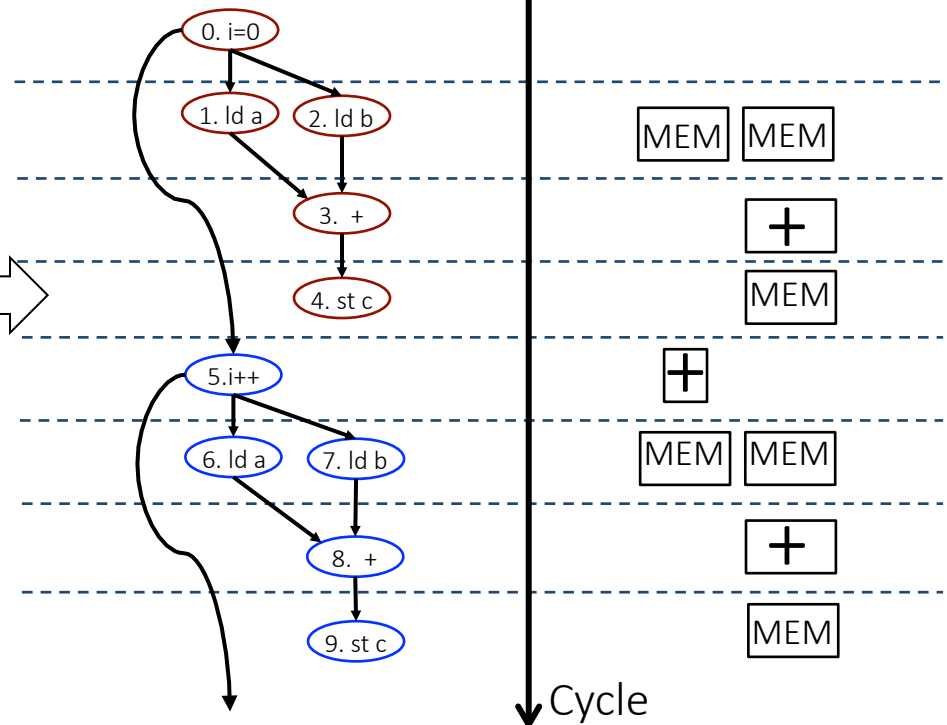
Idealistic DDDG



Acc Design Parameters:

- ✓ Memory BW ≤ 2
- ✓ **1** Adder

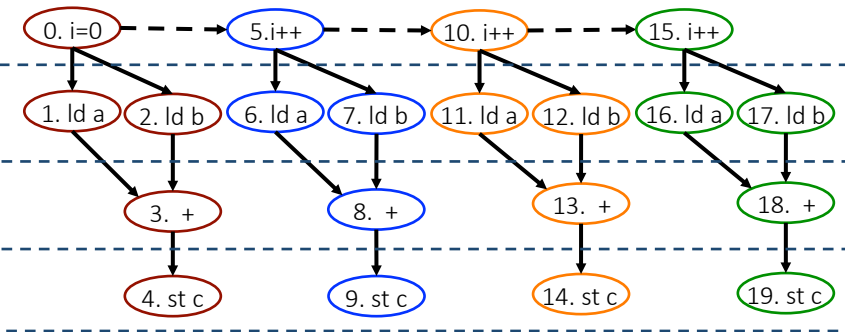
Resource Activity



From C to Design Space

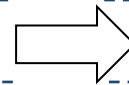
Another Design

Idealistic DDDG

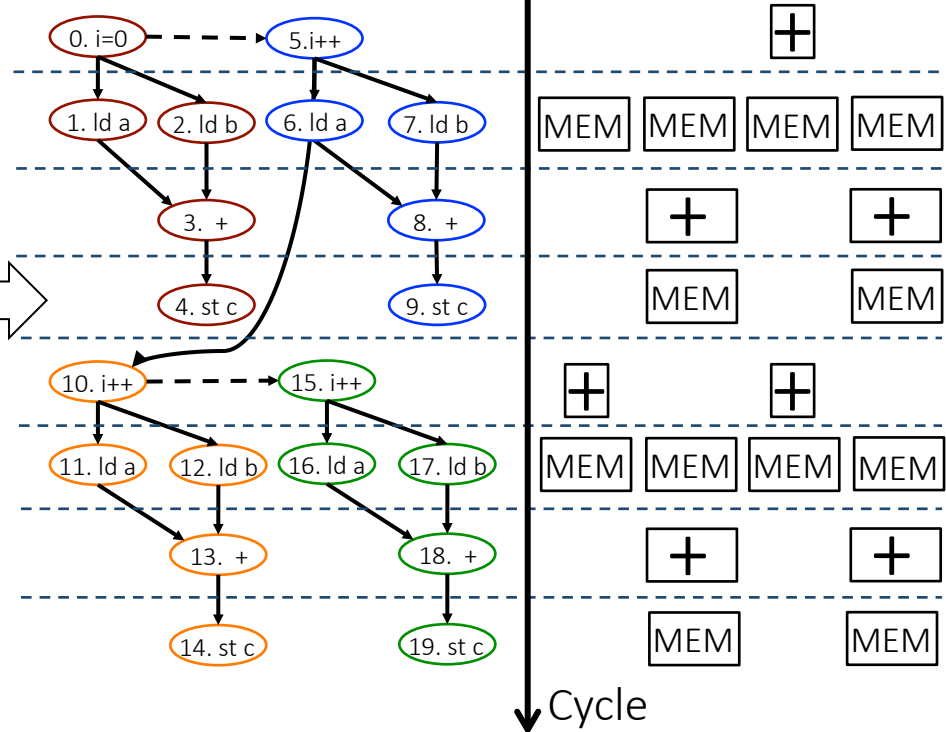


Acc Design Parameters:

- ✓ Memory BW ≤ 4
- ✓ **2** Adders

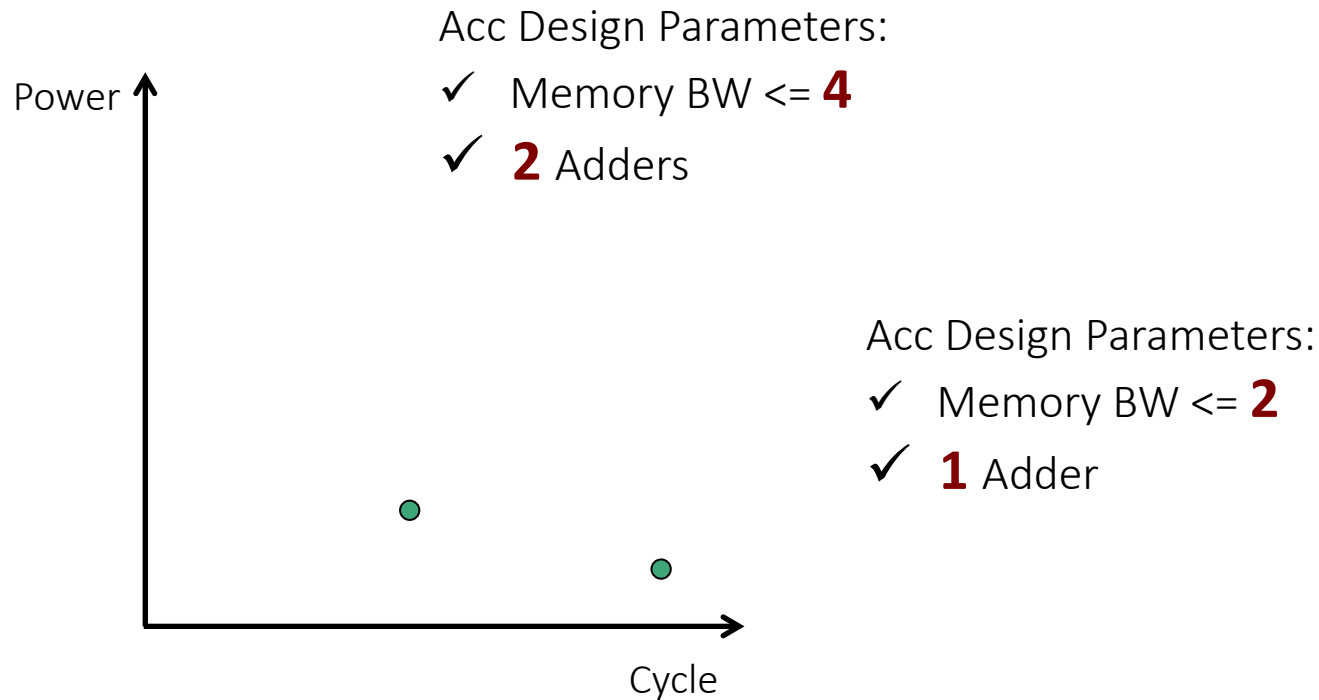


Resource Activity



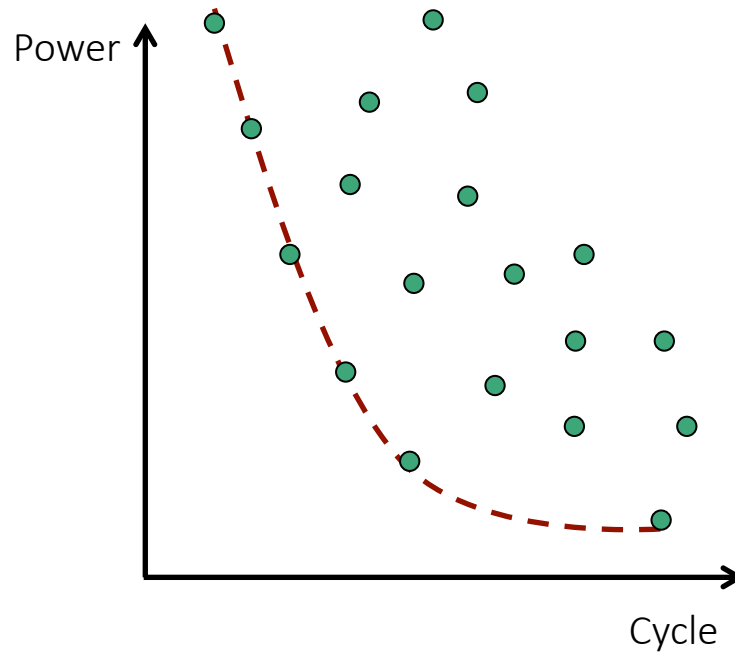
From C to Design Space

Power-Performance per Design

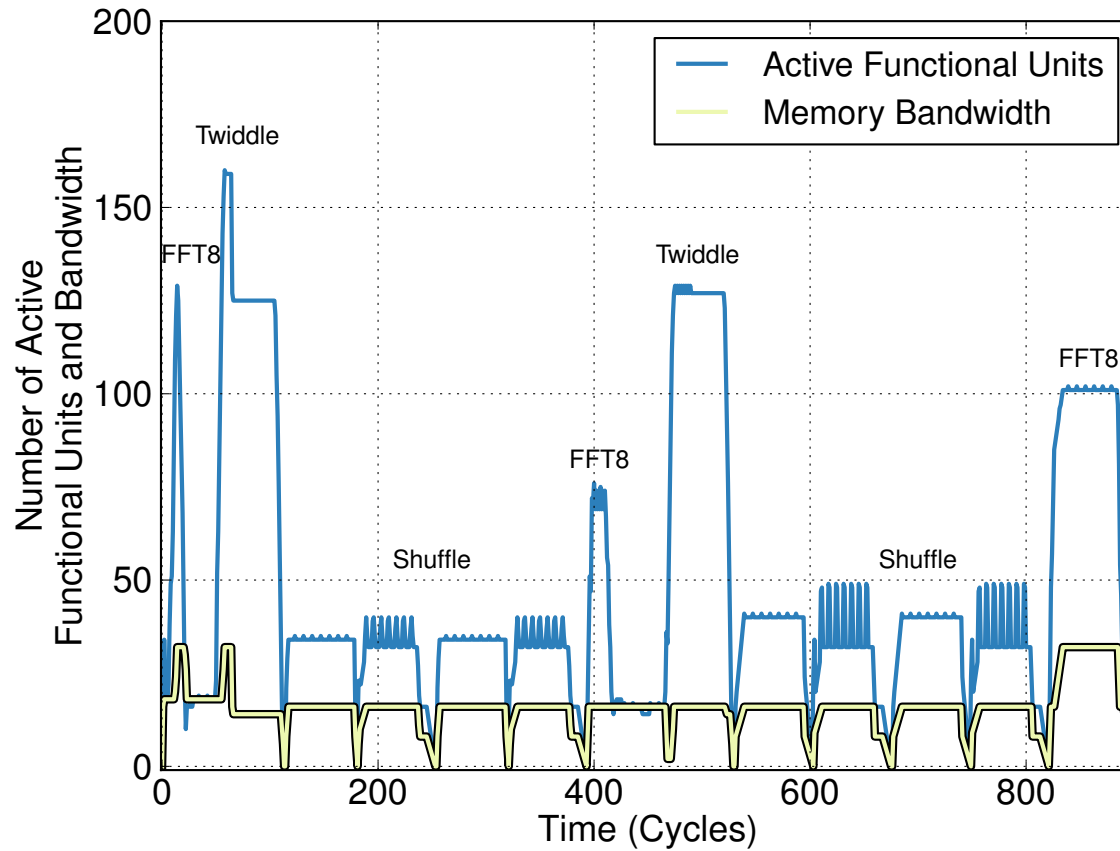


From C to Design Space

Design Space of an Algorithm



Cycle-Level Activity



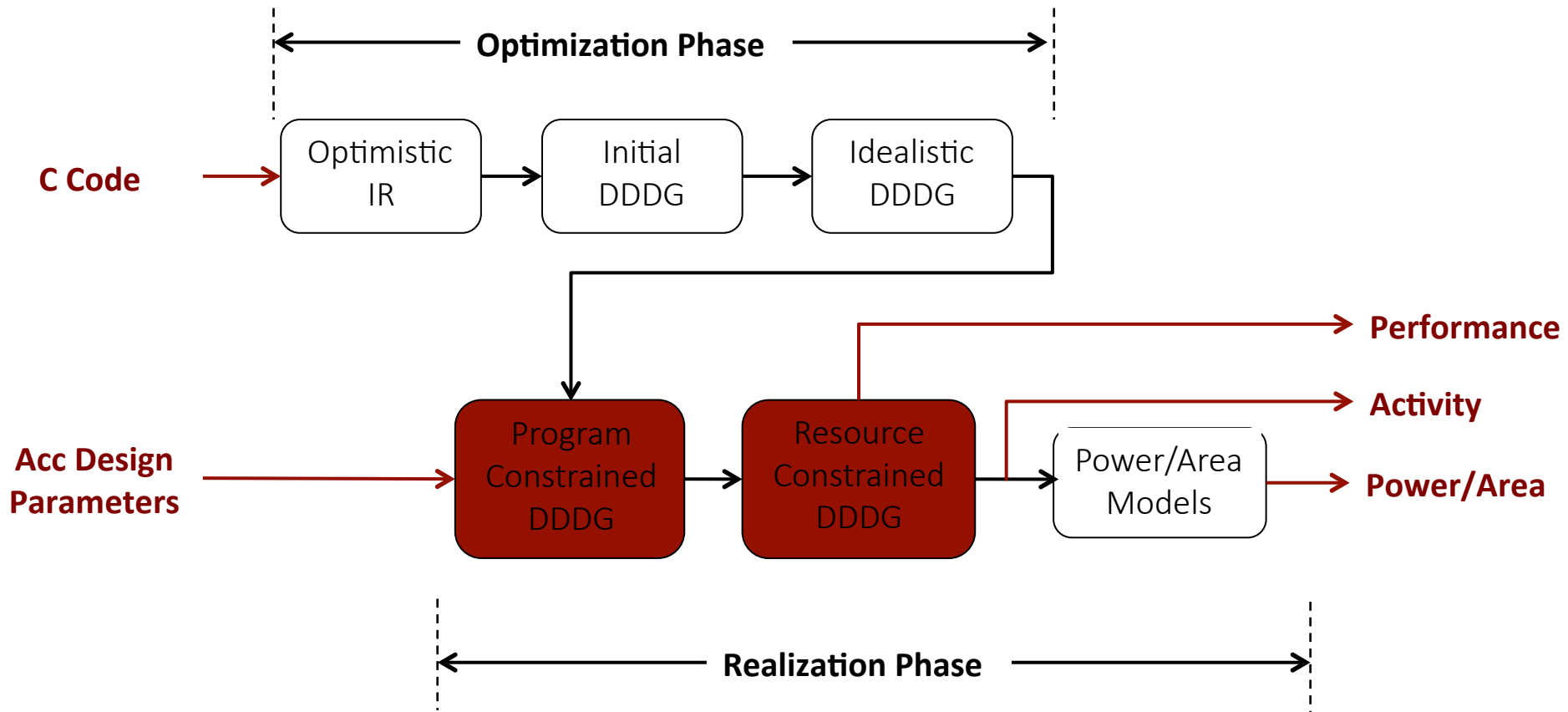
Power Model

- Functional Units Power Model
 - Microbenchmarks characterize various FUs.
 - Design Compiler with 40nm Standard Cell
 - FreePDK 45nm

$$Power = \sum_{1 \leq i \leq N} (activity_i * P_{i_{switching}}) + P_{i_{internal}} + P_{i_{leakage}}$$

- SRAM Power Model
 - Commercial register file and SRAM memory compilers with the same 40nm standard cell library
 - CACTI

Aladdin Overview



Realization Phase: DDDG->Power-Perf

- Constrain the DDDG with program and user-defined resource constraints
- Program Constraints
 - Control Dependence
 - Memory Ambiguation
- Resource Constraints
 - Loop-level Parallelism
 - Loop Pipelining
 - Memory Ports
 - # of FUs (e.g., adders, multipliers)

Control Dependence

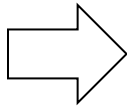
- Dynamic trace only has taken paths.
- Accelerators execute both taken and not taken paths until branch resolution.
- Aladdin brings code from the not-taken path in to the DDDG to account for additional power and resource requirements.

Memory Ambiguation

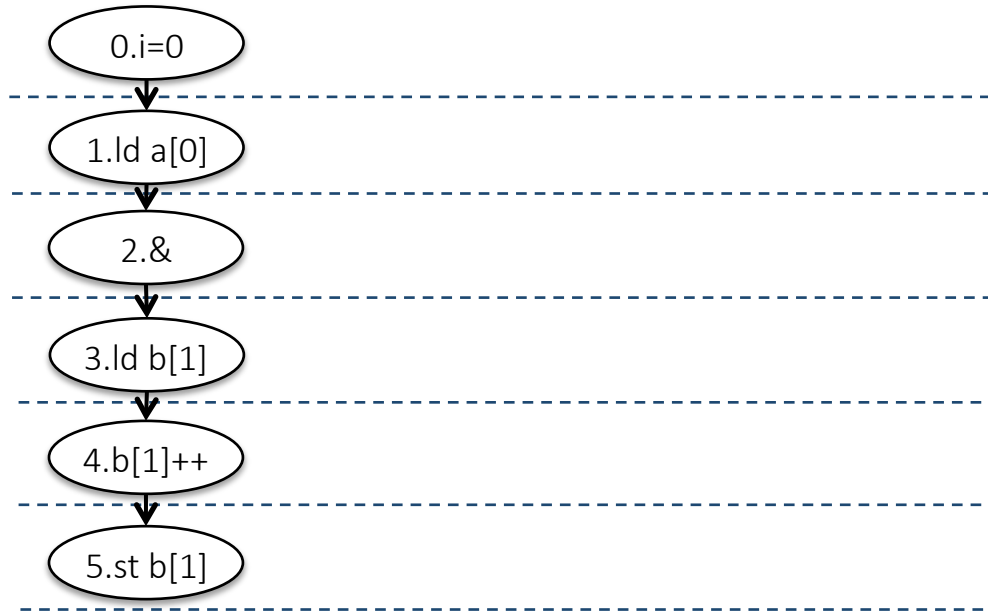
- Idealistic DDDG optimistically removes *all* false memory dependences
- Input-dependent memory accesses cannot be calculated statically.

Memory Ambiguation

```
for(i=0; i<N; ++i)
{
    bucket[ a[i] & 0x11 ]++;
}
```

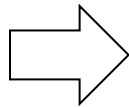


Input:
a[0] = 1;

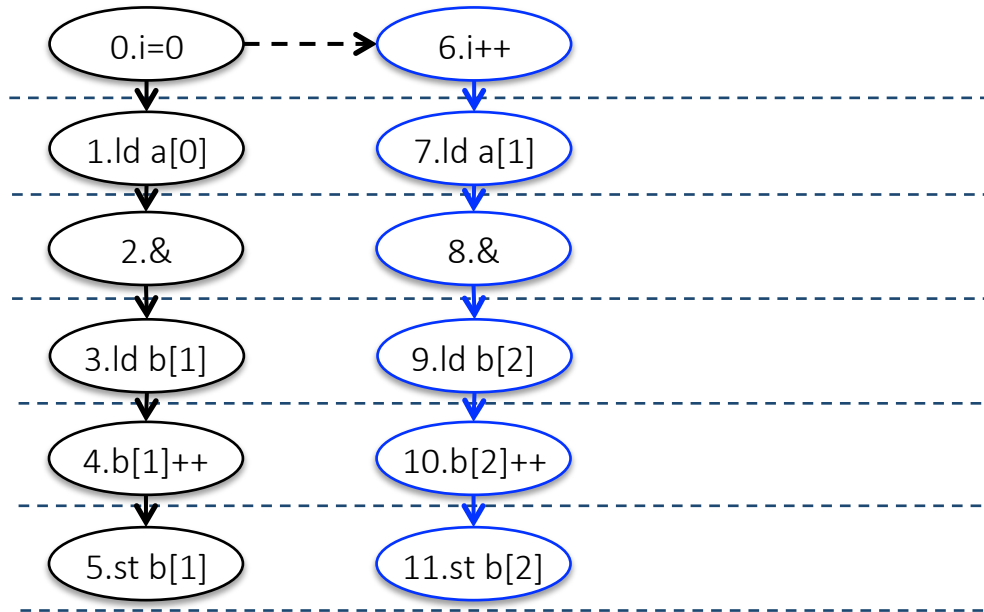


Memory Ambiguation

```
for(i=0; i<N; ++i)
{
    bucket[ a[i] & 0x11 ]++;
}
```

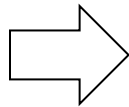


Input:
a[0] = 1;
a[1] = 2;



Memory Ambiguation

```
for(i=0; i<N; ++i)
{
    bucket[ a[i] & 0x11 ]++;
}
```



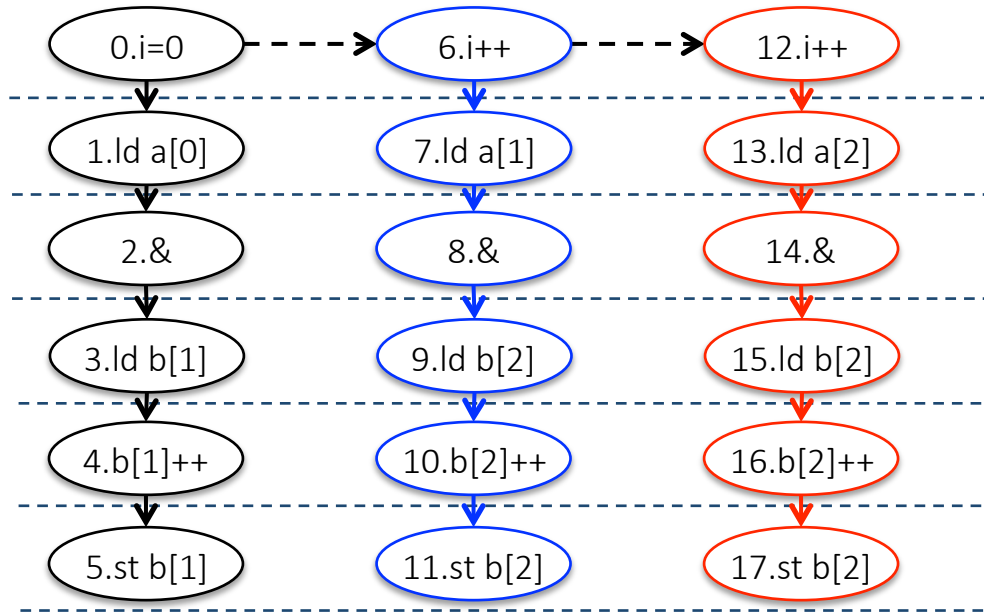
Input:

a[0] = 1;

a[1] = 2;

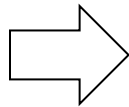
a[2] = 2;

...



Memory Ambiguation

```
for(i=0; i<N; ++i)
{
    bucket[ a[i] & 0x11 ]++;
}
```



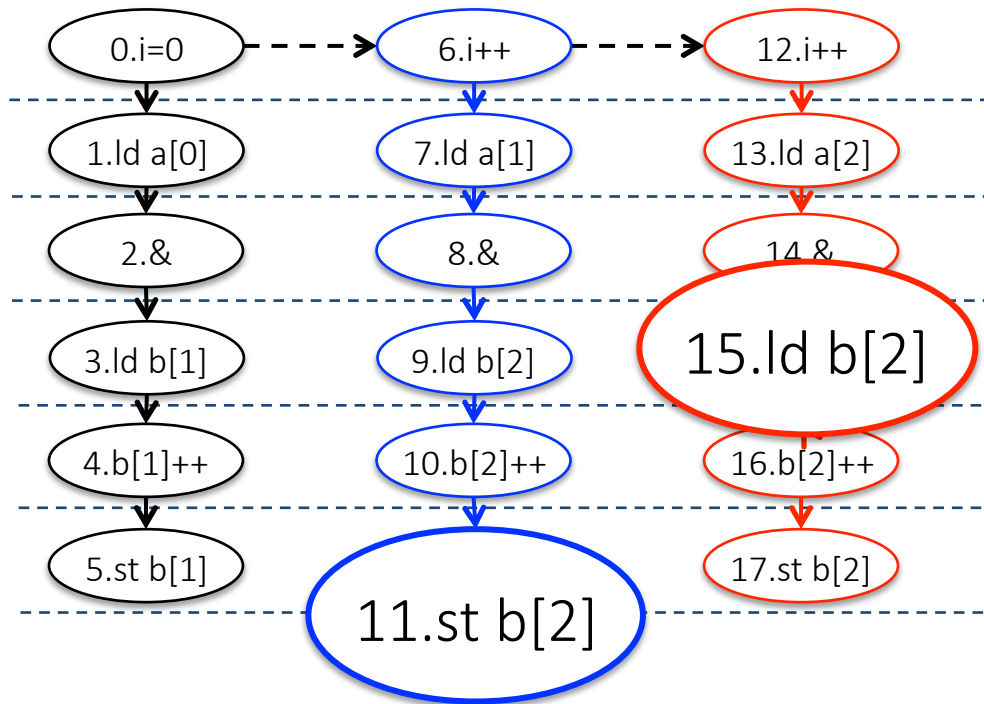
Input:

a[0] = 1;

a[1] = 2;

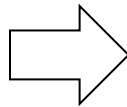
a[2] = 2;

...



Memory Ambiguation

```
for(i=0; i<N; ++i)
{
  bucket[ a[i] & 0x11 ]++;
}
```



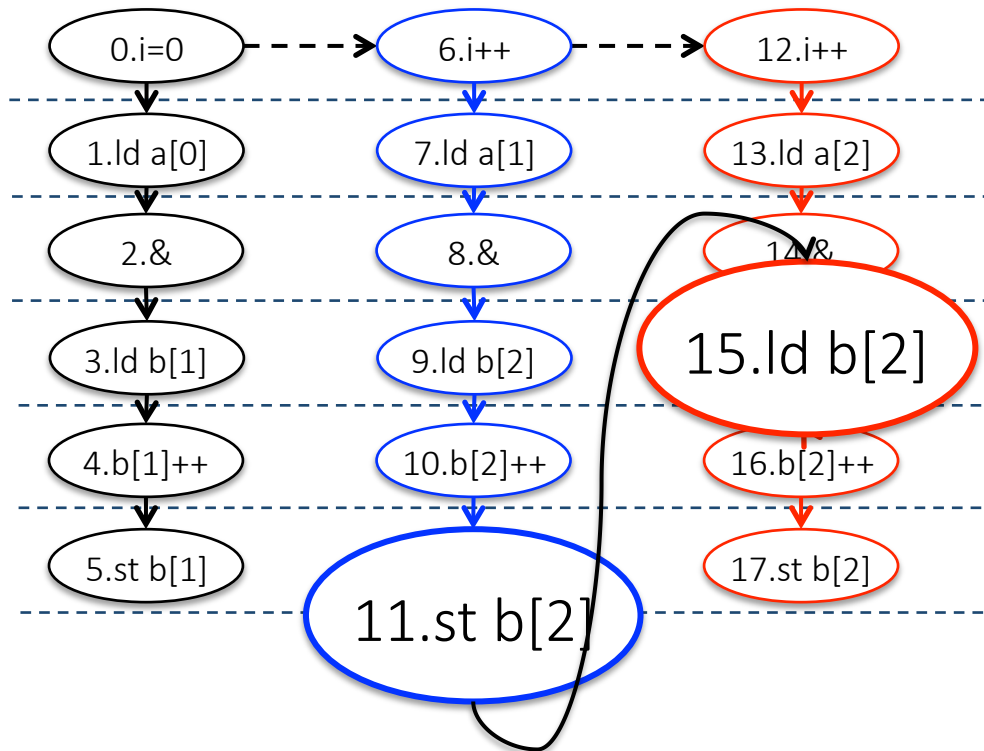
Input:

a[0] = 1;

a[1] = 2;

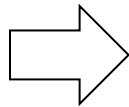
a[2] = 2;

...



Memory Ambiguation

```
for(i=0; i<N; ++i)
{
  bucket[ a[i] & 0x11 ]++;
}
```



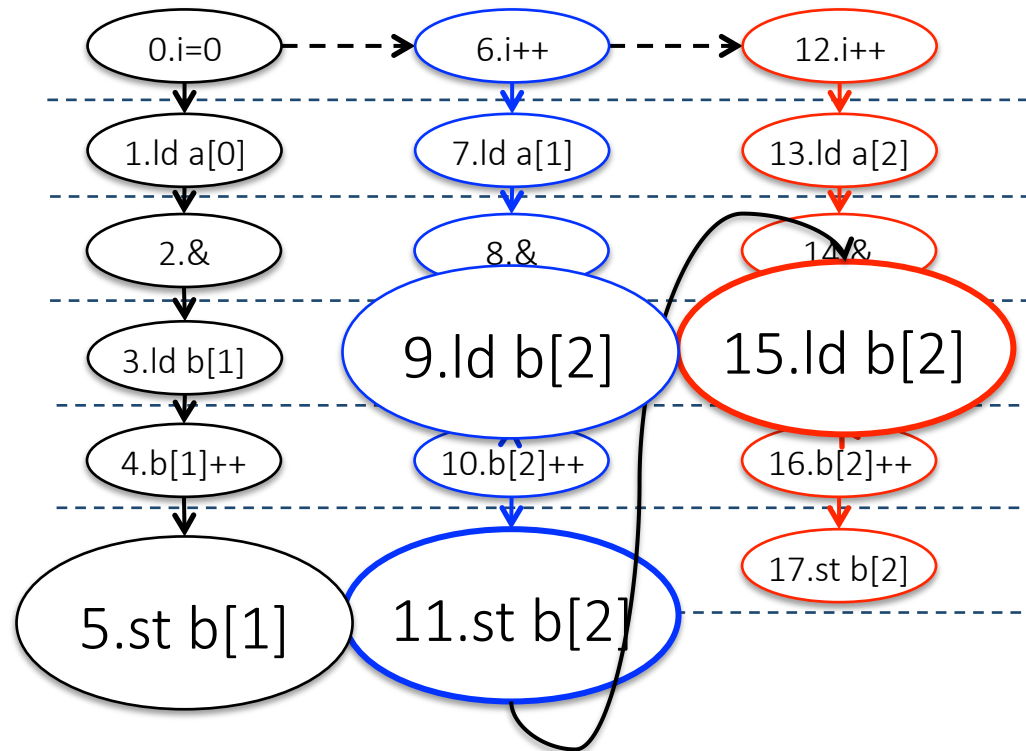
Input:

a[0] = 1;

a[1] = 2;

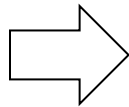
a[2] = 2;

...



Memory Ambiguation

```
for(i=0; i<N; ++i)
{
  bucket[ a[i] & 0x11 ]++;
}
```



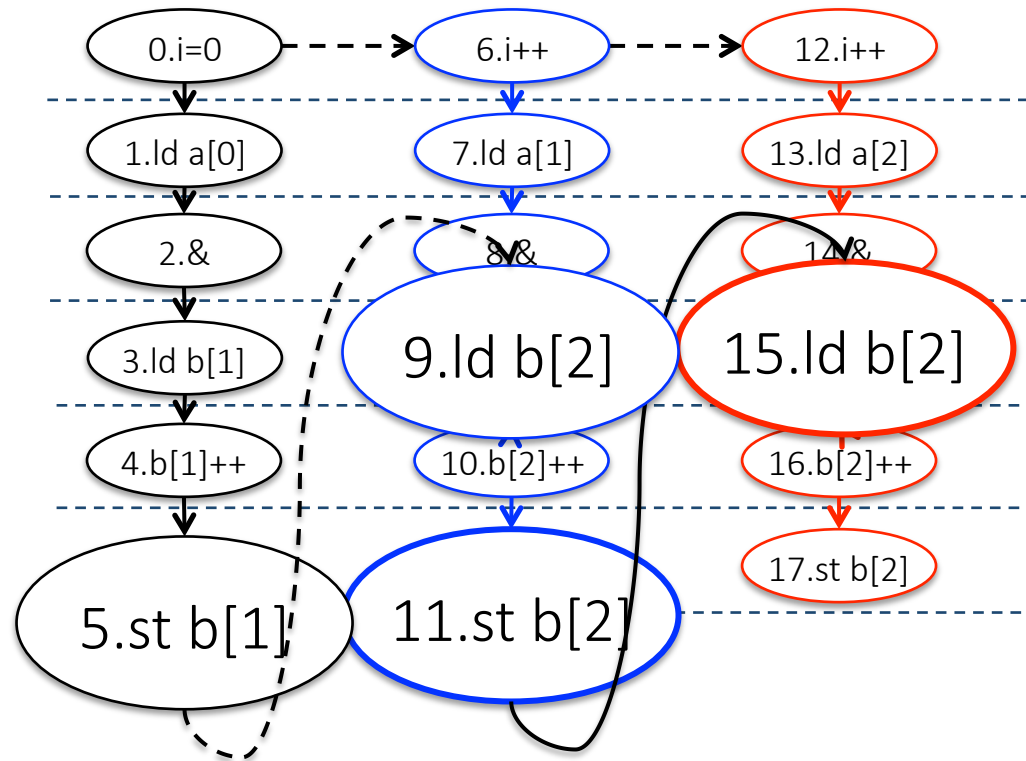
Input:

a[0] = 1;

a[1] = 2;

a[2] = 2;

...



Tutorial Outline

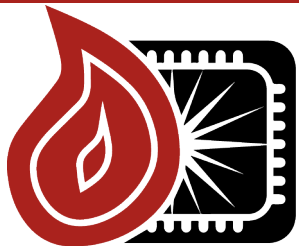
Time	Topic
8:30 am – 9:00 am	Introduction
9:00 am – 10:00 am	Pre-RTL Simulation Framework: Aladdin
10:00 am – 10:30 am	Break
10:30 am – 11:30 am	CAD & Benchmarks: HLS & MachSuite
11:30 am – 12:00 pm	Aladdin Validation and Case Studies
12:00 pm – 1:30 pm	Lunch
1:30 pm – 2:15 pm	Analytical Modeling: Lumos
2:15 pm – 2:45 pm	Workload Characterization Tool: WIICA
2:45 pm – 3:00 pm	Hands-on Exercise: Set up
3:00 pm – 3:30 pm	Break
3:30 pm – 5:00 pm	Hands-on Exercise



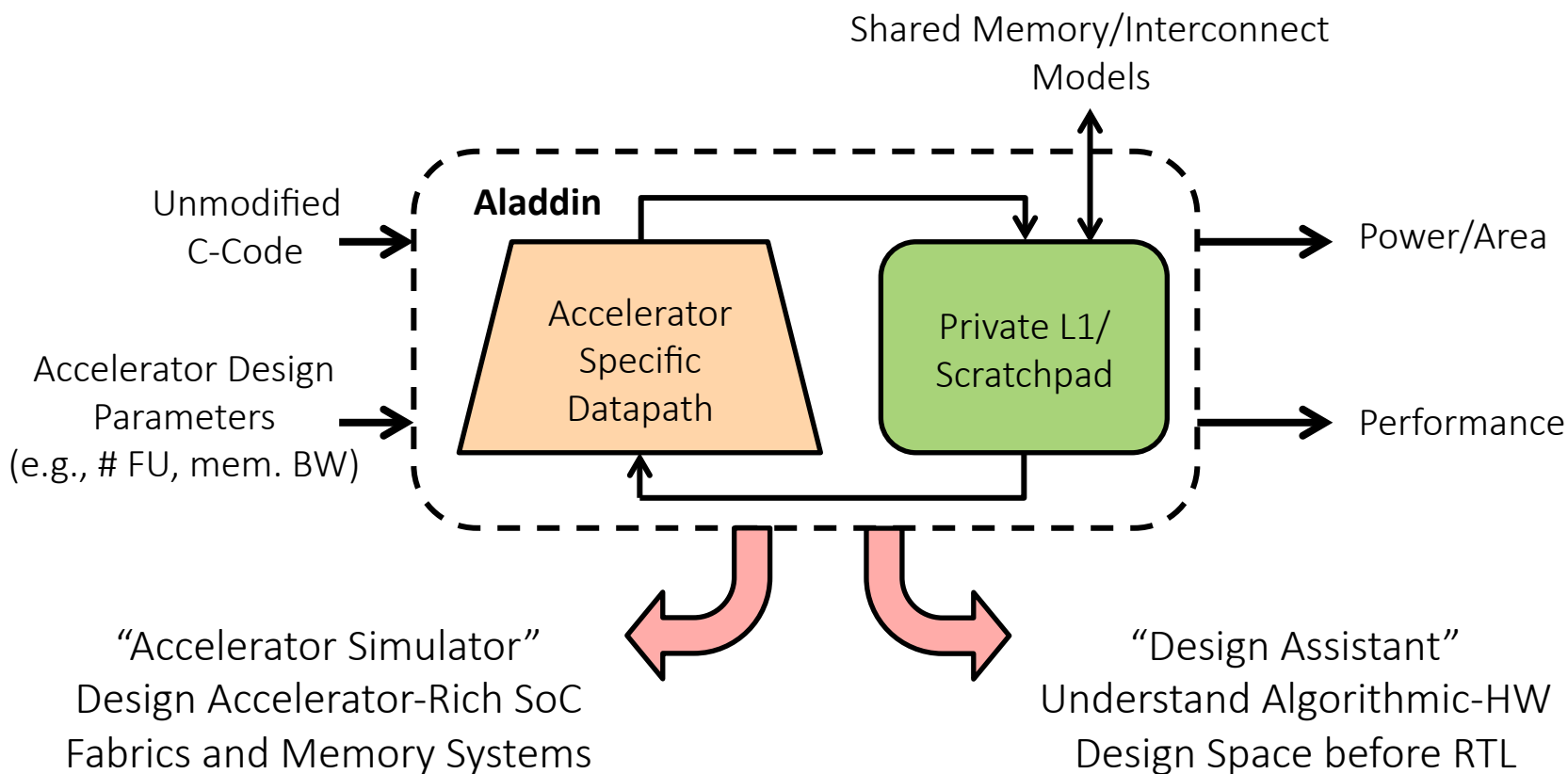
Tutorial Outline

Time	Topic
8:30 am – 9:00 am	Introduction
9:00 am – 10:00 am	Pre-RTL Simulation Framework: Aladdin
10:00 am – 10:30 am	Break
10:30 am – 11:30 am	CAD & Benchmarks: HLS & MachSuite
11:30 am – 12:00 pm	Aladdin Validation and Case Studies
12:00 pm – 1:30 pm	Lunch
1:30 pm – 2:15 pm	Analytical Modeling: Lumos
2:15 pm – 2:45 pm	Workload Characterization Tool: WIICA
2:45 pm – 3:00 pm	Hands-on Exercise: Set up
3:00 pm – 3:30 pm	Break
3:30 pm – 5:00 pm	Hands-on Exercise





Aladdin: A pre-RTL, Power-Performance Accelerator Simulator

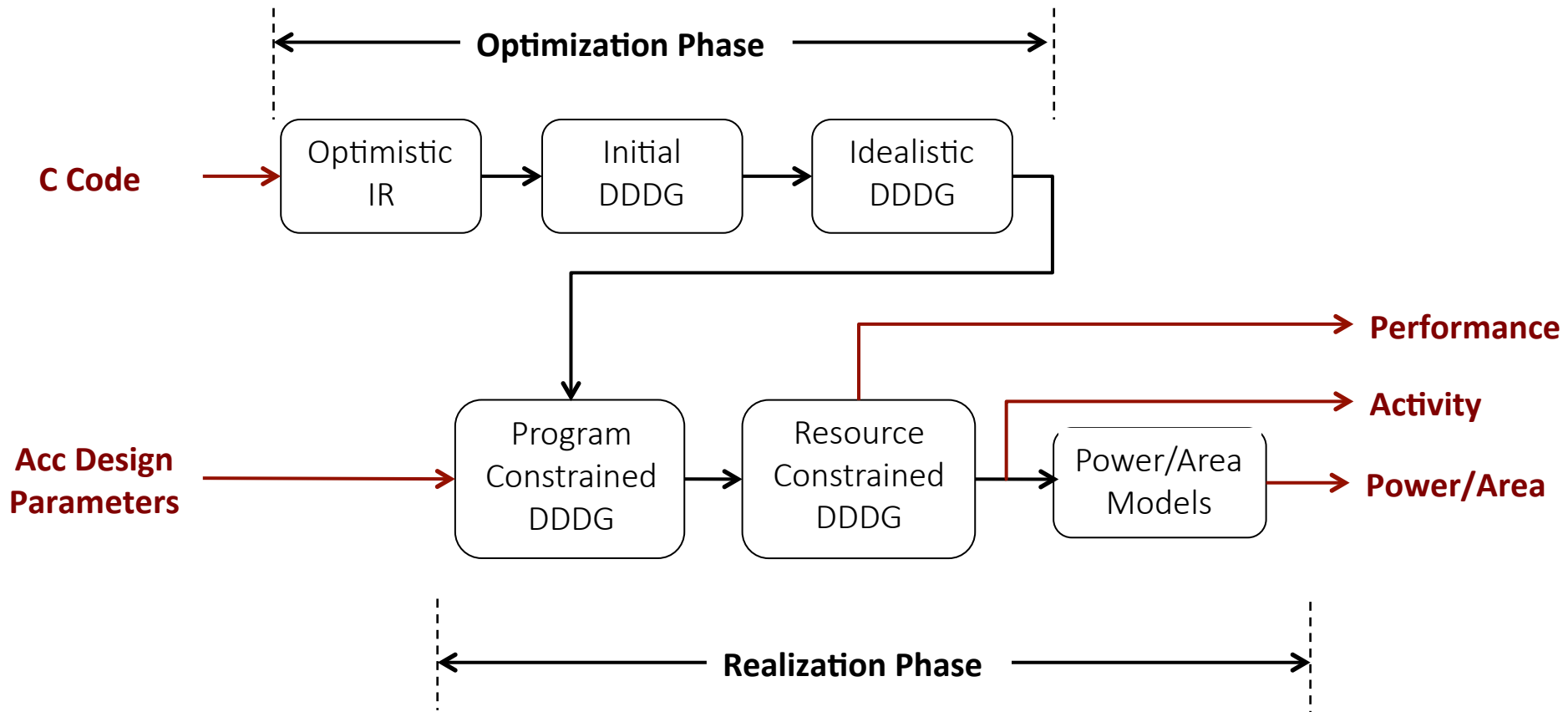


↑ **Flexibility**

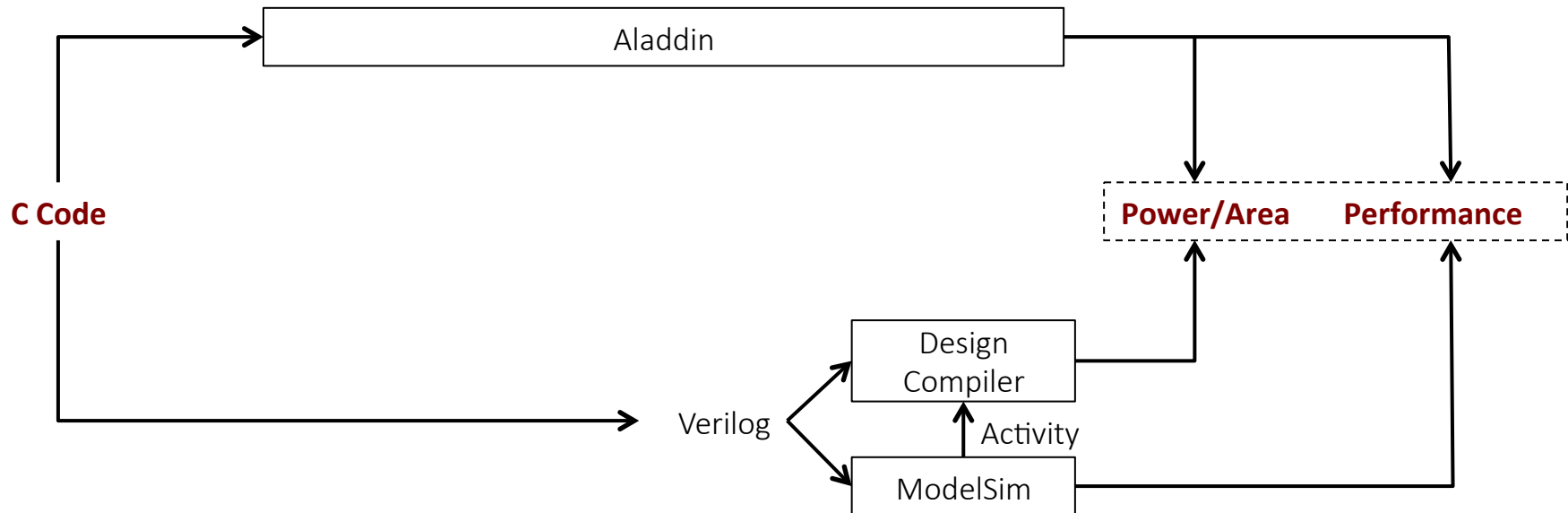
↑ **Programmability**

↑ **Design Cost**

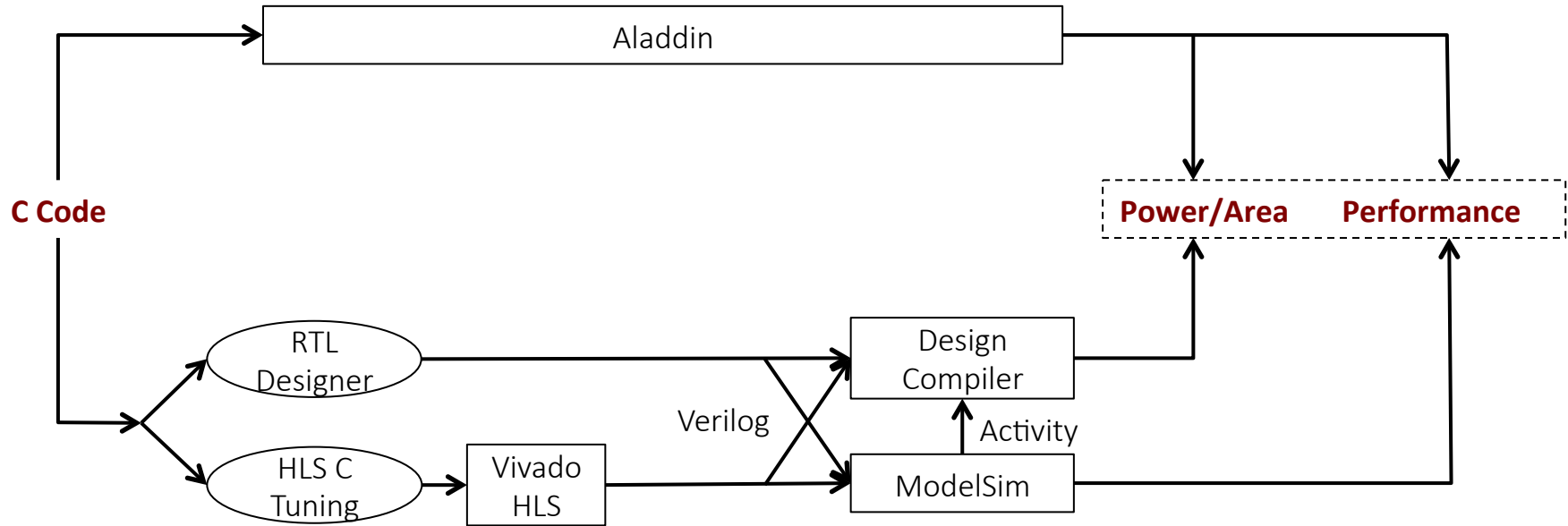
Aladdin Overview



Aladdin Validation



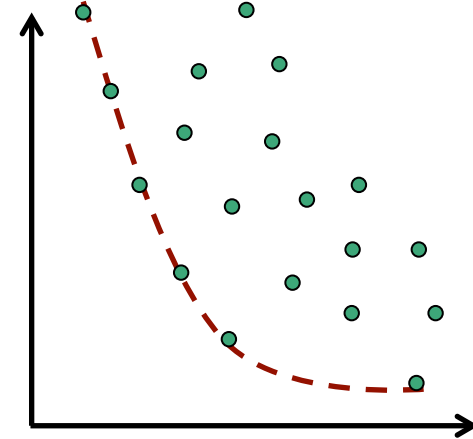
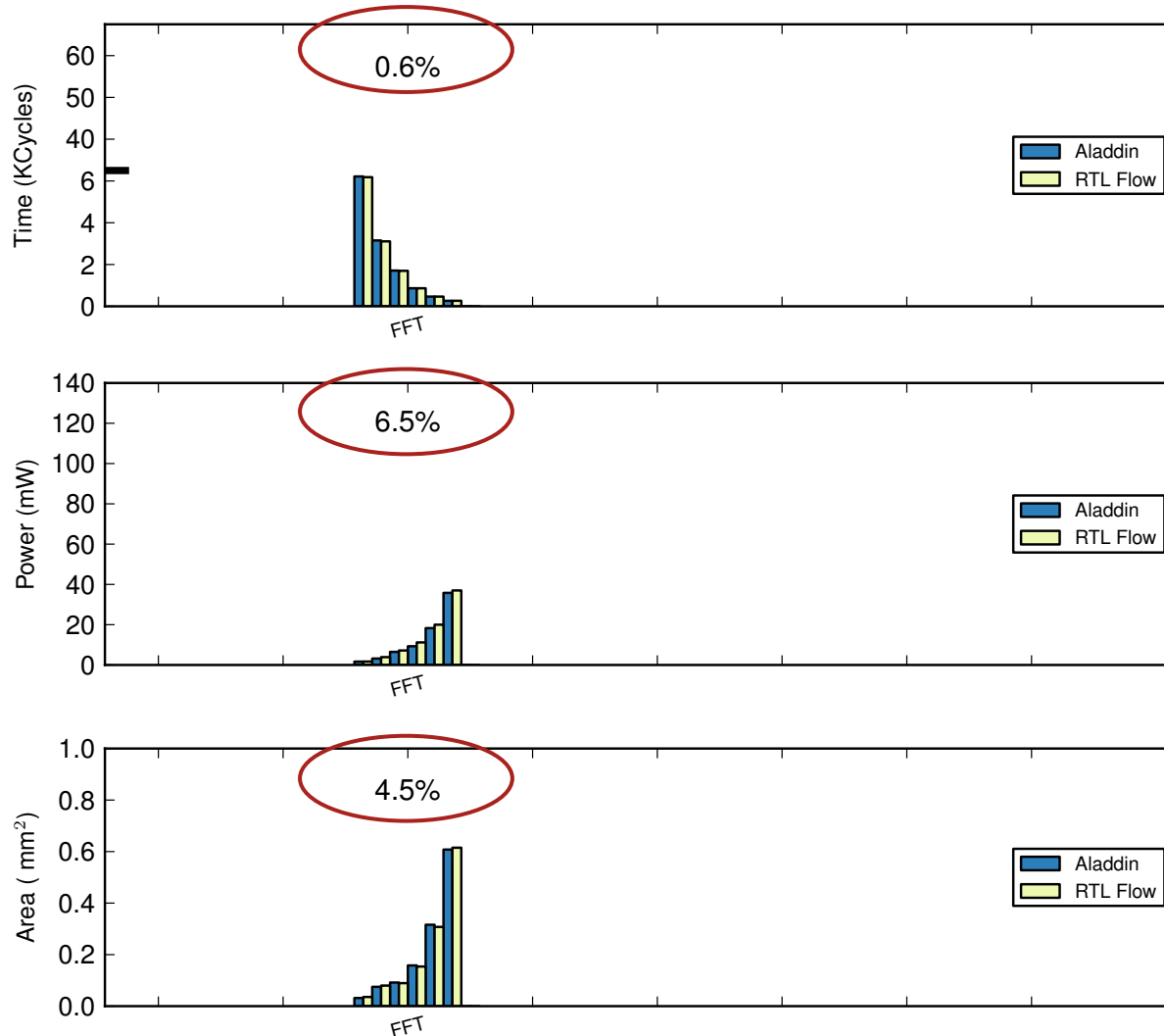
Aladdin Validation



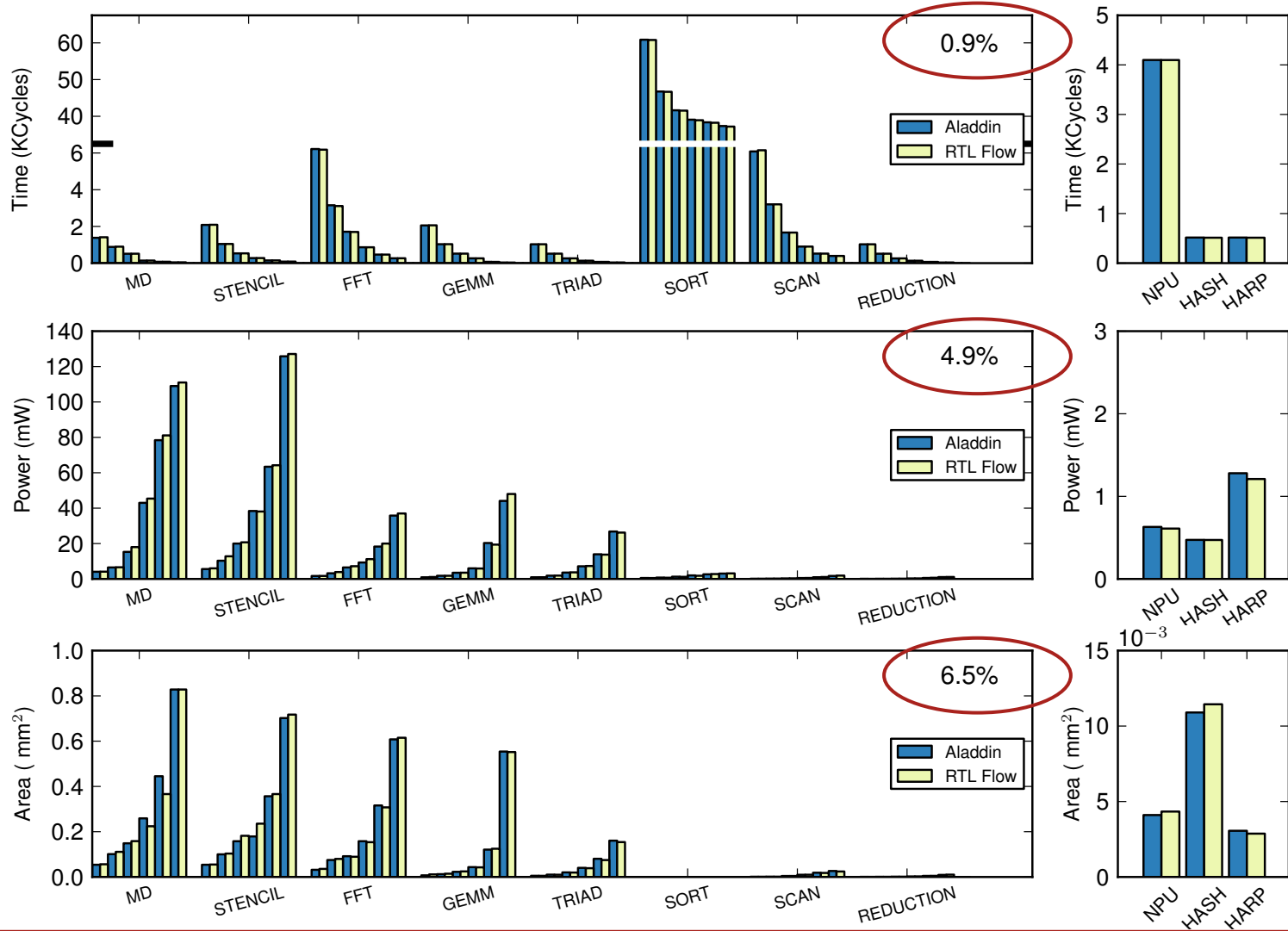
Validation Benchmarks

Type	Benchmark	Description	
SHOC Benchmark Suite	MD	Pairwise calculation of the L-J Potential	Optimized HLS Designs
	STENCIL	Apply 3x3 filter to an image	
	FFT	1D 512 FFT	
	GEMM	Blocked Matrix Multiply	
	TRIAD	Single Computation in DOALL loop	
	SORT	Radix Sort	
	SCAN	Parallel prefix sum	
	REDUCTION	Return sum of an array	
Proposed Accelerator Constructs	NPU	An individual neuron in a network [MICRO'12]	Hand RTL Designs
	Memcached	GET function in Memcached [ISCA'13]	
	HARP	Data partition accelerator [ISCA'13]	

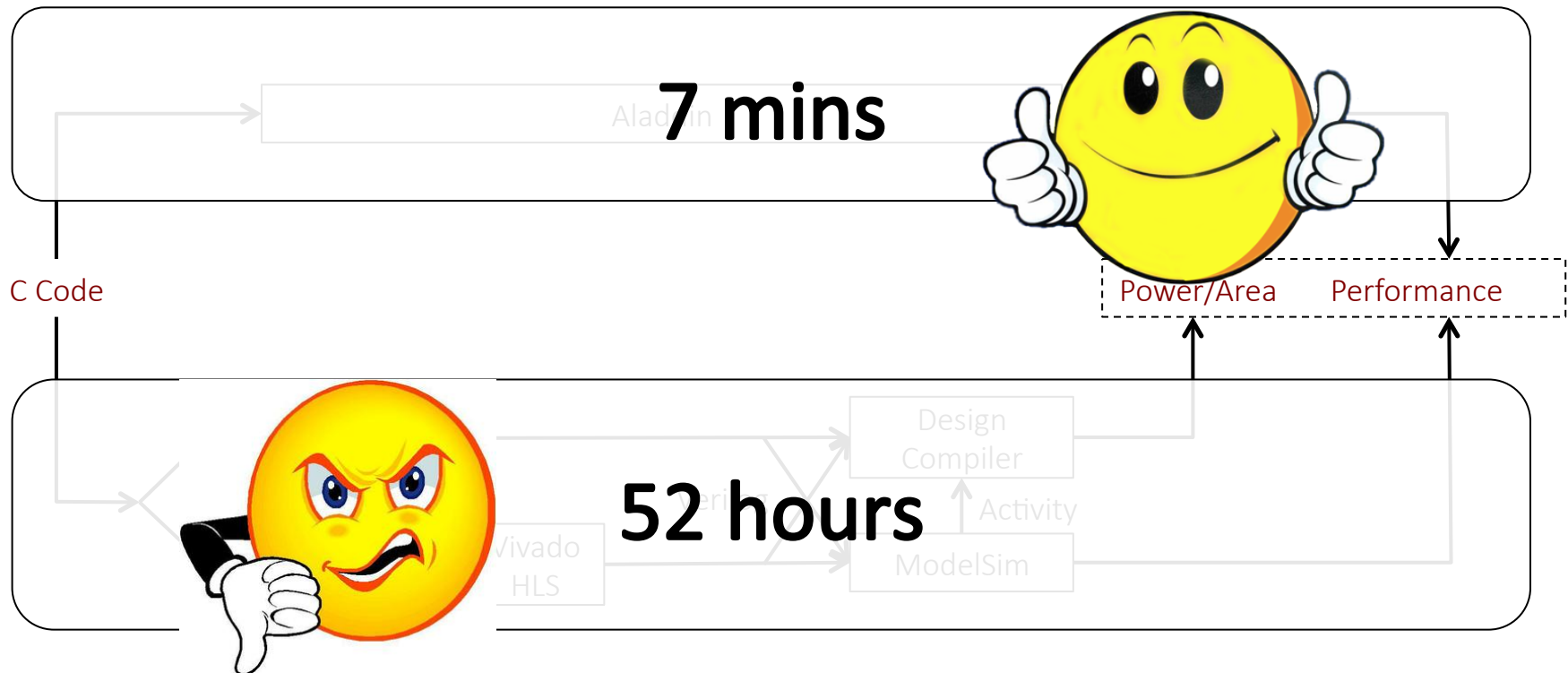
Aladdin Validation



Aladdin Validation



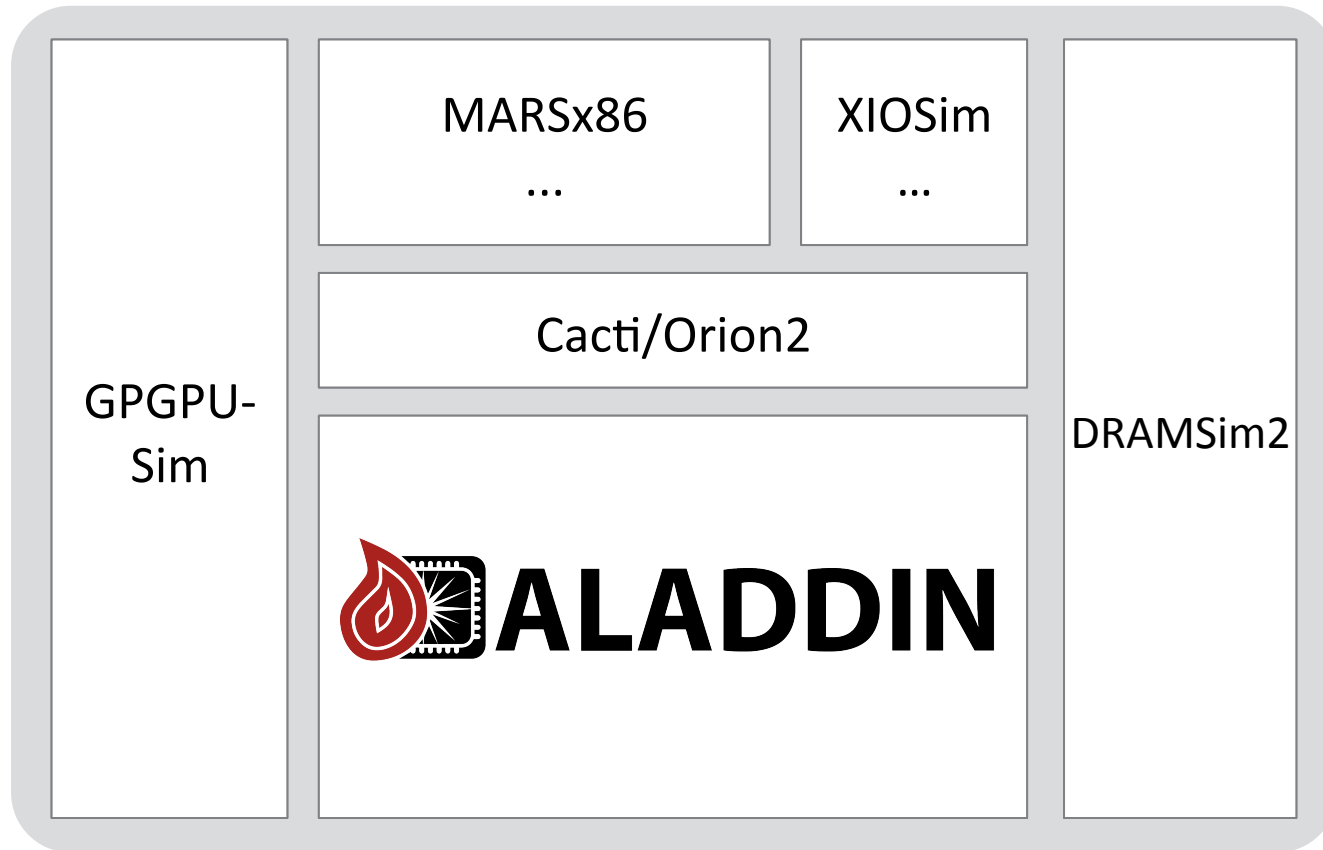
Aladdin enables rapid design space exploration for accelerators.



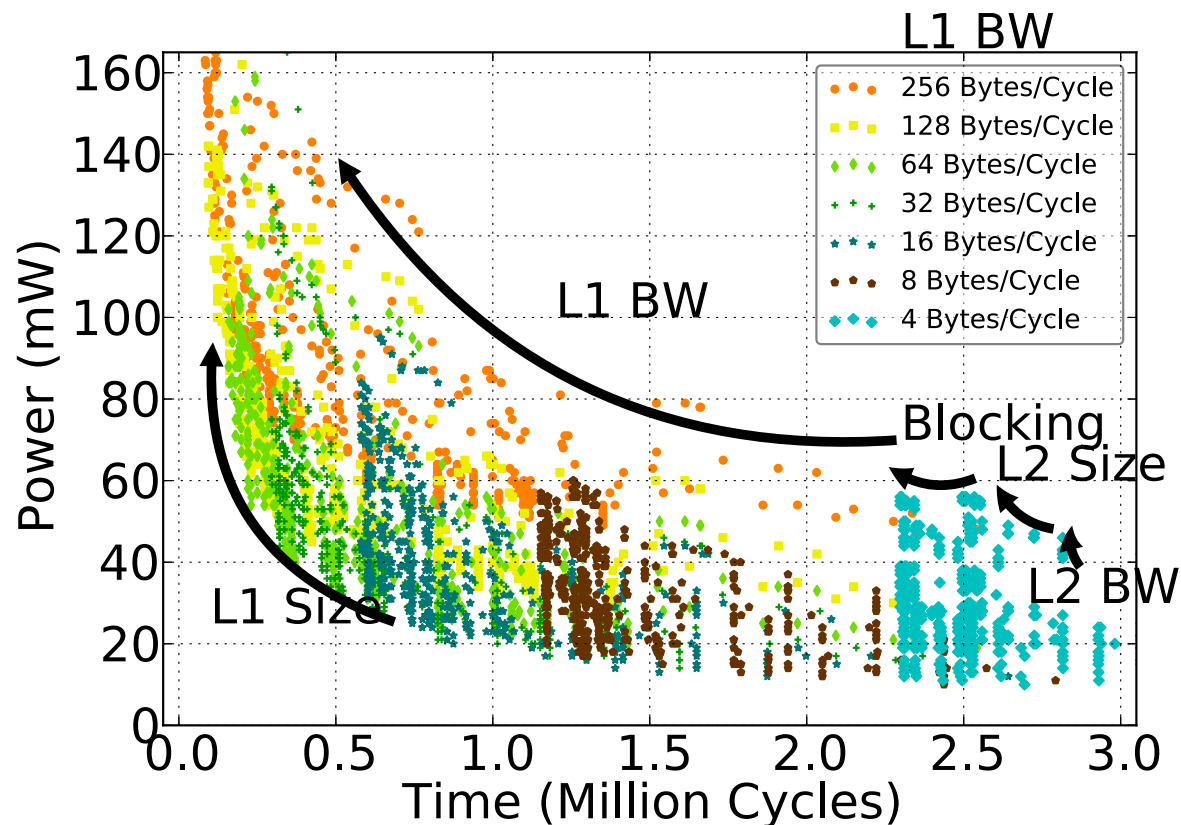
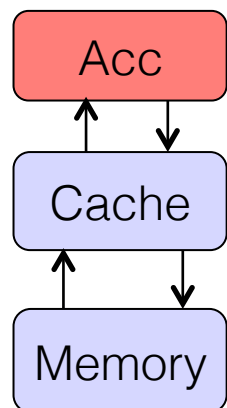
Limitations

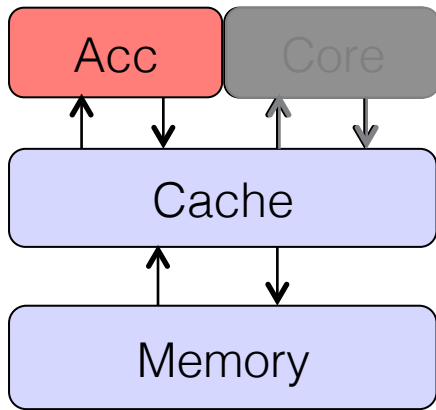
- Algorithm Choices
 - Aladdin generates a design space per algorithm
 - Can use Aladdin to quickly compare the design spaces of algorithms
- Input Dependent
 - Inputs that exercise all paths of the code
- Input C Code
 - Aladdin can create DDDG for any C code.
 - C constructs that require resources outside the accelerator, such as system calls and dynamic memory allocation, are not modeled.

Aladdin enables pre-RTL simulation of accelerators with the rest of the SoC.

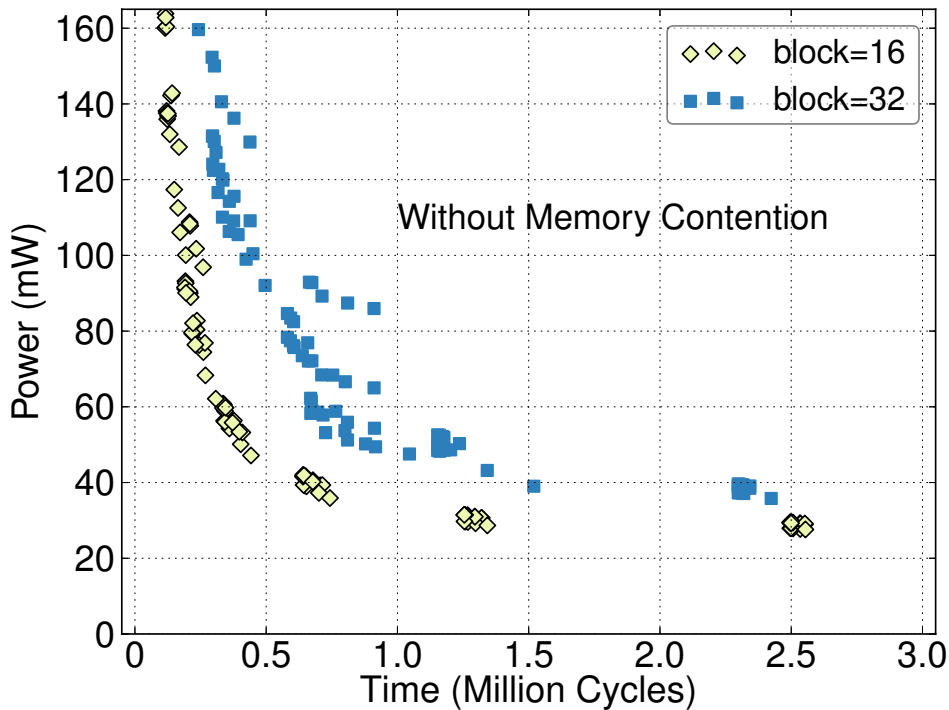


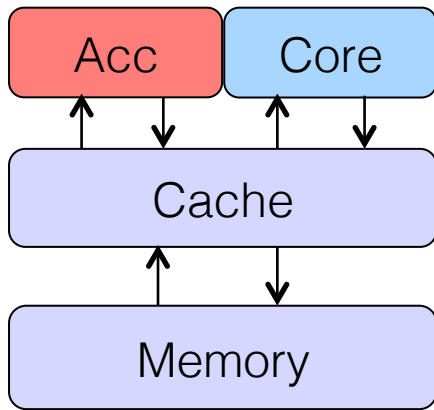
Simulating Accelerator with Memory System using Aladdin



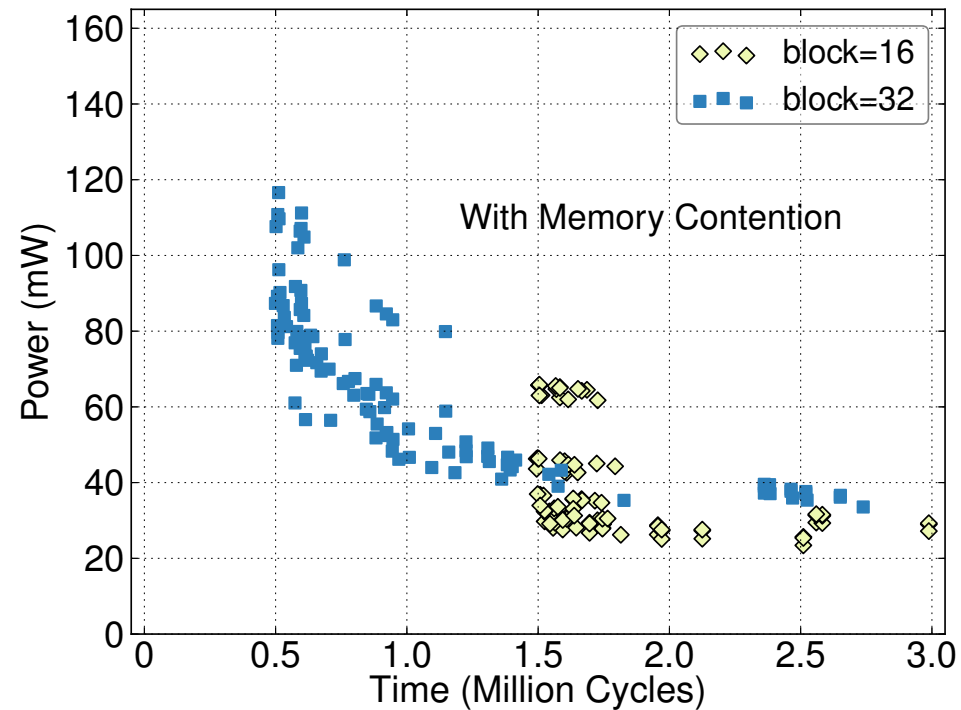
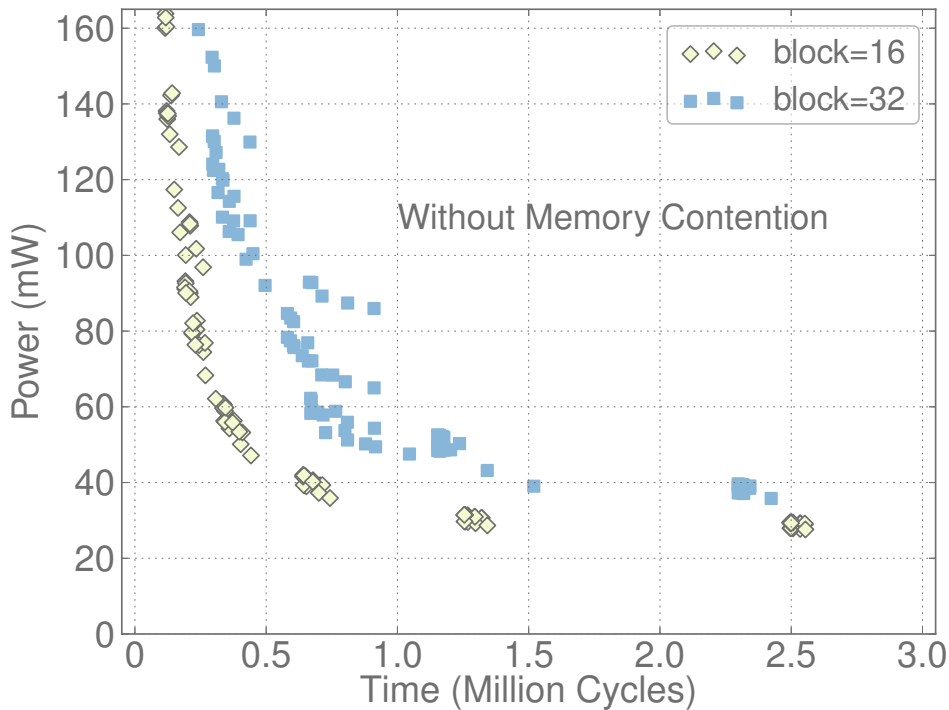


Modeling Accelerators in an SoC-like Environment





Modeling Accelerators in a SoC-like Environment



Publications

- Quantifying Acceleration – ISLPED 2013
 - Reagen, Shao, Wei, Brooks
- MachSuite – IISWC 2014
 - Reagen, Adolf, Shao, Wei, Brooks
- Aladdin – ISCA 2014
 - Shao, Reagen, Wei, Brooks

Accelerator Design, Tradeoffs, and Benchmarking

Vivado HLS

MachSuite [IISWC 2014]

Quantifying Acceleration [ISLPED 2013]

Brandon Reagen,
Yakun Sophia Shao, Bob Adolf,
Gu-Yeon Wei, David Brooks
Harvard University



Tutorial Outline

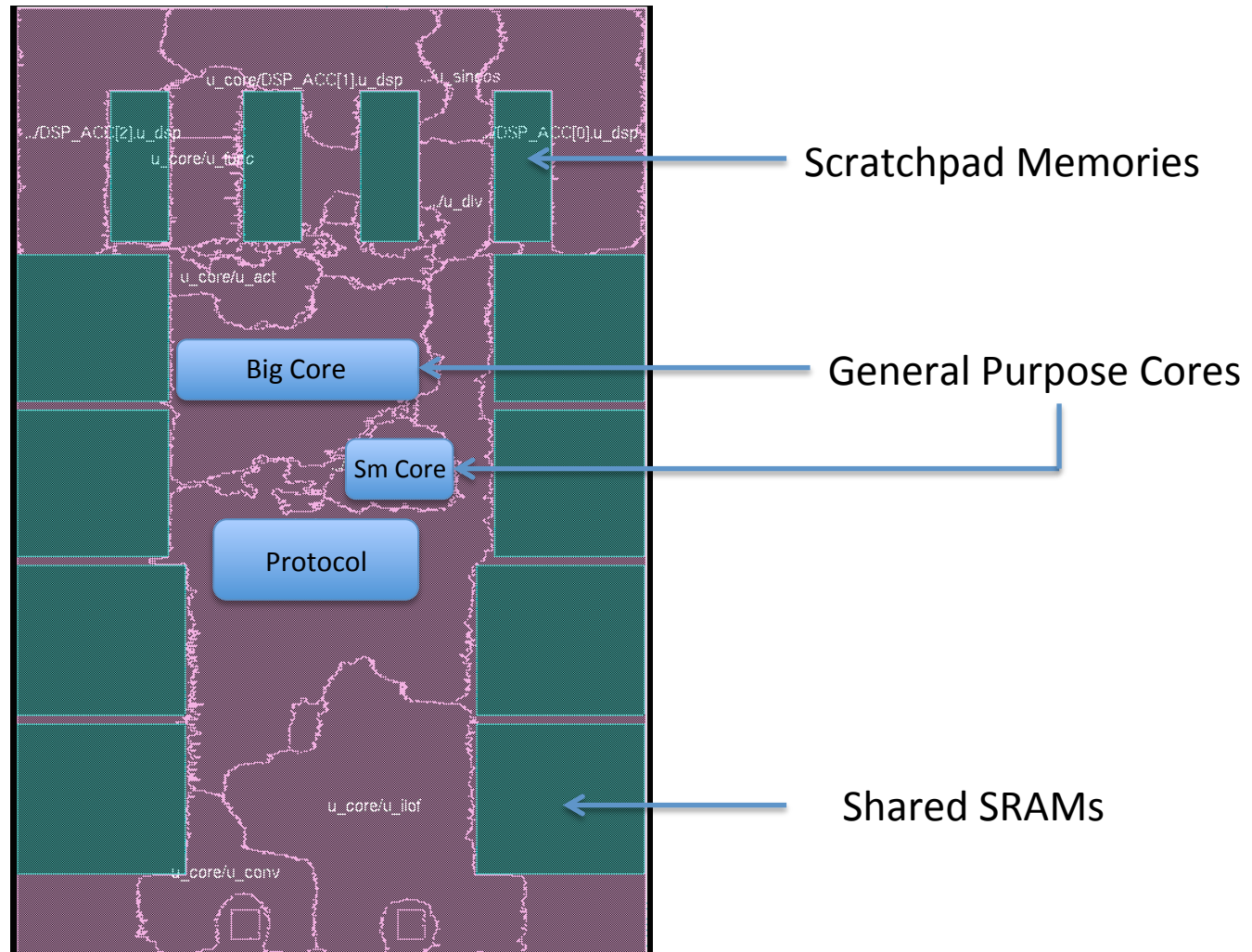
Time	Topic
8:30 am – 9:00 am	Introduction
9:00 am – 10:00 am	Pre-RTL Simulation Framework: Aladdin
10:00 am – 10:30 am	Break
10:30 am – 11:30 am	CAD & Benchmarks: HLS & MachSuite
11:30 am – 12:00 pm	Aladdin Validation and Case Studies
12:00 pm – 1:30 pm	Lunch
1:30 pm – 2:15 pm	Analytical Modeling: Lumos
2:15 pm – 2:45 pm	Workload Characterization Tool: WIICA
2:45 pm – 3:00 pm	Hands-on Exercise: Set up
3:00 pm – 3:30 pm	Break
3:30 pm – 5:00 pm	Hands-on Exercise

Accelerator

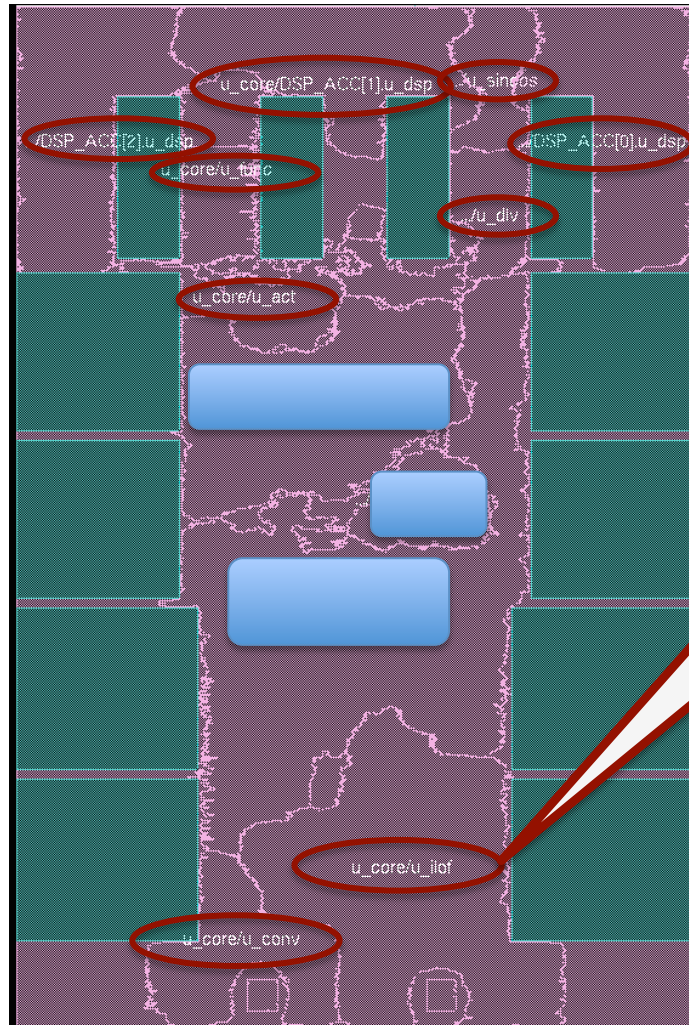
Design Tools and Workloads

- Introduction to HLS
- Coding for HLS
- Practices for Performance
- MachSuite
- Design Space Exploration

What is an Accelerator?



What is an Accelerator?



Fixed Function ASIC

Optical Flow Accelerator

- *Specialized* data path
- FSM controlled
- Single function

Orders of Magnitude
Energy Efficiency

CAD Tools

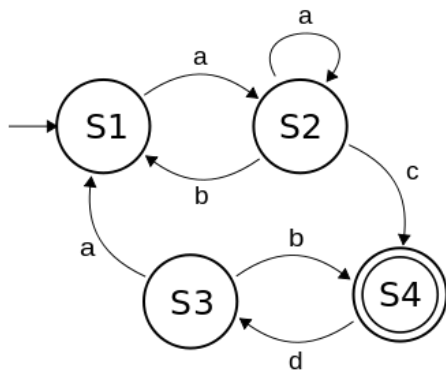
Accelerator Design Flow

Traditional Hardware Design

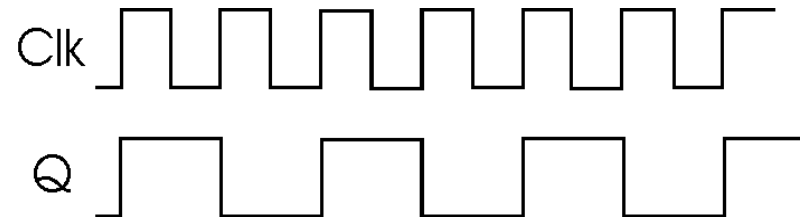
- Hand Code RTL
 - Understand problem
 - *SORT*
 - Consider various solutions
 - MERGE vs. RADIX
- Single Design Point
 - Power
 - Performance
 - Area

Traditional Hardware Design

- Pros
 - HDLs (Verilog) are **very** expressive
 - Designer can tune design perfectly
 - Engineers know and **trust** Verilog
 - Industry standard

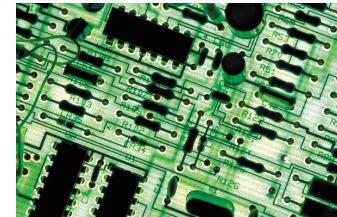


always @ (posedge Clk)



Traditional Hardware Design

- Cons
 - RTL is time consuming
 - Single implementations take weeks, months, to years
 - Intuition driven
 - Make decisions up front
 - Miscalculations can be costly



Traditional Hardware Design

- Cons
 - RTL is time consuming
 - Single implementations take weeks, months, to years
 - Intuition driven
 - Make decisions up front
 - Miscalculations can be costly



Traditional Hardware Design

- Cons
 - RTL is time consuming
 - Single implementations take weeks, months, to years
 - Intuition driven
 - Make decisions up front
 - Miscalculations can be costly

*No high level approach to
understand design space*

High Level Synthesis

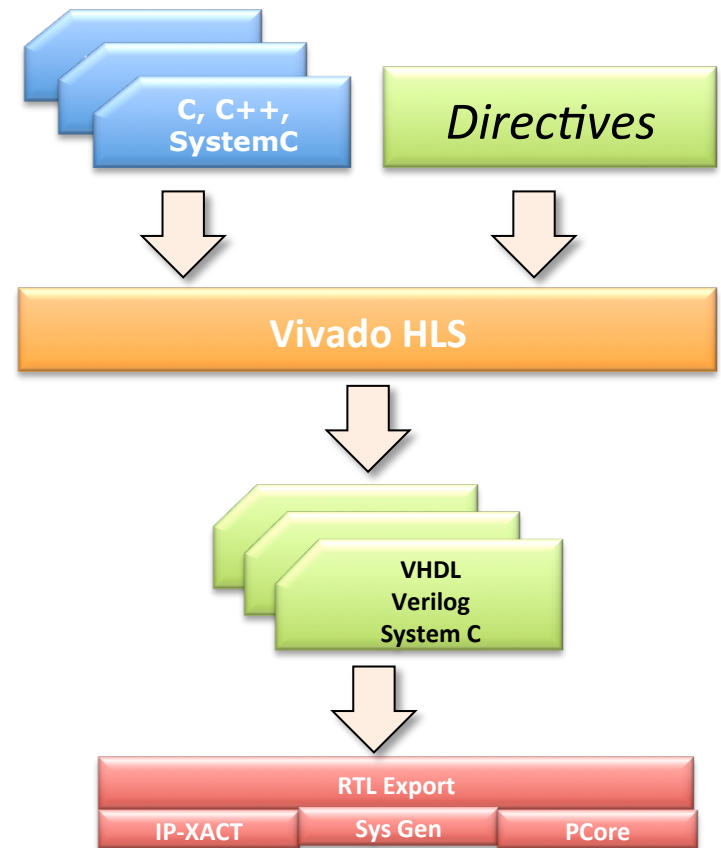
Vivado HLS

- Xilinx's HLS solution
 - C to Gates has been a pipe dream for decades
 - Vivado HLS paving the way
- Considers all aspects of Hardware Design



High Level Synthesis

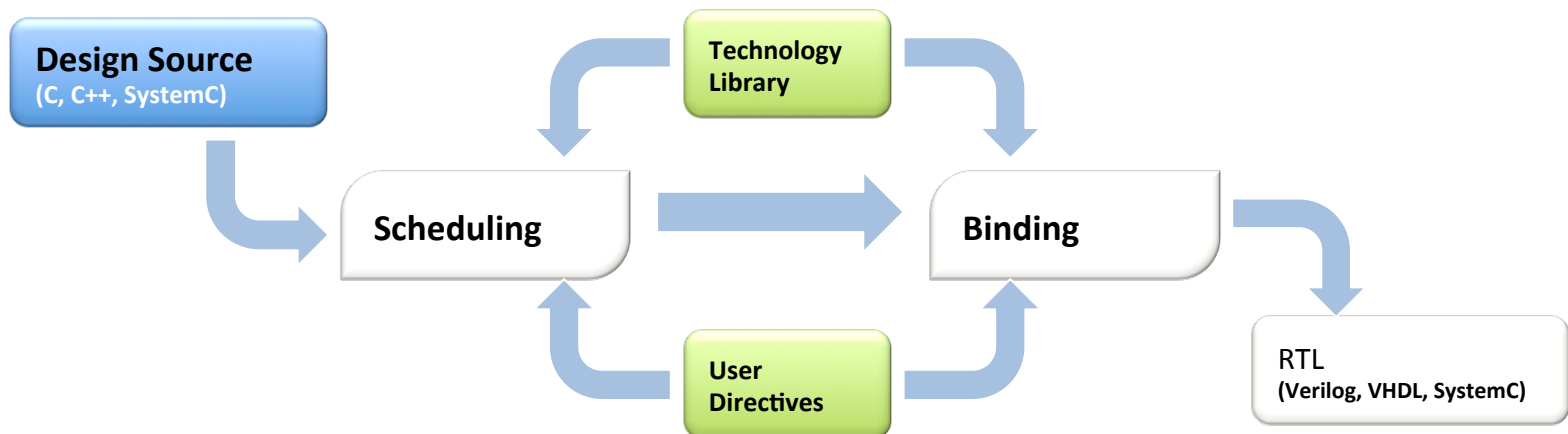
- Generate RTL from HLL
 - Restrict and abstract expressivity of HDL
 - Trust the compiler
 - Like writing C++ rather than ASM



High-Level Synthesis

Scheduling & Binding

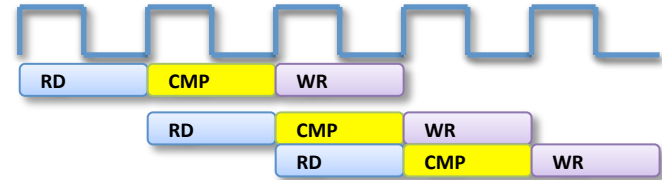
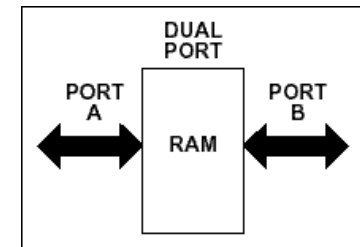
- Scheduling
 - Which clock cycle an operation will occur
 - Takes into account the control, dataflow and user directives
- Binding
 - Maps operations to available hardware
 - Takes into account component delays, user directives



High Level Synthesis

Vivado HLS

- Specify desired hardware
- Apply directives for performance



- Peripheral Support



High Level Synthesis

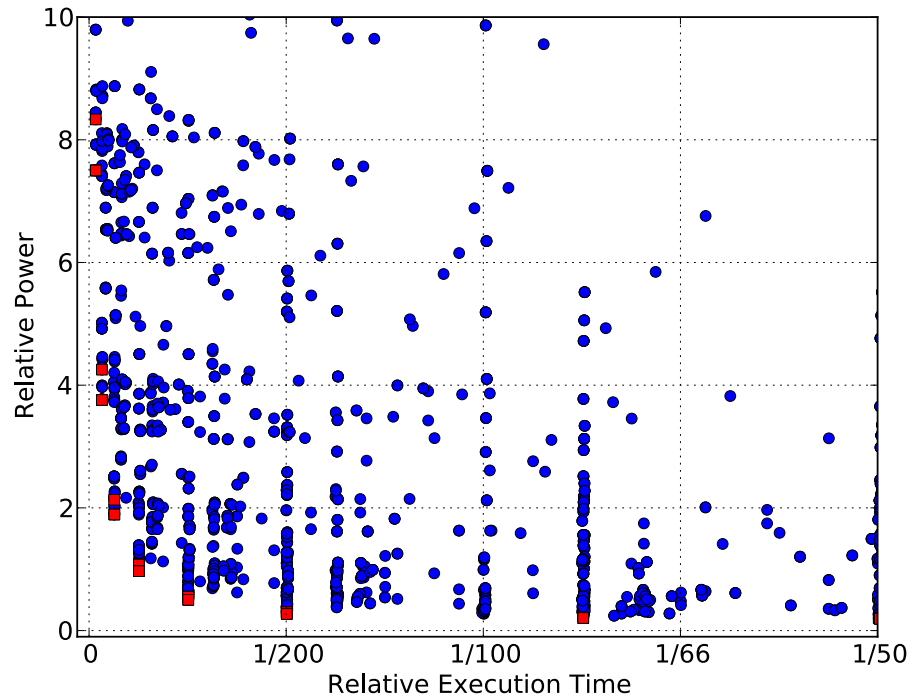
In this talk

- Focus on data path design
 - Achieving high performance
 - Loop Unrolling
 - Loop Pipelining
 - Functional Unit Selection and Allocation
 - Memory partitioning

High Level Synthesis

In this talk

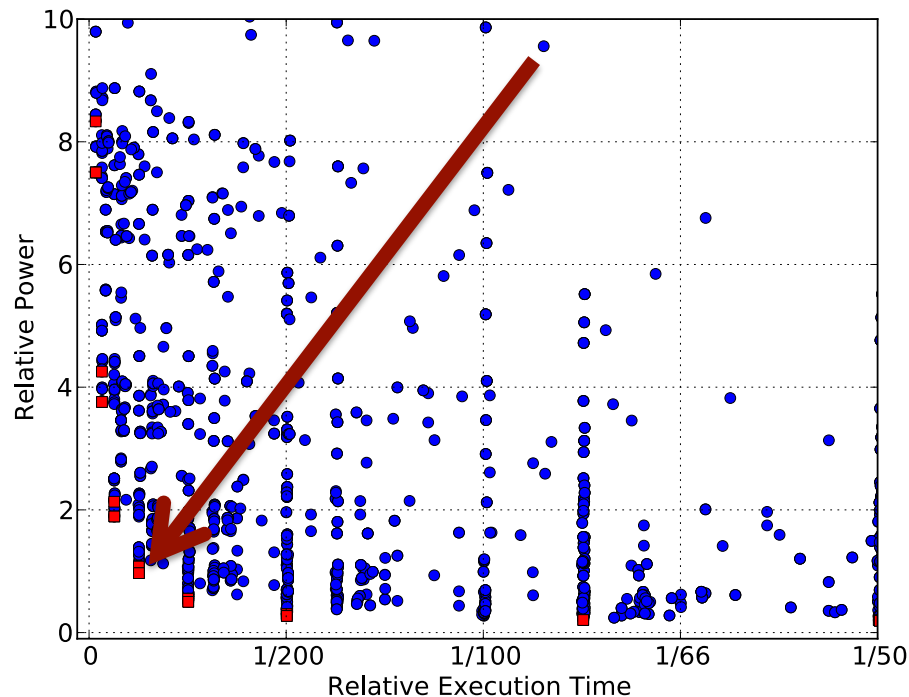
- Focus on data path design
 - Achieving high performance



High Level Synthesis

In this talk

- Focus on data path design
 - Achieving high performance



Accelerator Design, Tradeoffs, and Benchmarking

Vivado HLS

MachSuite [IISWC 2014]

Quantifying Acceleration [ISLPED 2013]

Brandon Reagen,
Yakun Sophia Shao, Bob Adolf,
Gu-Yeon Wei, David Brooks
Harvard University



Tutorial Outline

Time	Topic
8:30 am – 9:00 am	Introduction
9:00 am – 10:00 am	Pre-RTL Simulation Framework: Aladdin
10:00 am – 10:30 am	Break
10:30 am – 11:30 am	CAD & Benchmarks: HLS & MachSuite
11:30 am – 12:00 pm	Aladdin Validation and Case Studies
12:00 pm – 1:30 pm	Lunch
1:30 pm – 2:15 pm	Analytical Modeling: Lumos
2:15 pm – 2:45 pm	Workload Characterization Tool: WIICA
2:45 pm – 3:00 pm	Hands-on Exercise: Set up
3:00 pm – 3:30 pm	Break
3:30 pm – 5:00 pm	Hands-on Exercise

Accelerator

Design Tools and Workloads

- Introduction to HLS
- Coding for HLS
- Practices for Performance
- MachSuite
- Design Space Exploration

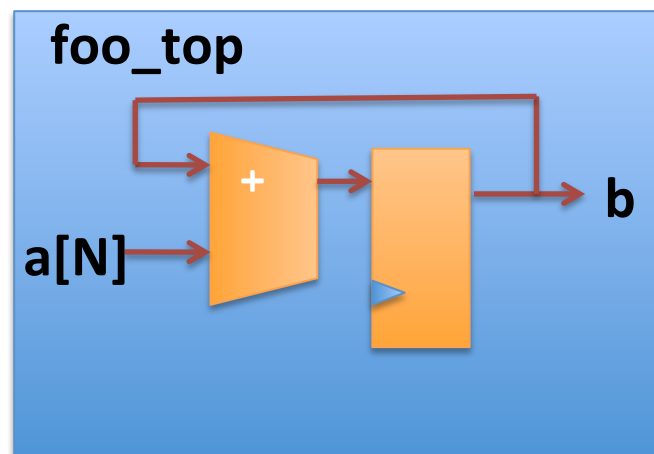
Leveraging HLS

Loop Unrolling

- By default, loops are rolled
 - Each C loop iteration
 - Implemented in the same state
 - Each C loop iteration
 - Implemented with same resources

```
void foo_top (...) {  
    ...  
    Add: for (i=3;i>=0;i--){  
        b = a[i] + b;  
    ...  
}
```

Synthesis

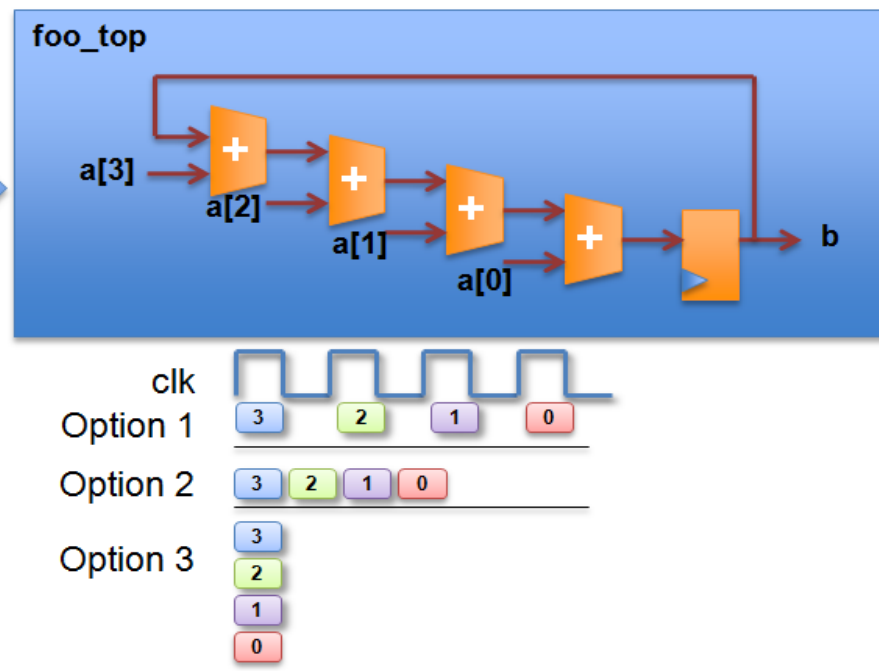


Loop Unrolling

- Unrolled Loops increase
 - Performance, Area, Power
- Lower energy

```
void foo_top (...) {  
    ...  
    Add: for  
    (i=3;i>=0;i--){  
        b = a[i] + b;  
    ...  
}
```

Unrolled

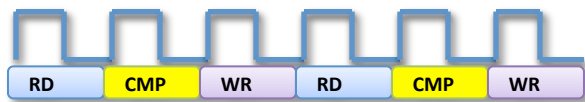


Loop Pipelining

Without Pipelining

```
Loop_tag : for( ll = 1 ; ll < 3 ; ll++ ) {  
  op_Read;  
  op_Compute;  
  op_Write;  
}
```

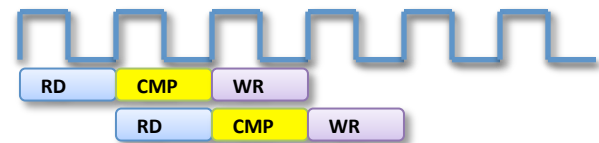
With Pipelining



Throughput = 3 cycles

Latency = 3 cycles

Loop Latency = 6 cycles



Throughput = 1 cycle

Latency = 3 cycles

Loop Latency = 4 cycles

Performance ↑

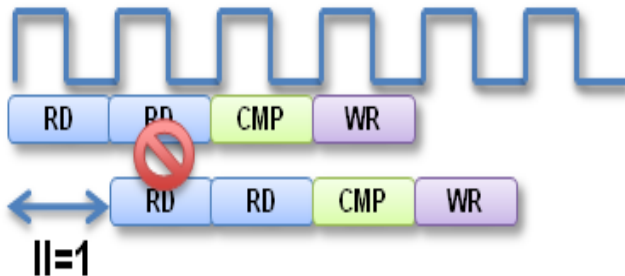
Area ↑

Power ↑

Loop Pipelining Initiation Intervals

- Iteration interval (Π) is the time loop must wait for the next execution to begin

(A) Pipeline with $\Pi=1$



```
void foo(m[2]...) {
```

```
  op_Read_m[0];
```

```
  op_Read_m[1];
```

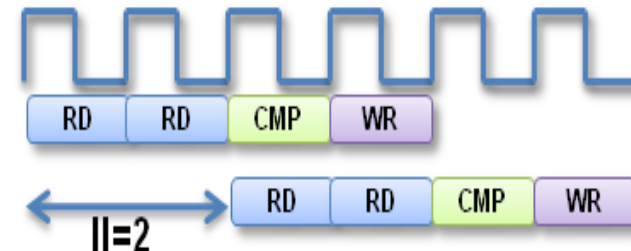
```
  op_Compute;
```

```
  op_Write;
```

```
}
```



(B) Pipeline with $\Pi=2$



- An $\Pi=1$ cannot be implemented
 - Port cannot be read at the same time
 - Similar effect with other resource limitations
 - For example if functions or multipliers etc. are limited

Hardware Resource Selection and Allocation

- ***Allocation*** directive limits different types
 - Operations
 - Functional Units
 - Functions
 - Maps multiple instantiations to same RTL

Hardware Resource Selection and Allocation

- **Allocation** directive limits different types
 - Operation
 - Functional Units
 - Functions
 - Maps multiple instantiations to same RTL
- Select operation's **implementation**
 - “Tag” functional units

$a[i] = b[i] * c[i]$



thisMult = b[i] * c[i];
a[i] = thisMul;

Hardware Resource Selection and Allocation

- **Allocation** directive limits different types
 - Operation
 - Functional Units
 - Functions
 - Maps multiple instantiations to same RTL
- Select operation's **implementation**

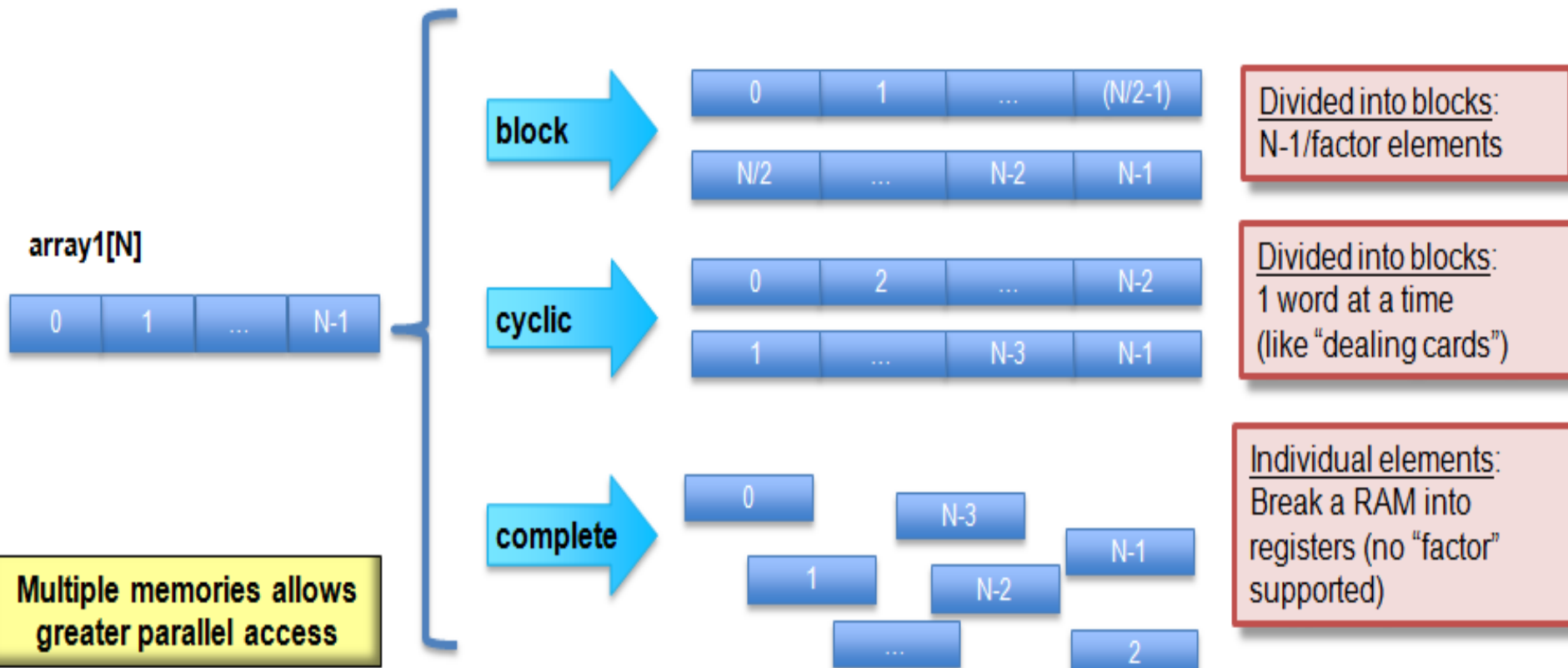
- Select core for *thisMult*

thisMult = b[i] * c[i];

Core	Description
Mul	Combinational mult
Mul3s	3-Stage pipelined mult
MulnS	HLS determine stages

Array Partitioning

- Partitioning breaks an array into smaller elements



High Level Synthesis

Vivado HLS

- Covered most impactful directives
 - Loop Unrolling
 - Loop Pipelining
 - Memory Partitioning
 - Resource selection and allocation
- Many more exist..
 - ~30 explicit directives with multiple dimensions
 - With the correct directives and code,
designs can often compete with handwritten RTL

Coding For HLS

HLS Compatible Code

- Language Support
 - What you CAN do
- Unsupported Language features
 - What you CAN NOT do
- Building on Language standards
 - Adding expressivity in HLLs
 - Coding Suggestions..

Language Support

- Vivado HLS supports C, C++, and SystemC
 - Provided static definition at compile time
 - run time definitions are not synthesizable
- Data types
 - Float and double cores generated
 - Supports arbitrary precision types
 - User defined fixed point types
- Validation & Verification Environment
 - Complete bit-accurate validation of the C model
 - C-RTL co-simulation verification solution

Unsupported Features

Dynamic Memory Allocation

- Dynamic memory allocation
 - Requires construction of hardware at runtime
 - *malloc*, *alloc*, *free* are not synthesizable
 - Use persistent static variables and fixed-size arrays

```
long long x = malloc (sizeof(long long));  
  
int* arr = malloc (64 * sizeof(int));
```



```
static long long x;  
  
int array[64];
```

Unsupported Features

System Calls

- System calls
 - C system calls do not have hardware
 - printf(), getc(), time(), ...
 - Vivado HLS will ignore system calls
 - Use “__SYNTHESIS__”
 - Synthesis will ignore block
 - Automatically defined

**Only read this code if macro
__SYNTHESIS__ is not set**

```
void foo (...) {  
    #ifndef __SYNTHESIS__  
  
    Code will be seen by simulation.  
  
    But not synthesis.  
  
    #endif  
}
```


Unsupported Features

General Pointer Casting

- Pointer reinterpretation
 - Can't cast a pointer to a different type

```
struct {  
    short first;  
    short second;  
} pair;  
  
*(unsigned*)pair = -1U;
```

Solution

assign values using original type

```
struct {  
    short first;  
    short second;  
} pair;  
  
pair.first = -1U;  
pair.second = -1U;
```

- Pointer casting allowed between native integer types

Unsupported Features

Recursive Functions

- Avoid recursive functions
 - Not synthesizable in general
 - Code re-entrance indirectly uses dynamic memory allocation

```
unsigned foo (unsigned n) {  
    if (n == 0 || n == 1) return 1;  
    return (foo(n-2) + foo(n-1));  
}
```

Not synthesizable: Endless recursion

Language Support

- Lots of support
- Pointers bring conservativeness
 - Memory disambiguation
- Suggestion
 - Help the compiler for best results
 - Limit pointer use
 - Limit potential for false dependencies

Powerful tool –

but conservatism for correctness can ruin performance

Implementation and Co-optimizations

Importance Implementation

- Consider Simple Application Kernel
 - In place SCAN

Importance Implementation

- Consider Simple Application Kernel
 - In place SCAN

Data = [1, 10, 4, 5]



Importance Implementation

- Consider Simple Application Kernel
 - In place SCAN

Data = [1, 10, 4, 5]  Scan(data)

Importance Implementation

- Consider Simple Application Kernel
 - In place SCAN

Data = [1, 10, 4, 5] $\longrightarrow$ Scan(data) $\longrightarrow$ Data = [1, 11, 15, 20]

Importance Implementation

- Consider Simple Application Kernel
 - In place SCAN

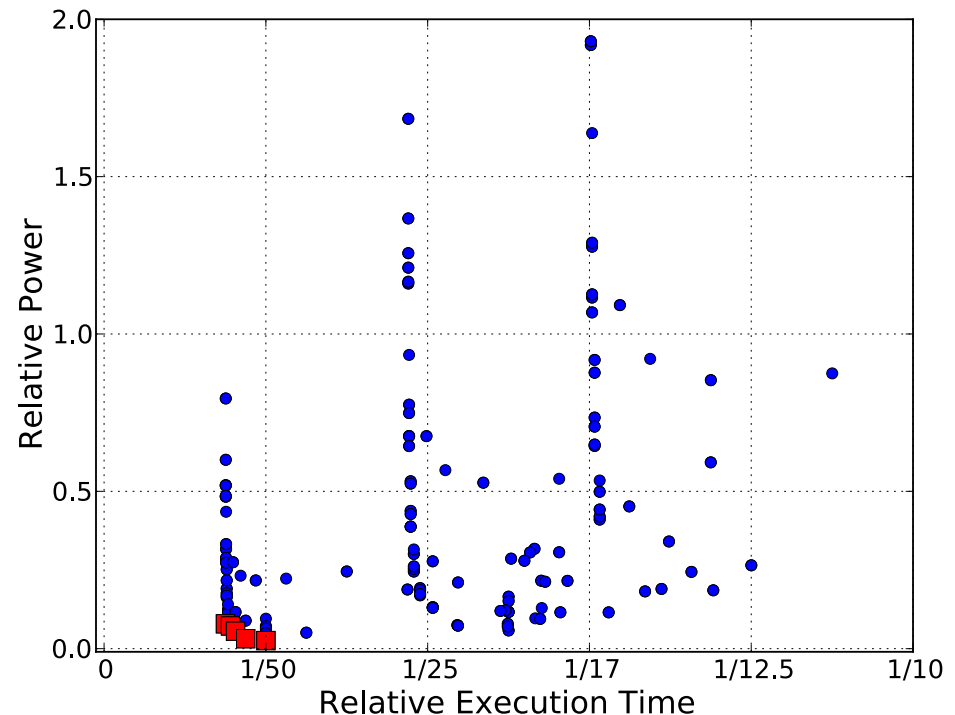
Data = [1, 10, 4, 5] $\longrightarrow$ Scan(data) $\longrightarrow$ Data = [1, 11, 15, 20]

$$Data[x] = \sum_{i=0}^{i=x} Data[i]$$

Importance Implementation

- Consider Simple Application Kernel
 - In place SCAN

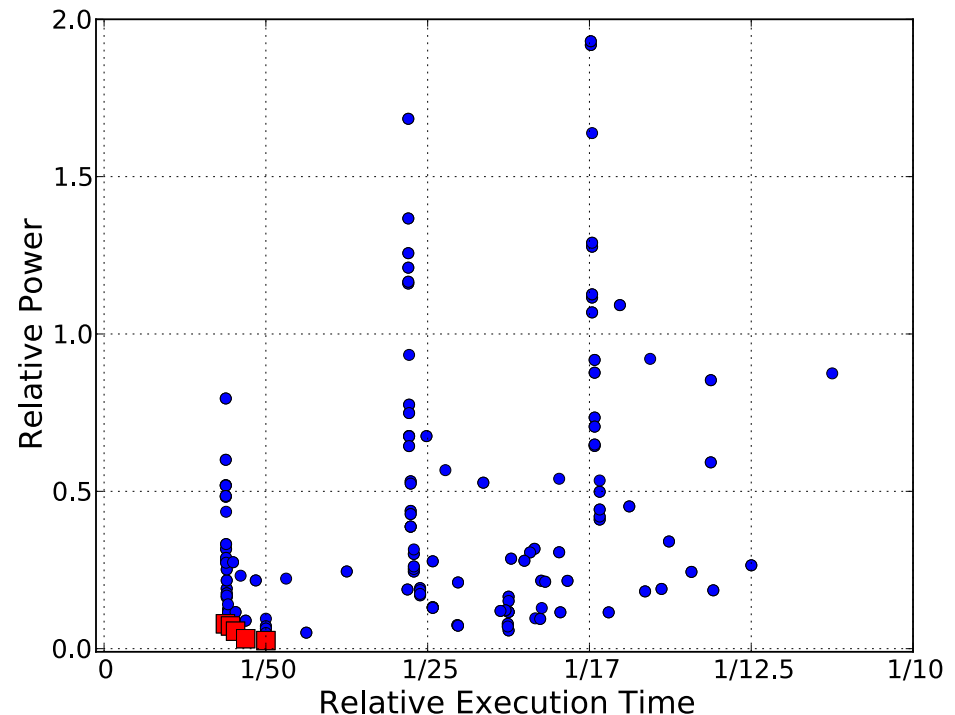
*different frequencies



Importance Implementation

- Consider Simple Application Kernel

- Resource Contention
- Cannot resolve RAW hazard



Importance Implementation

- Consider Simple Application Kernel
 - Increases II
 - Instantiates more HW
 - But compiler cannot make good use of it

Importance Implementation

- Consider Simple Application Kernel
 - Increases II
 - Instantiates more HW
 - But cannot make good use of it

```
@I [HLS-10] -----
@I [HLS-10] -- Scheduling module 'scan'
@I [HLS-10] -----
@I [SCHED-11] Starting scheduling ...
@I [SCHED-61] Pipelining loop 'scan_loop'.
@W [SCHED-68] Unable to enforce a carried dependency constraint (II = 1, distance = 1)
  between 'store' operation (scan.c:14) of variable 'tmp_3_14' on array 'out_r' and
  'load' operation ('out_load', scan.c:14) on array 'out_r'.
@W [SCHED-69] Unable to schedule 'load' operation ('out_load', scan.c:14) on array
  'out_r' due to limited memory ports.
@I [SCHED-61] Pipelining result: Target II: 1, Final II: 18, Depth: 18.
@I [SCHED-11] Finished scheduling.
```

Importance Implementation

- Consider Simple Application Kernel
 - Increases II
 - Instantiates more HW
 - But cannot make good use of it

```
@I [HLS-10] -----  
@I [HLS-10] -- Scheduling module 'scan'  
@I [HLS-10] -----  
@I [SCHED-11] Starting scheduling ...  
@I [SCHED-61] Pipelining loop 'scan_loop'.  
@W [SCHED-68] Unable to enforce a carried dependency constraint (II = 1, distance = 1)  
  between 'store' operation (scan.c:14) of variable 'tmp_3_14' on array 'out_r' and  
  'load' operation ('out_load', scan.c:14) on array 'out_r'.  
@W [SCHED-69] Unable to schedule 'load' operation ('out_load', scan.c:14) on array  
  'out_r' due to limited memory ports.  
@I [SCHED-61] Pipelining result: Target II: 1, Final II: 18, Depth: 18.  
@I [SCHED-11] Finished scheduling.
```

Importance of co-design

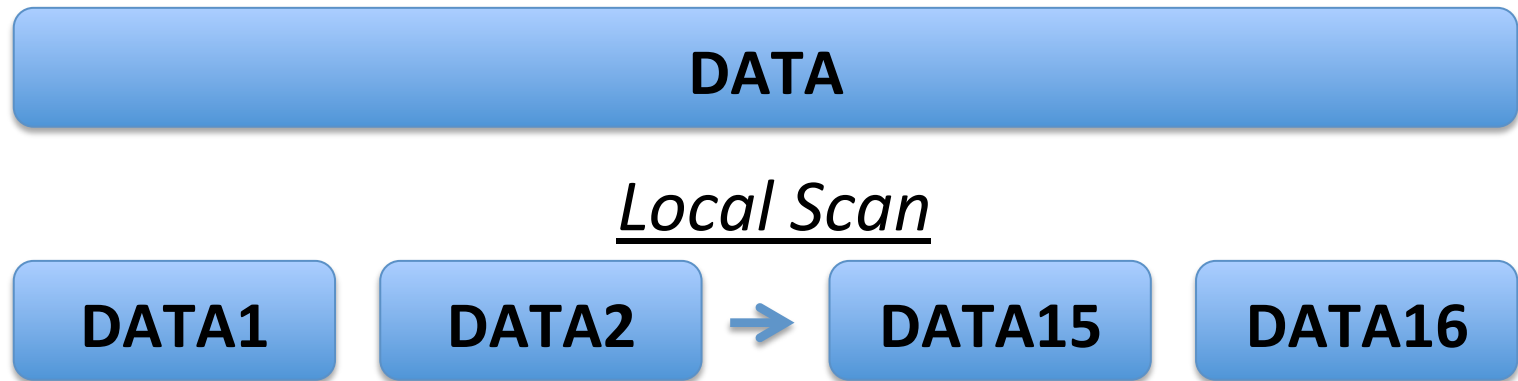
- Instead of the naïve, serial SCAN

Can we try something smarter?

What about parallel SCAN?

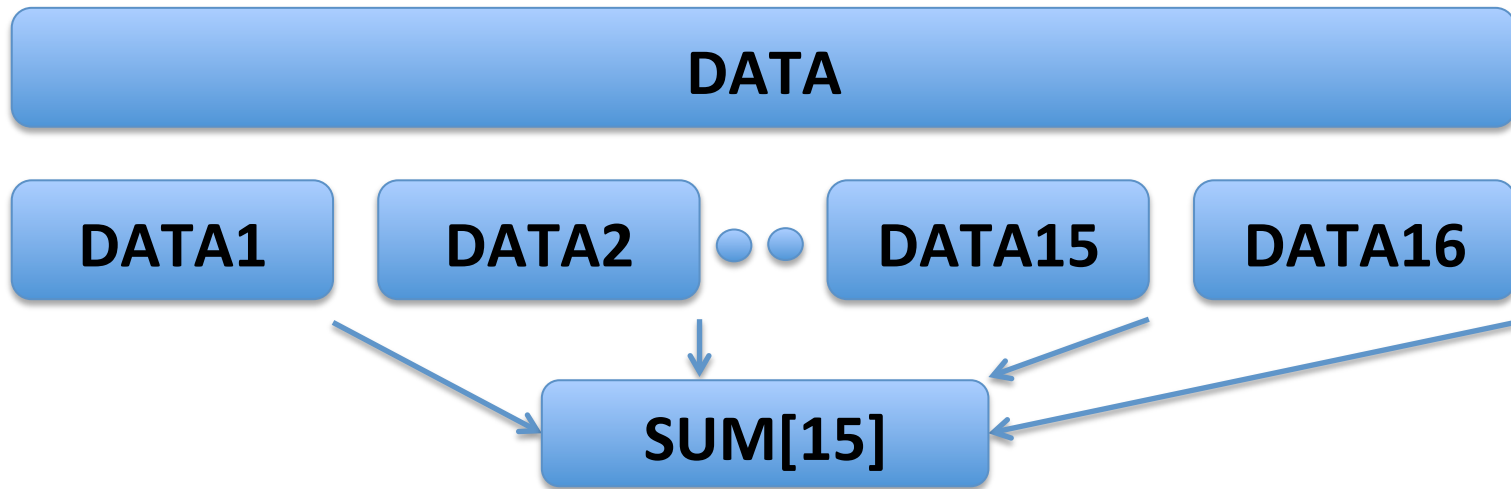
Importance of co-design

- Parallel SCAN



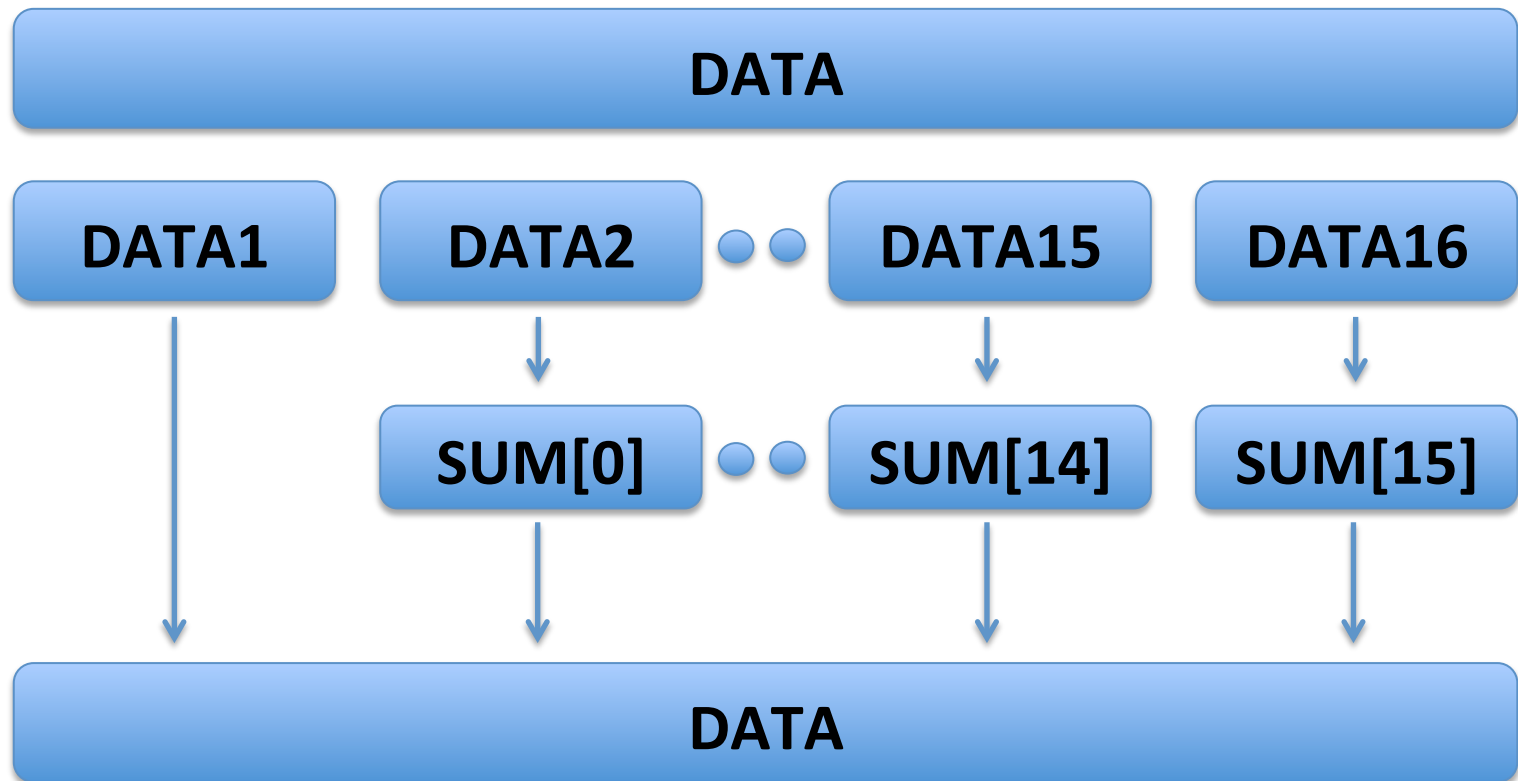
Importance of co-design

- Instead of the naïve, serial SCAN



Importance of co-design

- Instead of the naïve, serial SCAN



Importance of co-design

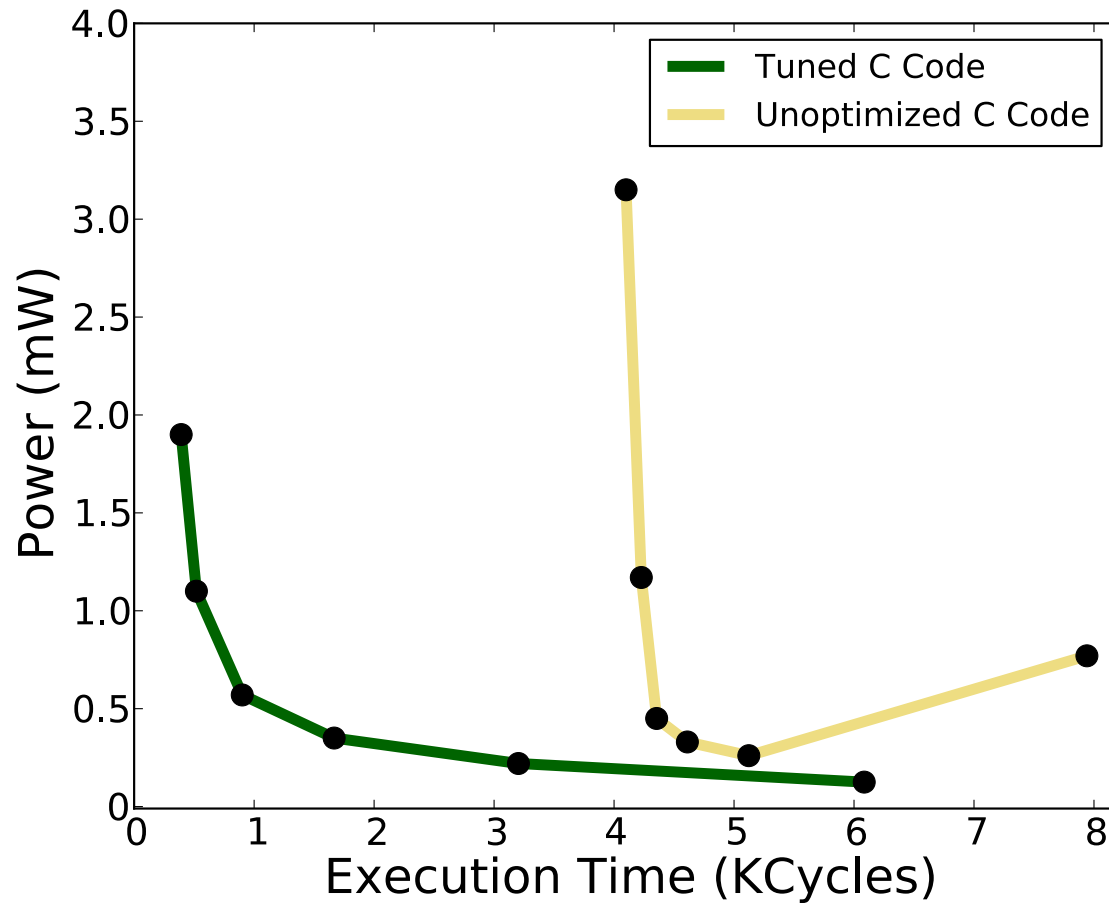
- Exposing parallelism explicitly to HLS tool helps
- Must still tune parallel code

Importance of co-design

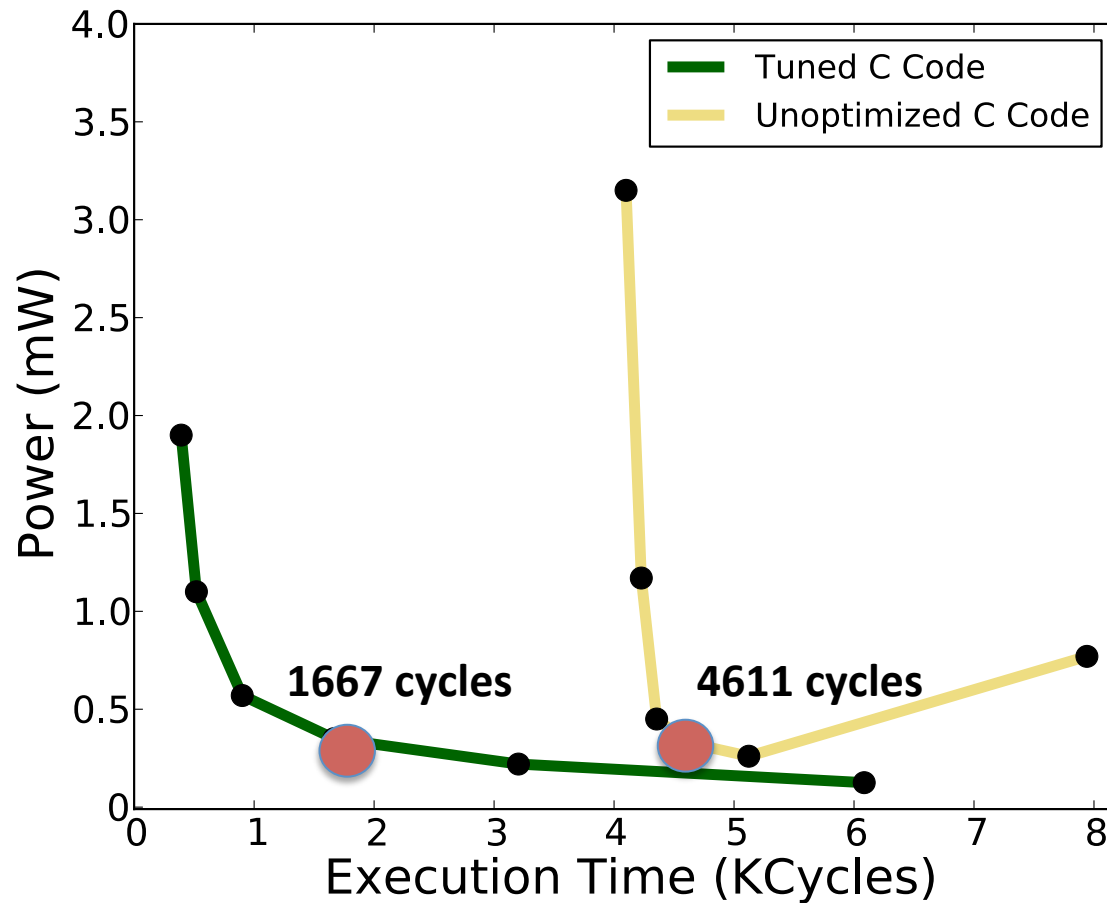
- Exposing parallelism explicitly to HLS tool helps
- Must still tune parallel code
 - Co-optimize data access patterns and *Array partitioning*

How much difference can such tuning make?

A Lot.



What Happened?

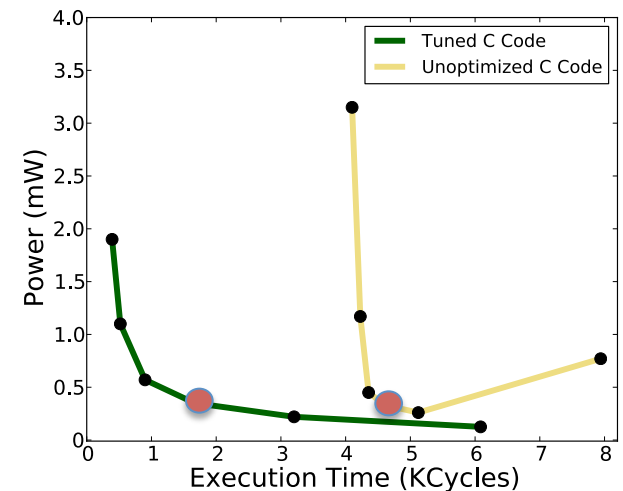


What Happened?

- Same directives
 - Single port SRAMs
 - Arrays factor 4 partition, cyclic
 - Outer Loops pipelined
 - Automatically completely unrolls inner loop

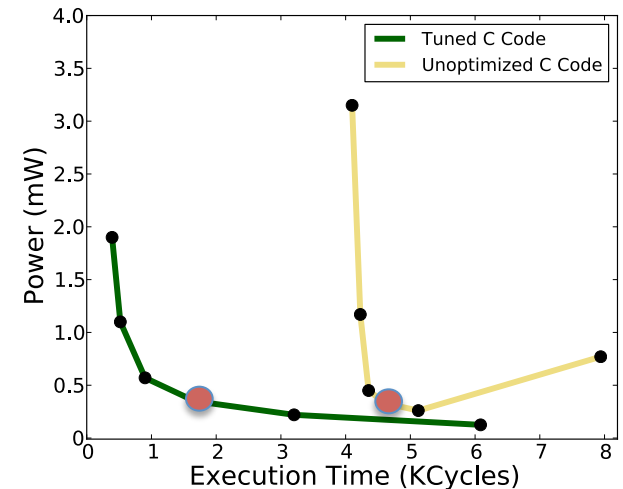
Same *Algorithm*

Different *Implementation*



What Happened?

- “Unoptimized C Code”
 - Pipelining result:
Target II: 1, **Final II: 30**
- “Optimized C Code”
 - Pipelining result:
Target II: 1, **Final II: 8**



What Happened?

Unoptimized C Code

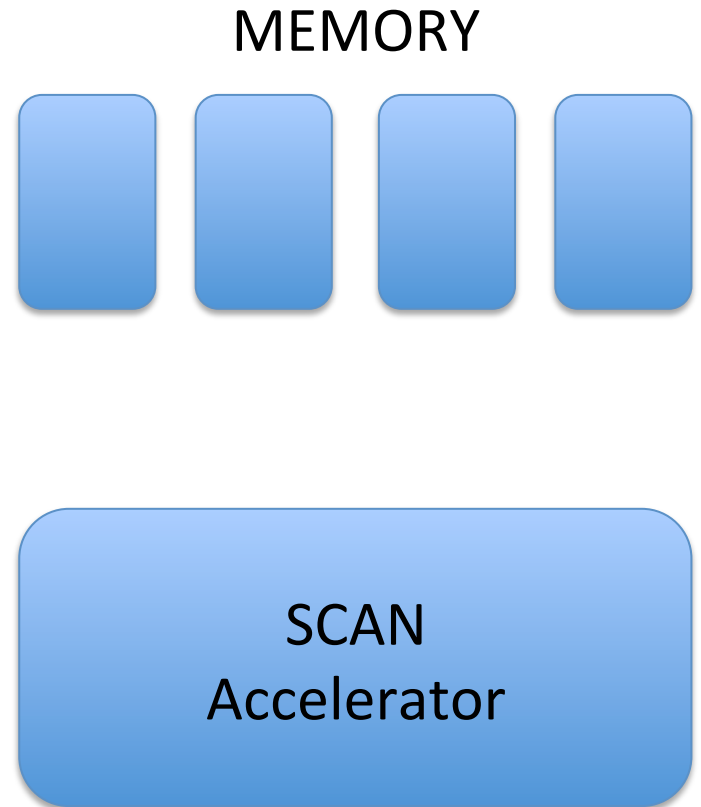
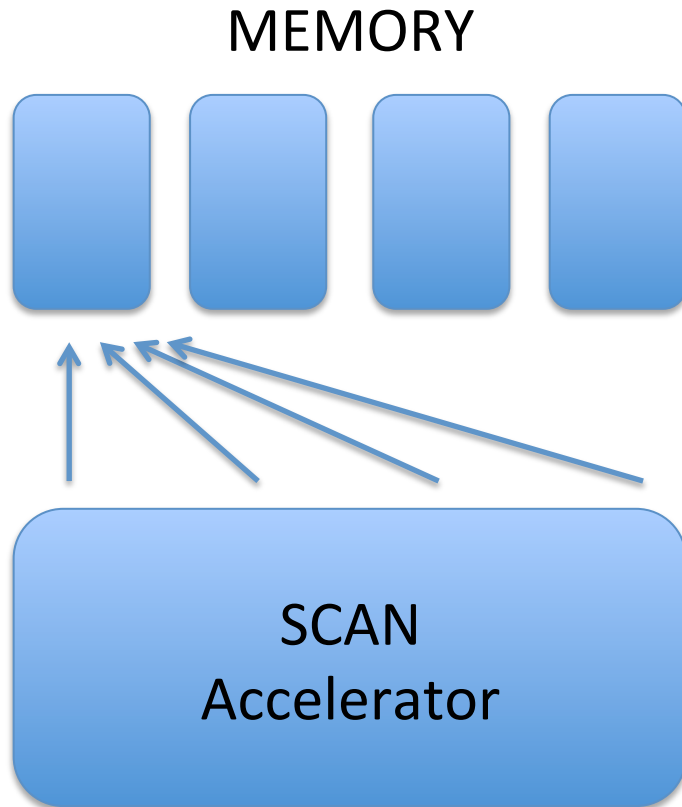
```
for i = 1 : Block
    for radixID : Radix
        bucket[i*Block+radixID ] +=
            bucket[i*Block+ radixID-1];
```

What Happened?

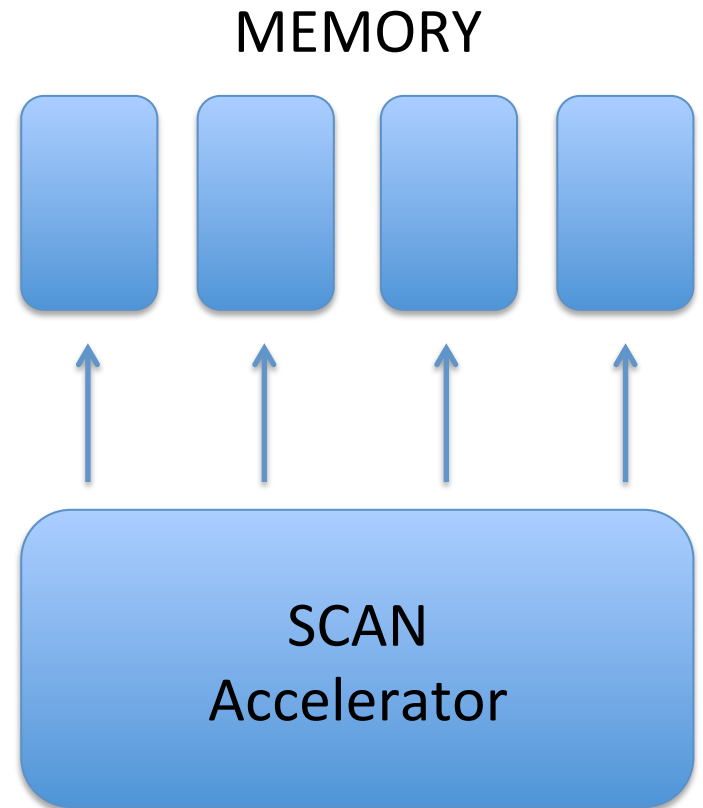
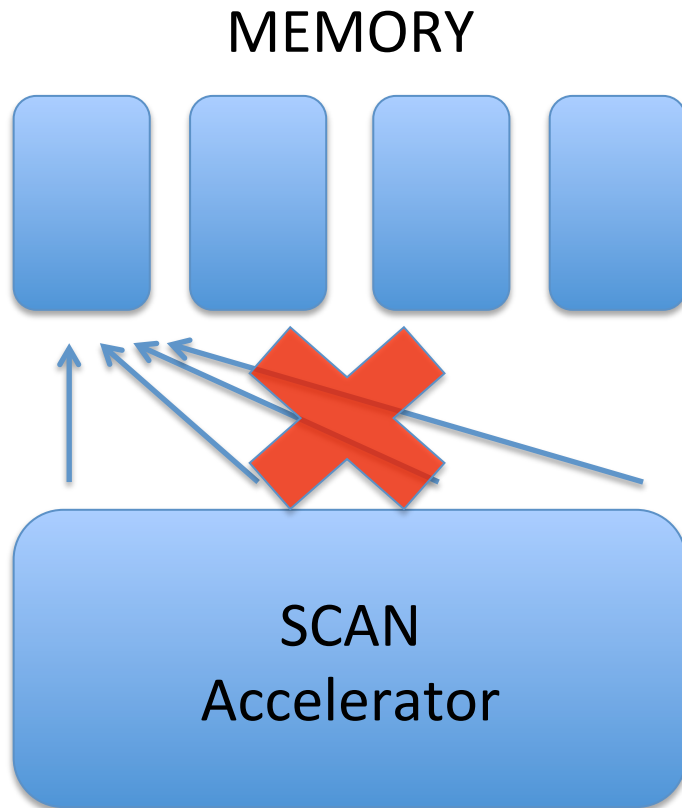
Optimized C Code

```
for radixID : Radix
  for i = 1 : Block
    bucket[i*Block + radixID ] +=
      bucket[i*Block + radixID - 1];
```

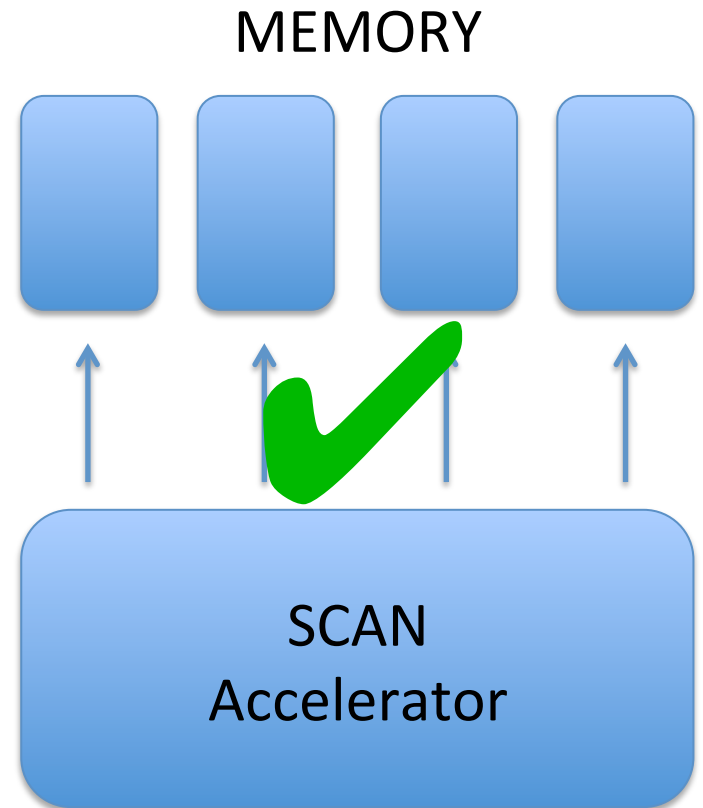
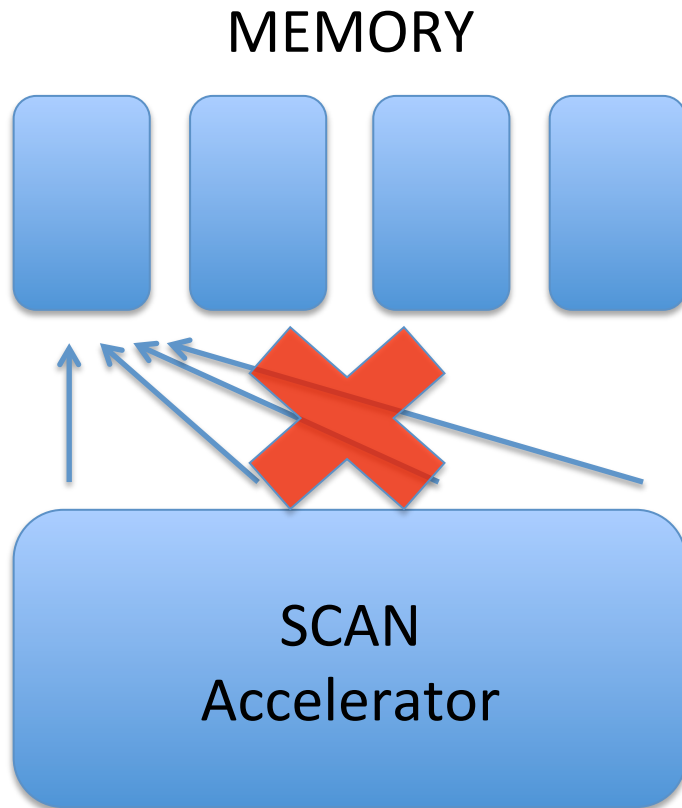
Solution



Solution



Solution

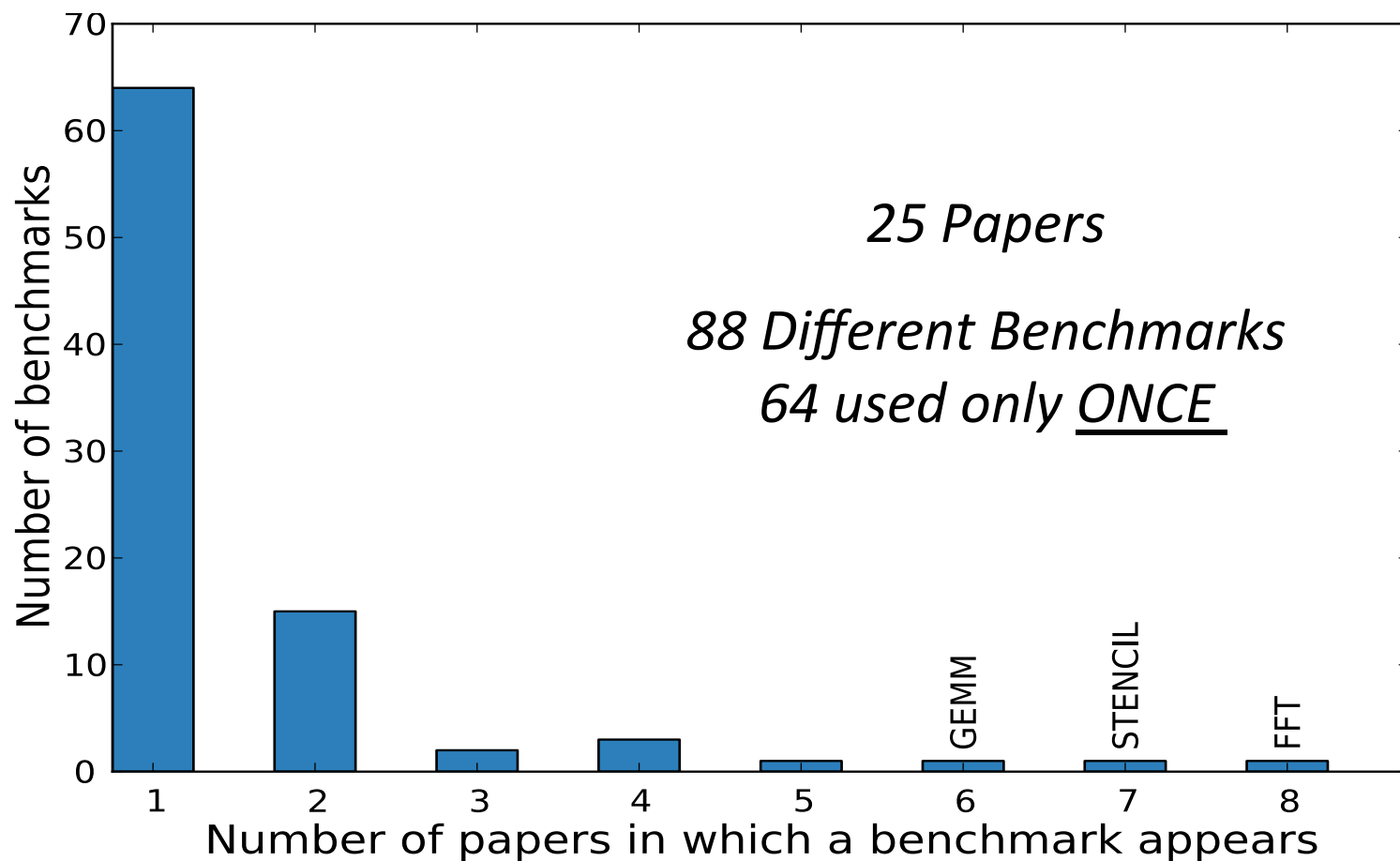


MachSuite:
An Accelerator Benchmark Suite
[IISWC 2014]

MachSuite

- Motivation
 - No commensurability across publications
 - Just saw how much implementation matters
 - It takes a lot of time to write benchmarks...
Doesn't make sense!

Literature Survey



MachSuite

Kernel/Algorithm	Description	Berkeley Dwarf [5]
AES/AES	AES encryption	Combinational logic
BACKPROP/BACKPROP	Neural network training	Unstructured grids
BFS/BULK	Breadth-first search	Graph traversal
BFS/QUEUE	Breadth-first search	Graph traversal
DES/DES	DES encryption	Combinational logic
FFT/SIMPLE	Fast Fourier transform	Spectral methods
FFT/TRANSPOSE	Fast Fourier transform	Spectral methods
GEMM/SIMPLE	Matrix multiplication	Dense linear algebra
GEMM/BLOCKED	Matrix multiplication	Dense linear algebra
KMP/KMP	String matching	Finite state machines
MD/KNN	Molecular dynamics	N-body methods
MD/GRID	Molecular dynamics	N-body methods
NW/NW	DNA alignment	Dynamic programming
SHA/SHA	SHA1 hashing	Combinational logic
SORT/MERGE	Sorting	Map reduce
SORT/RADIX	Sorting	Map reduce
SPMV/CRS	Sparse matrix/vector multiplication	Sparse linear algebra
SPMV/ELLPACK	Sparse matrix/vector multiplication	Sparse linear algebra
STENCIL/SIMPLE	Stencil computation	Structured grids
STENCIL/STENCIL3D	Stencil computation	Structured grids
VITERBI/VITERBI	Hidden Markov model estimation	Graphical models



MachSuite

Kernel/Algorithm	Description	Berkeley Dwarf [5]
AES/AES	AES encryption	Combinational logic
BACKPROP/BACKPROP	Neural network training	Unstructured grids
BFS/BULK	Breadth-first search	Graph traversal
BFS/QUEUE	Breadth-first search	Graph traversal
DES/DES	DES encryption	Combinational logic
FFT/SIMPLE	Fast Fourier transform	Spectral methods
FFT/TRANSPOSE	Fast Fourier transform	Spectral methods
GEMM/SIMPLE	Matrix multiplication	Dense linear algebra
GEMM/BLOCKED	Matrix multiplication	Dense linear algebra
KMP/KMP	String matching	Finite state machines
MD/KNN	Molecular dynamics	N-body methods
MD/GRID	Molecular dynamics	N-body methods
NW/NW	DNA alignment	Dynamic programming
SHA/SHA	SHA1 hashing	Combinational logic
SORT/MERGE	Sorting	Map reduce
SORT/RADIX	Sorting	Map reduce
SPMV/CRS	Sparse matrix/vector multiplication	Sparse linear algebra
SPMV/ELLPACK	Sparse matrix/vector multiplication	Sparse linear algebra
STENCIL/SIMPLE	Stencil computation	Structured grids
STENCIL/STENCIL3D	Stencil computation	Structured grids
VITERBI/VITERBI	Hidden Markov model estimation	Graphical models

Kernel

Algorithms



Why not use existing GPU Benchmark?

- GPUs and Accelerators
different approaches
 - CUDA/OpenCL targets pre-defined hardware (GPU)
 - Accelerators make sense when GPUs **do not**
 - Accelerators need to account for more than TLP
 - FU selection/allocation
 - ILP
 - Arbitrary memory hierarchy

Usability and Characterization

- MachSuite prides itself on usability
 - *MakeFile generates binaries and RTL!*
- Scripts to sweep directives
 - All loops, FUs, Memories are “*tagged*”
- Paper provides rigorous characterization
 - Application Space coverage
 - Locality, parallelism, etc....

Try it Out!

- BETA version of MachSuite
on you're Aladdin USB drive
- Public .git repository
and pre-print paper coming soon

Register online and we'll send you a link!

Design Space Exploration [ISLPED 2013]

What HLS Enables

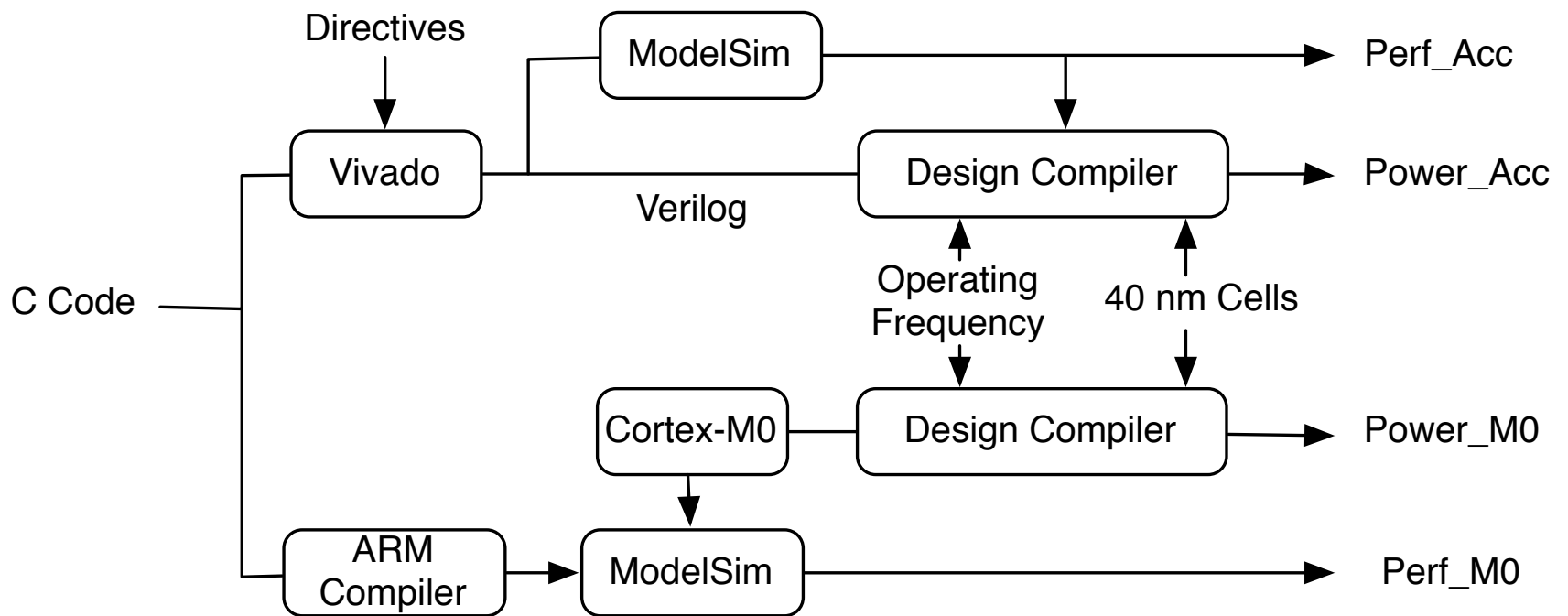
- Quantitative approach to DSE
 - Not as dependent on intuition
- Feedback from compiler as to what to tune
 - As seen before, keep II low
- Leave nothing on the table
 - Entire Design Space
 - *How good is 10x?*

Quantifying Acceleration

ISLPED 2013

- Explore large design spaces
- Analyze Pareto Frontier designs
 - Provide insight on performance scaling
- Compare Accelerator designs to GP-Core

Accelerator Design Flow

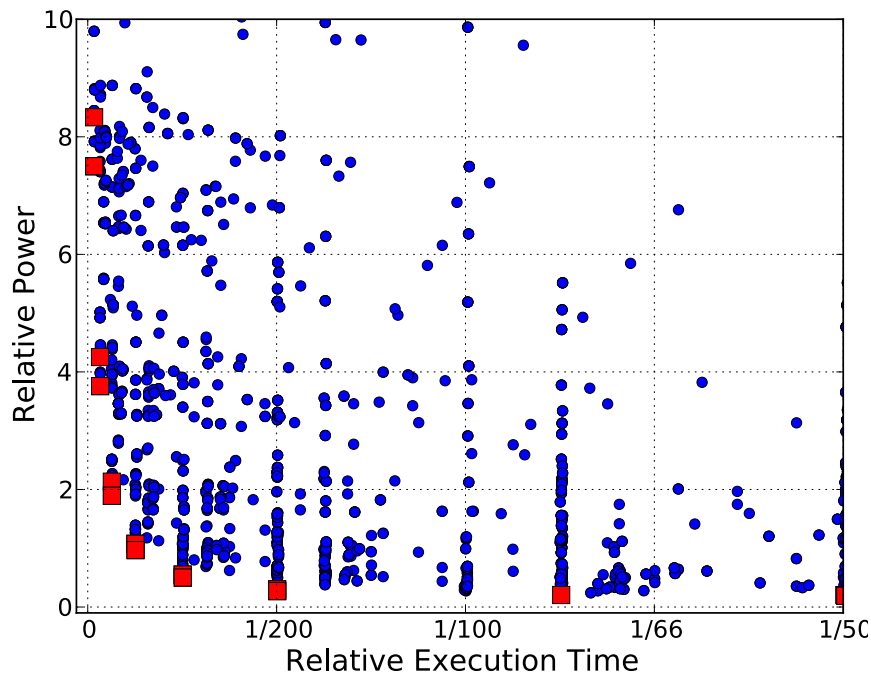


**note here we also swept FU implementations and frequencies*

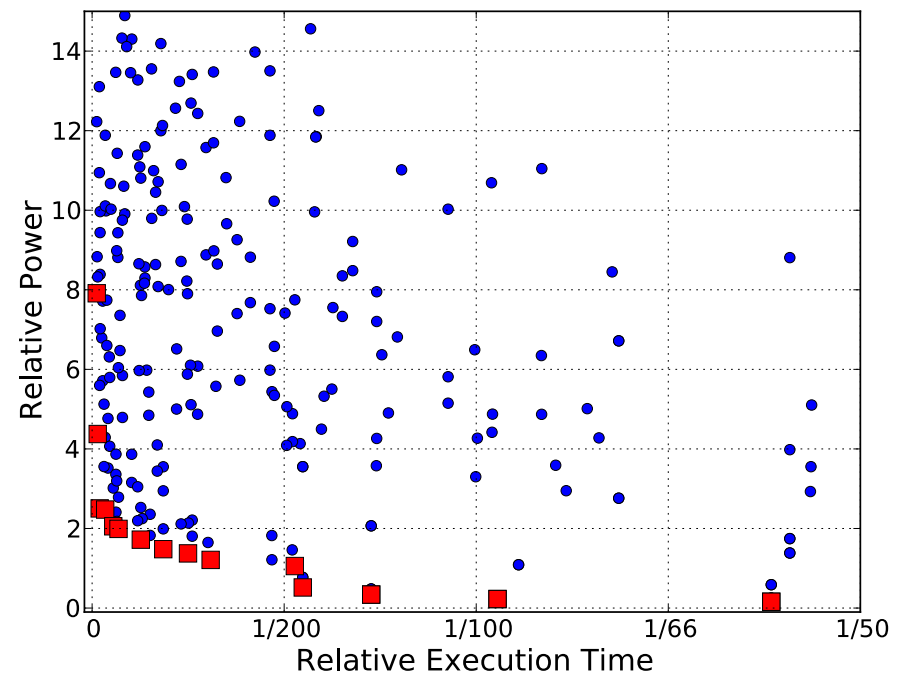
Application Kernels

Simple workloads, huge design space

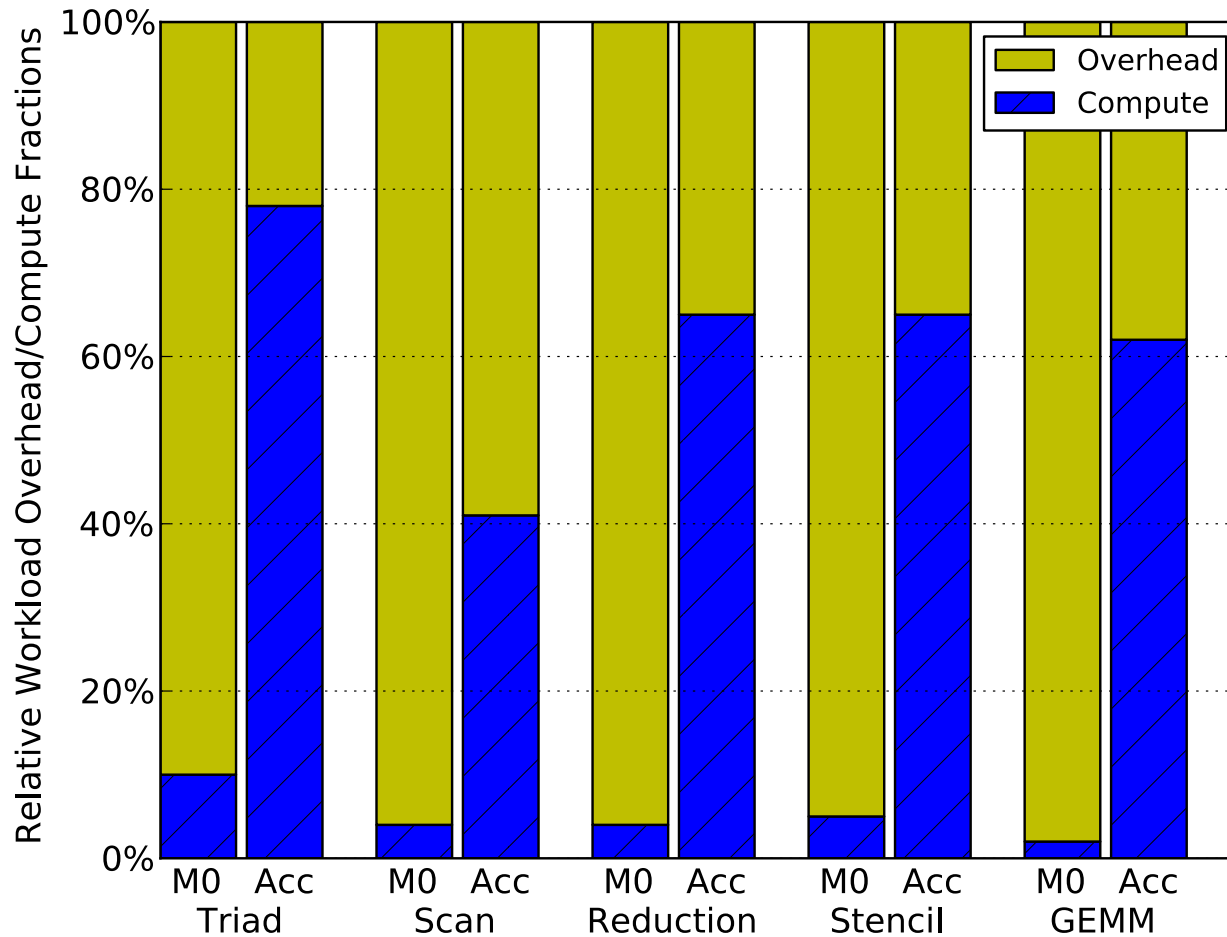
TRIAD



STENCIL

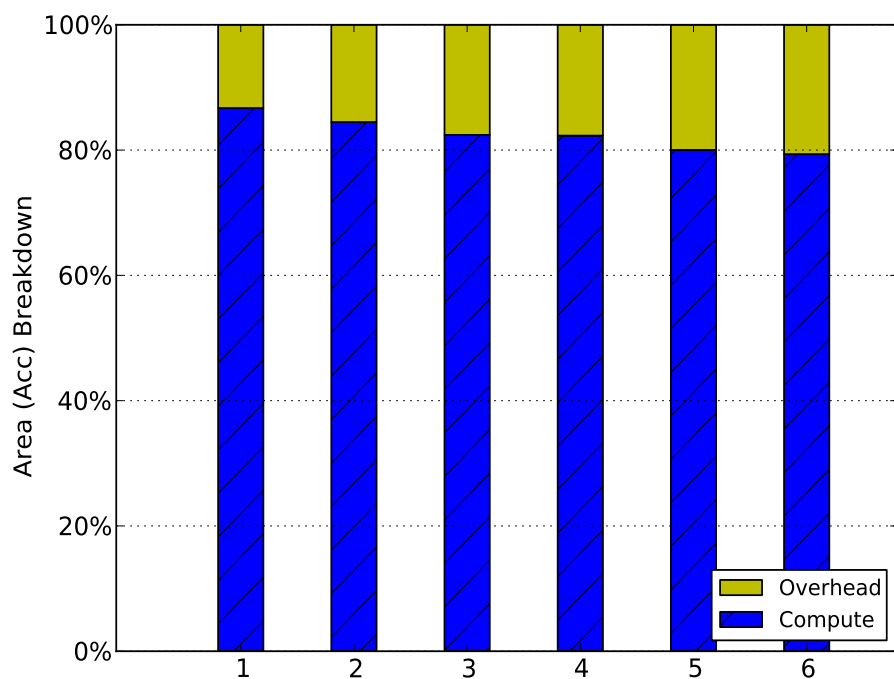


Instruction/Area Breakdown

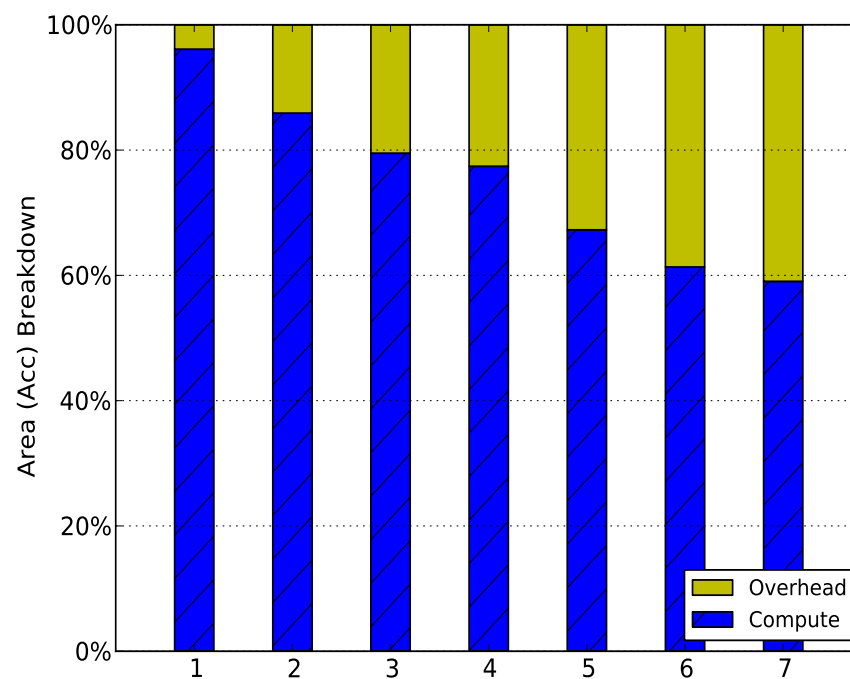


Different Energy Scaling

TRIAD



STENCIL



Conclusions

- HLS can generate high performing Hardware
 - If you do it right...
- Very sensitive to implementation
 - ***MachSuite*** offers benchmarking standard!
- Despite specificity, Huge Space!
- But really... I only care about Pareto Points
 - Energy breakdown example

Conclusions

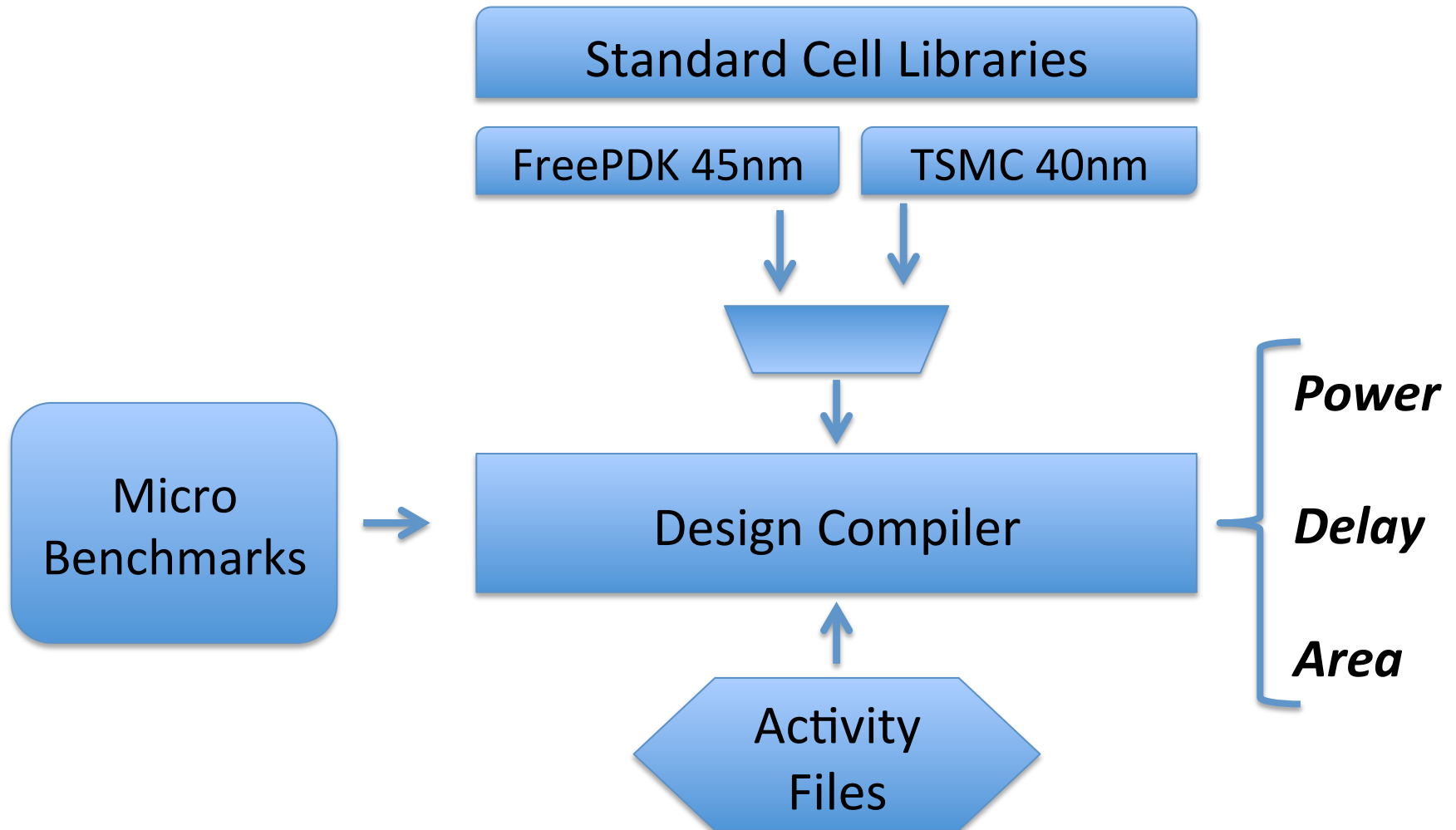
- With ***Aladdin***, you get the Pareto Points **100x** faster!
- Avoid ***cumbersome*** C code tuning
 - More powerful analysis, less restrictions

MachSuite (backup)

Kernel/Algorithm	Description	Berkeley Dwarf [5]
AES/AES	AES encryption	Combinational logic
BACKPROP/BACKPROP	Neural network training	Unstructured grids
BFS/BULK	Breadth-first search	Graph traversal
BFS/QUEUE	Breadth-first search	Graph traversal
DES/DES	DES encryption	Combinational logic
FFT/SIMPLE	Fast Fourier transform	Spectral methods
FFT/TRANSPOSE	Fast Fourier transform	Spectral methods
GEMM/SIMPLE	Matrix multiplication	Dense linear algebra
GEMM/BLOCKED	Matrix multiplication	Dense linear algebra
KMP/KMP	String matching	Finite state machines
MD/KNN	Molecular dynamics	N-body methods
MD/GRID	Molecular dynamics	N-body methods
NW/NW	DNA alignment	Dynamic programming
SHA/SHA	SHA1 hashing	Combinational logic
SORT/MERGE	Sorting	Map reduce
SORT/RADIX	Sorting	Map reduce
SPMV/CRS	Sparse matrix/vector multiplication	Sparse linear algebra
SPMV/ELLPACK	Sparse matrix/vector multiplication	Sparse linear algebra
STENCIL/SIMPLE	Stencil computation	Structured grids
STENCIL/STENCIL3D	Stencil computation	Structured grids
VITERBI/VITERBI	Hidden Markov model estimation	Graphical models



Power Model(backup)



LUMOS

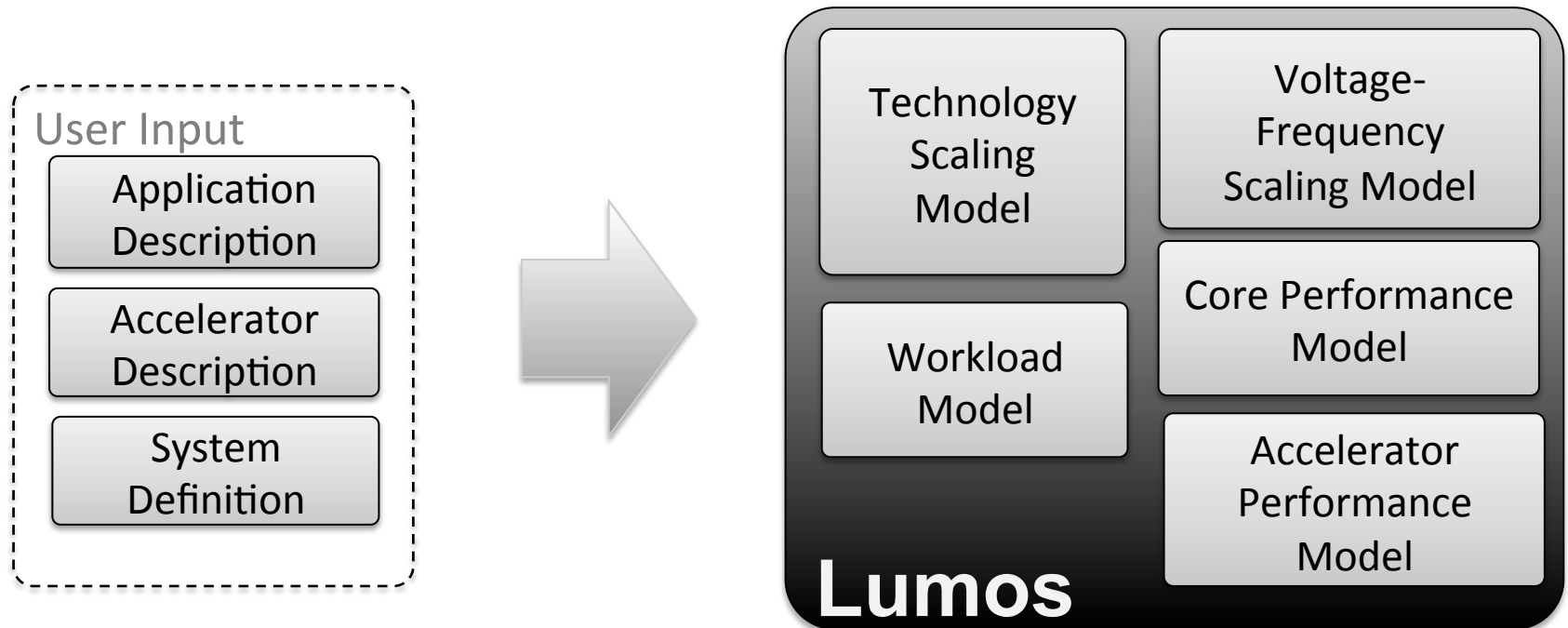
A Framework with Analytical Models for
Heterogeneous Architectures

Liang Wang, and Kevin Skadron
(University of Virginia)



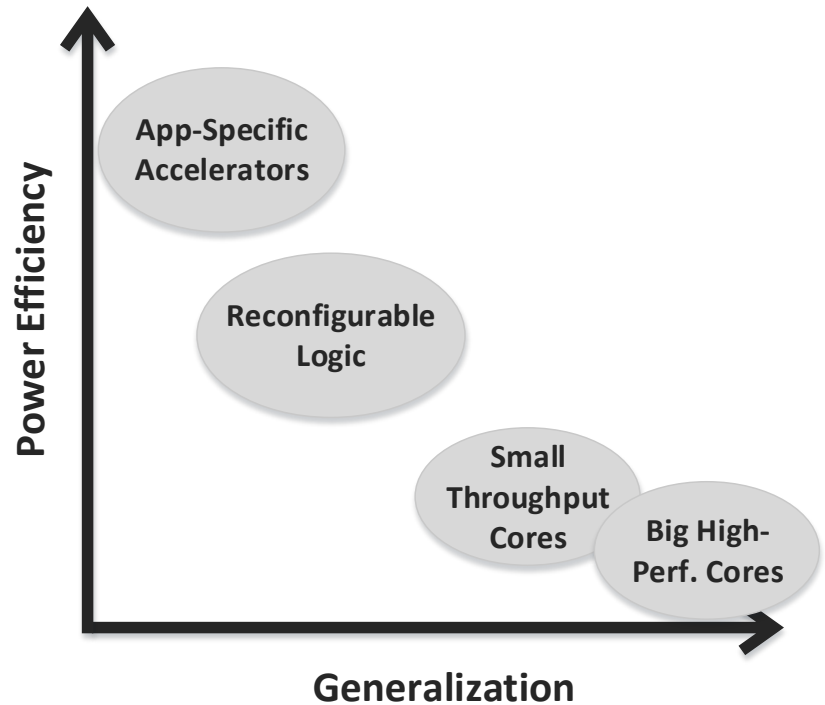
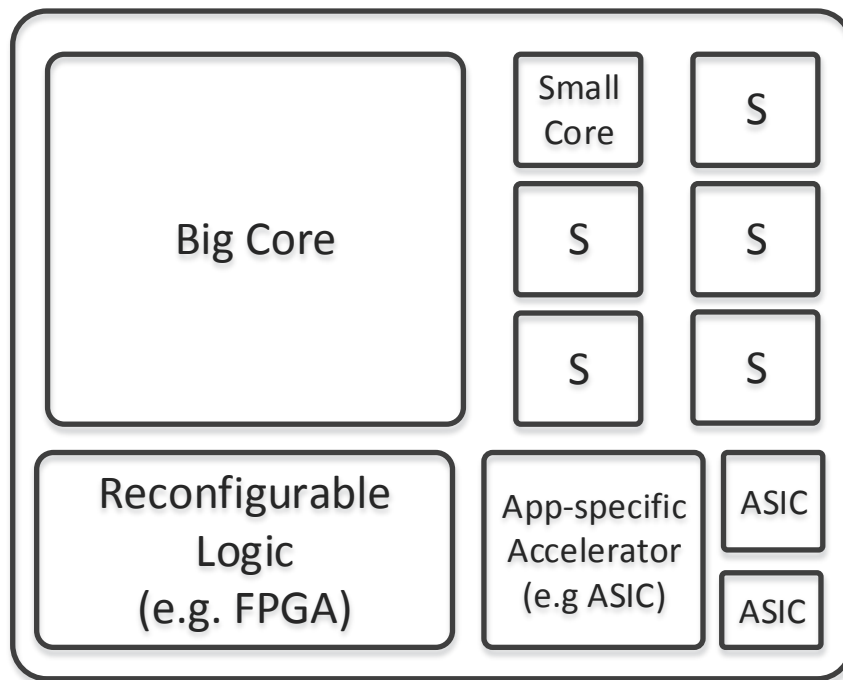
What is LUMOS

- A set of first-order analytical models targeting heterogeneous system including: conventional cores and accelerators



Why LUMOS

- Heterogeneous systems with accelerators have huge design space



Why LUMOS (cont.)

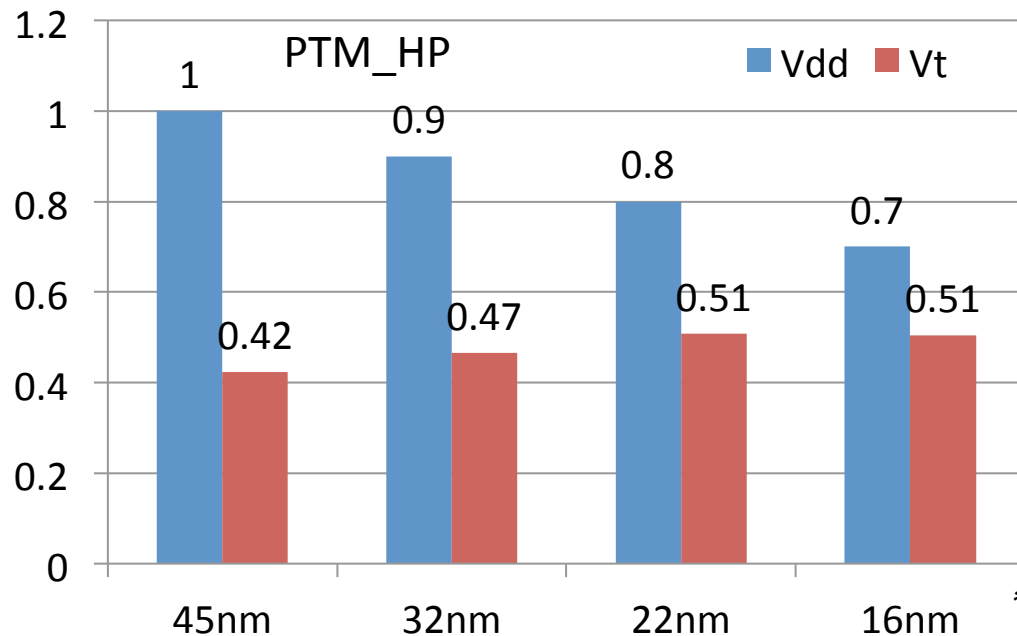
- Quickly explore answer for:
 - Big cores vs. many small cores
 - Are massive small cores (potentially at lower Vdd.) better than a couple of big cores?
 - Conventional cores vs. hardware accelerators
 - Are hardware accelerators actually needed?
 - Reconfigurable vs. app-specific
 - In which case, reconfigurable logic is favored over app-specific accelerators, and vice versa.

Agenda

- Model Details
- Framework Interfaces
- Case Studies
- Extension to LUMOS

Technology Scaling Model

- Based on Predictive Technology Model (PTM)*
 - From 45nm to 16nm
 - Both High-Performance (HP) and Low-Power (LP) variants



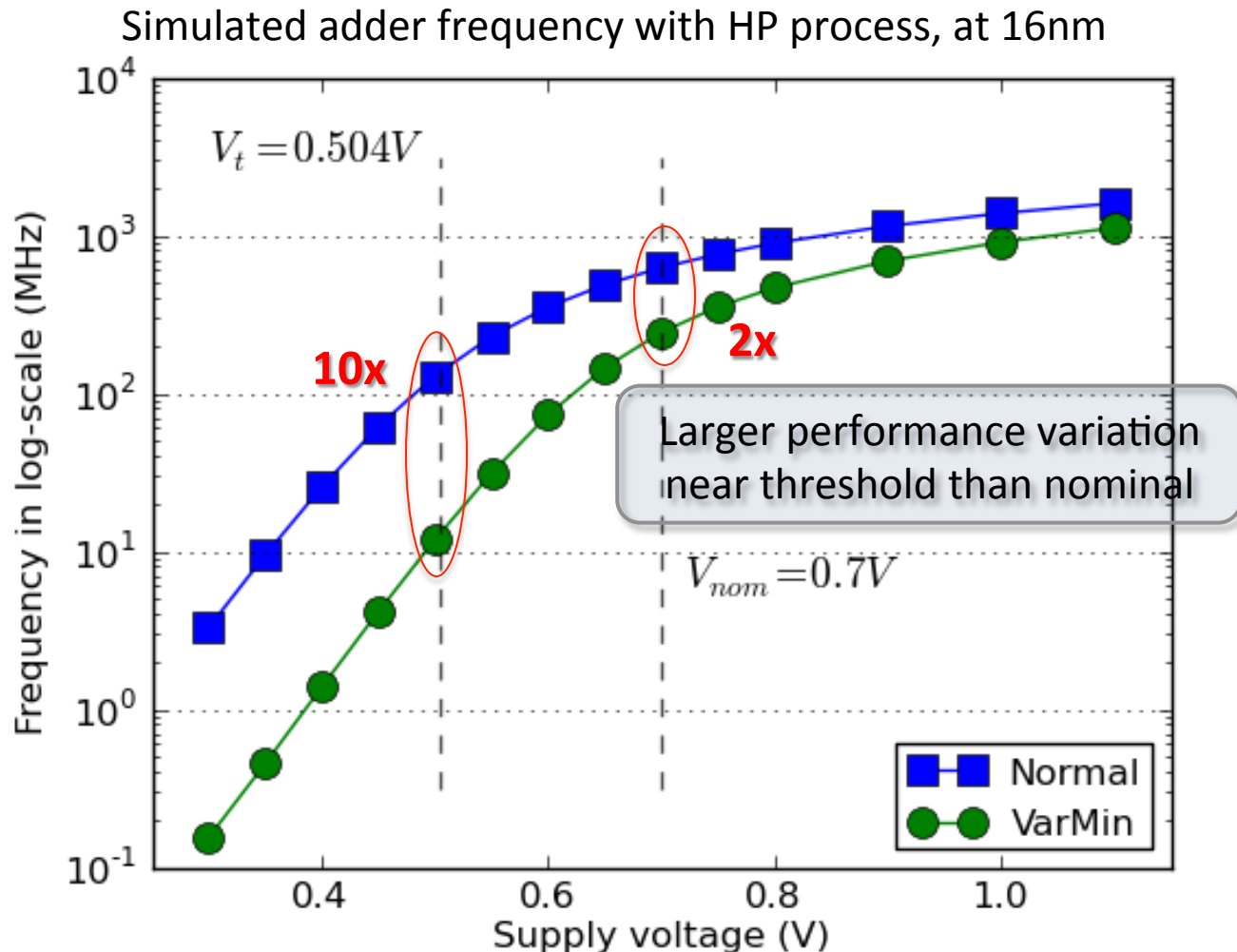
*[B. H. Calhoun et al, at ISCAS 2009]

Voltage-Frequency Scaling Model

- Simulate an 32-bit ripple carry adder with Predictive Technology Model (PTM)*, from 45nm to 16nm
 - Sweep voltage from 0.3V to 1.1V
- Using 3- σ in Monte-Carlo simulations to determine the worst frequency due to process variations

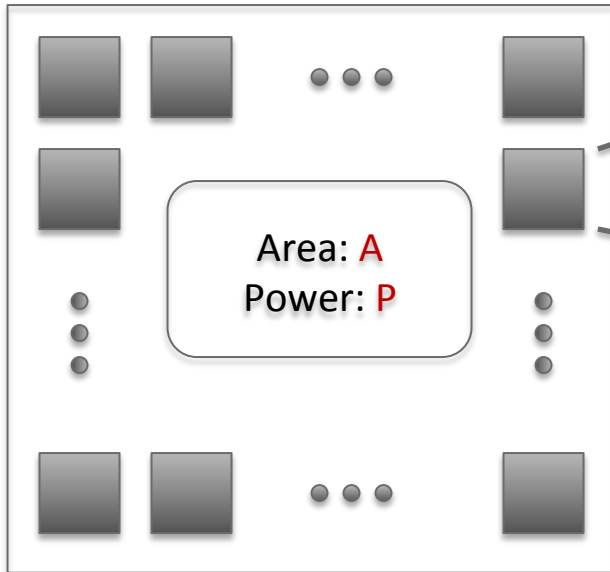
**[B. H. Calhoun et al, at ISCAS 2009]*

Impact of Process Variations

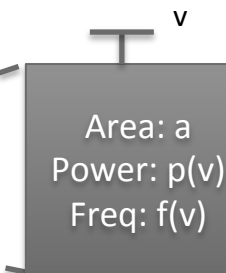


Core Performance Model

Symmetric Multi-core System



A Single core



Dynamic Power	$\propto f \cdot v^2$
Static Power	Fitted to circuit sim.
Frequency	Fitted to circuit sim.

Number of
active cores

$$n = \min\left(\frac{P}{p(v)}, \frac{A}{a}\right)$$

Use Amdahl's Law to model multi-core speedup in terms of throughput.

$$speedup = \frac{1}{\frac{1 - \rho}{perf(V_{max})} + \frac{\rho}{n \cdot perf(V)}}$$

ρ is the parallel fraction of a application

Accelerator Performance Model

Accelerator Parameters*

$A_{\downarrow 0}$: Area of a single baseline core

A : Area of an accelerator

μ : Relative performance of an accelerator with the size of $A_{\downarrow 0}$

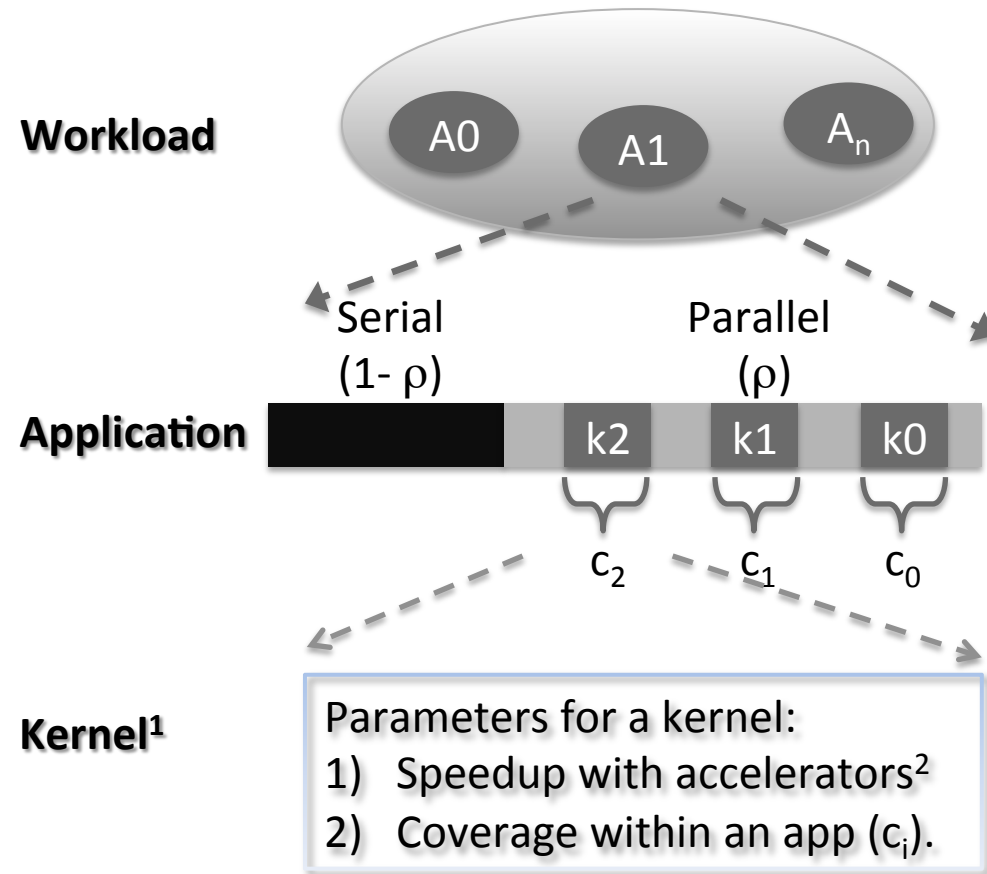
ϕ : Relative power of an accelerator with the size of $A_{\downarrow 0}$

* Similar to [E. S. Chung, et. al. , MICRO10]

Accelerator Modeling

$$perf = \mu \cdot A / A_{\downarrow 0} \quad power = \phi \cdot A / A_{\downarrow 0}$$

Workload model



Execution Model

Serial

Executed by a conventional core

Parallel

Executed by conventional cores in parallel

Kernel

Executed by the best available accelerators (e.g. app-specific will be favored over reconfigurable), otherwise executed by conventional cores in parallel

¹Kernels are always within parallel portion of an application

²Accelerator's speedup is proportional to its area

Agenda

- Model Details
- Framework Interfaces
- Case Studies
- Extension to LUMOS

Application Description

- Encoded in XML format
- Each application is characterized by:
 - Parallel fraction (f_parallel)
 - Kernels' relative execution time (cov)

```
<workload>
  <app name="sigproc">
    <f_parallel>0.7</f_parallel>
    <kernel_config>
      <kernel name="sort" cov="0.3"/>
      <kernel name="fft" cov="0.35"/>
    </kernel_config>
  </app>
  ...
</workload>
```

Accelerator Description

- Encoded in XML format, indexed by kernels
- Accelerator devices are characterized by:
 - Relative performance (μ)
 - Relative power (ϕ)
 - Scaled to the same size of a baseline core

```
<kernels>
  <kernel name="fft">
    <fpga mu="2.4", phi="1.5"/>
    <asic mu="10.2" phi="2.3"/>
  </kernel>
  ...
</kernels>
```

System Definition

- Command-line parameter (could be migrated to XML in the future)
- Chip-level design constraints
 - Area
 - TDP
 - Technology node
- Core parameters
 - Core type (e.g. IOCore, O3Core)
- Accelerator related parameters
 - Area allocation
 - Type of ASIC accelerators

System Def.

```
area=100  
power=60  
tech=16  
Core_type=IOCore  
fpga=10%  
asic_sort=10%  
asic_fft=5%
```

Agenda

- Model Details
- Framework Interfaces
- Case Studies
 - Case I: Conventional Cores
 - Case II: Optimal Accelerator Allocation
 - Case III: Benefit of App-Specific Accelerators
- Extension to LUMOS

Case Study I: Conventional Cores

- Baseline (Niagara2-like IOCore)

Frequency (GHz)	Dynamic Power (W)	Leakage Power (W)	Area (mm ²)
4.20	6.14	1.06	7.65

** Derived from McPAT, scaled to 45nm*

- Chip-level budgets

System Type	Reference	Area (mm ²) [†]	TDP (W) [‡]	Max # of Base Cores at 45nm*
Desktop	Core2 Quad	107	33	14
Small Server	Xeon	130	65	17
Large Server	SPARC T4	200	120	26

[†] Assume 50% budget for core-only components

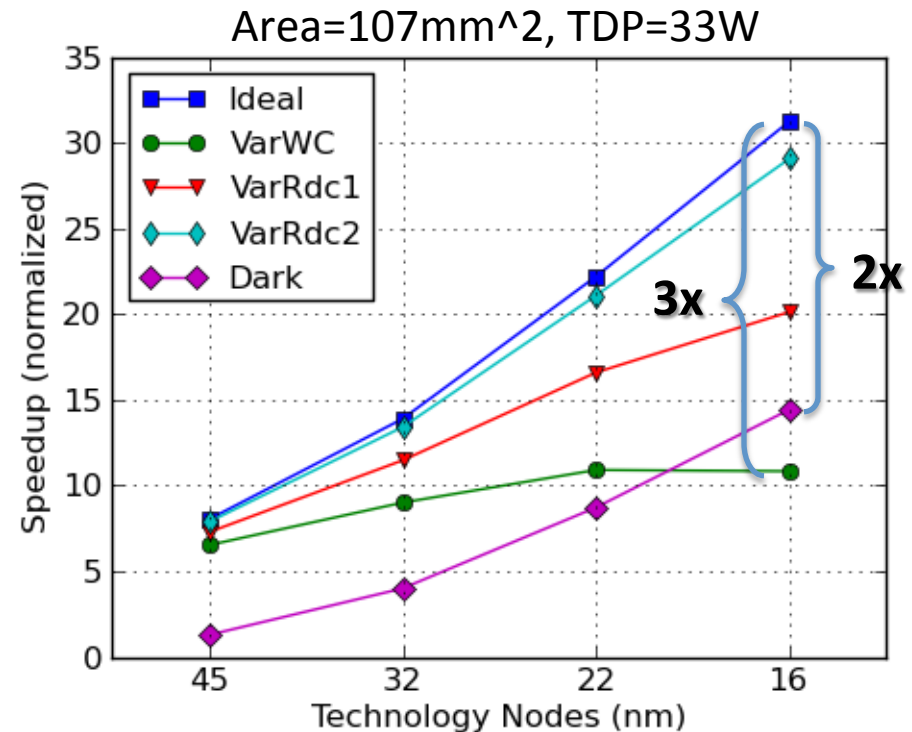
* Calculated by dividing base core area from total core area budget

Case Study I: Assumptions

- Sufficient bandwidth
- Highly parallel application with $\rho=0.99$. (e.g. HPC)
- Speedup normalized to a single baseline core
- When accelerating parallel part of an application, LUMOS searches for the best operating supply voltage achieving the highest throughput

Case Study I: Dark Si. vs. Dim Si.

- Alternative cases
 - Ideal: ignore variation impacts maximize freq subject to TDP
 - VarWC: worst-case variation penalty
 - VarRdc1: 50% of WC penalty
 - VarRdc2: 10% of WC penalty



- 2x speedup if ignoring variation impacts
- 3x slowdown with worst-case frequency

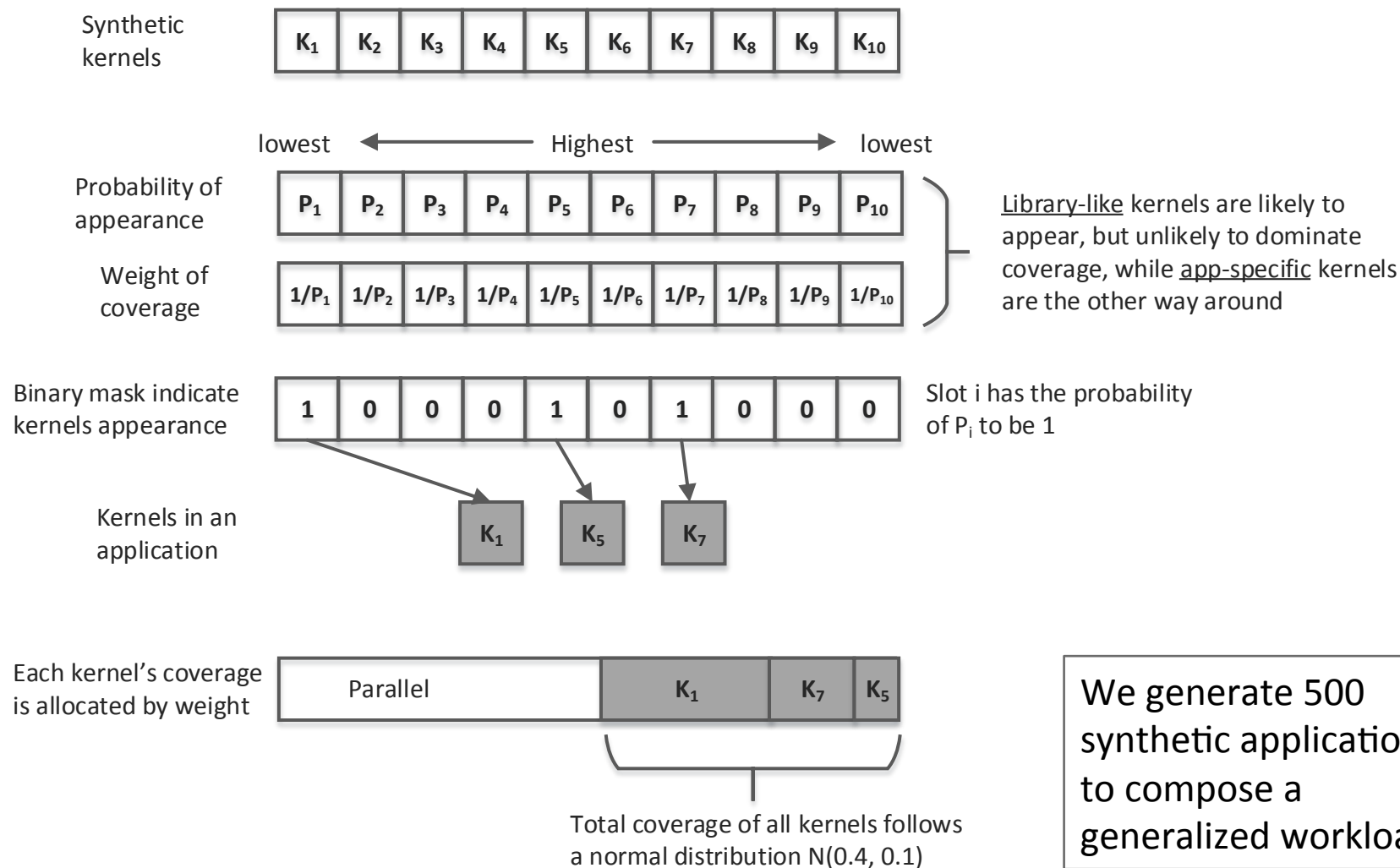
Case Study II: Accelerator Setup

- Synthetic kernels and accelerators
 - Reconfigurable accelerator relative performance (μ) ranging from 20x to 60x
 - Assume a fixed ratio (5x) of App-specific over reconfigurable accelerators

Relative Performance (μ) of Synthetic Accelerators

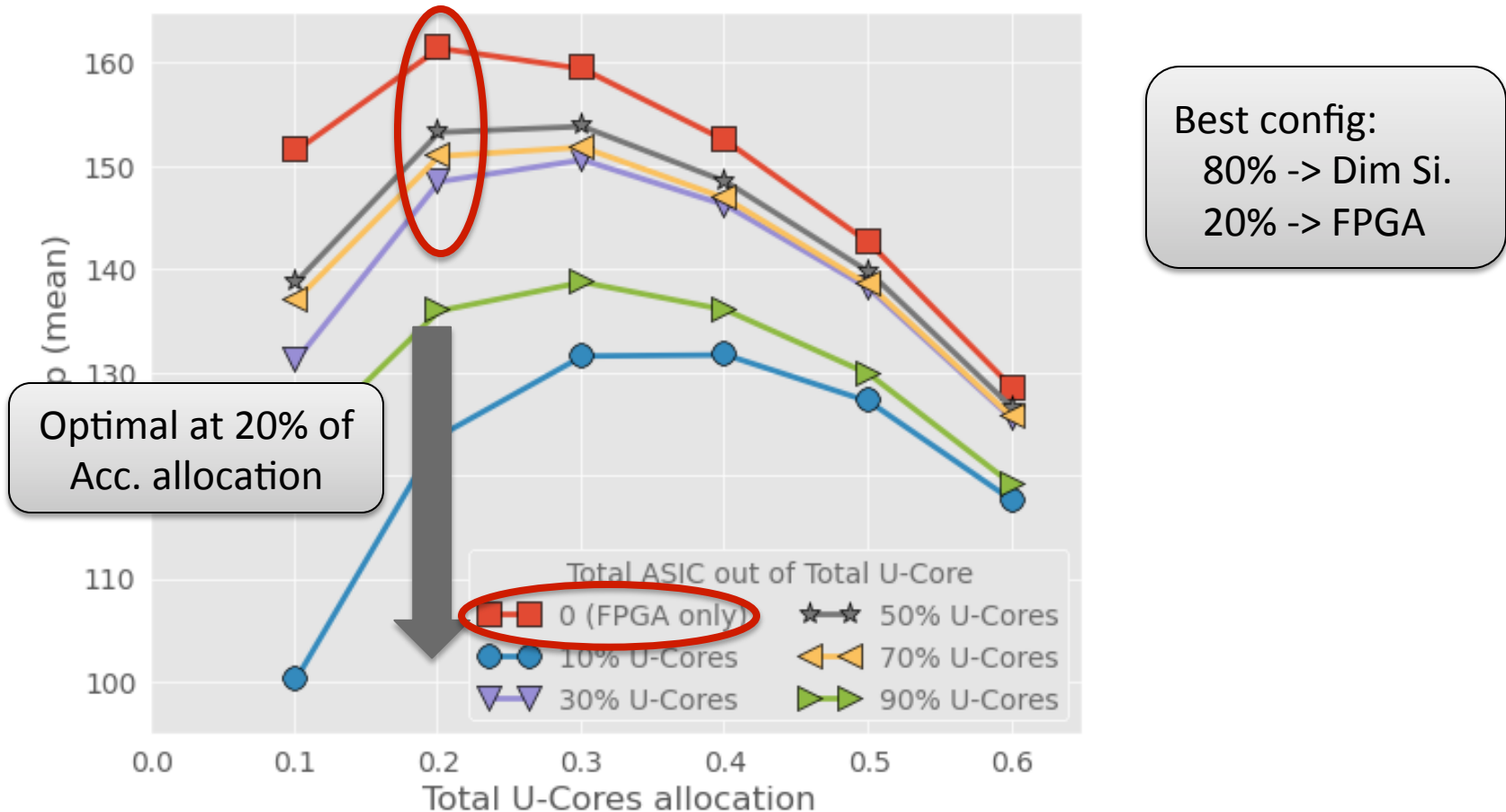
	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10
RL	20	24	28	33	37	42	46	51	55	60
ASIC	100	120	140	165	185	210	240	255	275	300

Case Study II: Synthetic Applications



Case Study II: Optimal Accl. Configuration

- 16nm, ideal parallelism for all applications



Why RL Only?

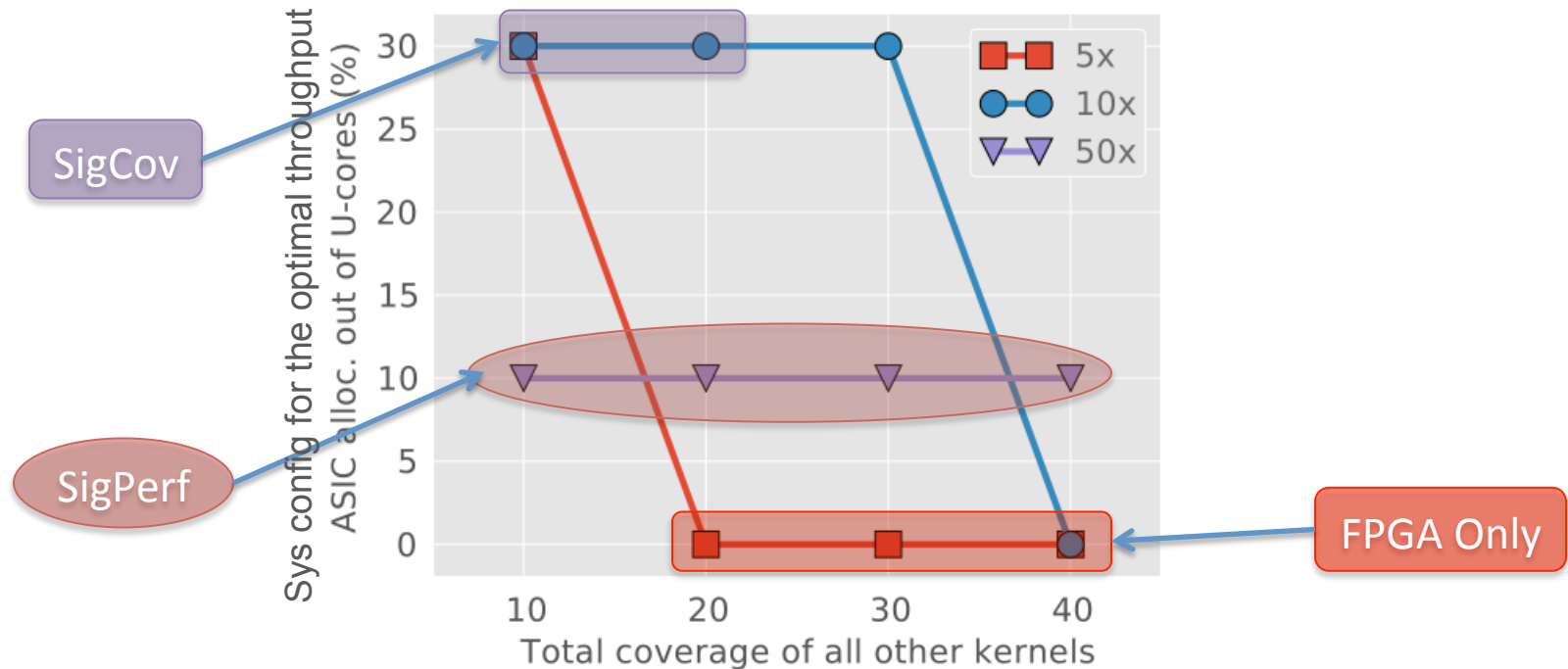
- RL is almost “as powerful as” ASIC
 - Assume 5x performance ratio between RL and ASIC
 - Assume linear performance scaling to allocated area
 - A large RL allocation is justified by the high utilization achievable across multiple kernels
- A single ASIC is too expensive due to small average coverage of the target kernel
 - Small coverage within applications (library-call kernels)
 - Rare presence among applications (app-specific kernels)

Case Study III: Benefit of ASIC accelerators

- Alternative ratio between RL and ASIC
 - 5x, 10x, 50x
- Add one more synthetic kernel
 - RL has a relative performance $\mu=40$
 - Keep its coverage at 20% across applications
- Total coverage for all other kernels is fixed across applications.

Case Study III: Benefit of ASIC accelerators

- Dedicated ASIC is beneficial, when:
 - Significant coverage of the targeted kernel (SigCov)
 - Significant performance gap between ASIC and FPGA (SigPerf)



20% coverage for the targeted kernel

Agenda

- Model Details
- Framework Interfaces
- Case Studies
- Extension to LUMOS

Extension to LUMOS

- New technology model (FinFET, T-FET, etc.)
- Detailed performance model (memory impacts)
- Realistic accelerator and workload characteristics

LUMOS Release

- LUMOS is available at:
 - liangwang.github.io/lumos
- Related publication
 - L. Wang, K. Skadron, “Implications of the Power Wall: Dim Cores and Reconfigurable Logic.” IEEE MICRO, Sep.-Oct. 2013
 - L. Wang, K. Skadron, “Dark vs. Dim Silicon and Near-Threshold Computing Extended Results.” TR, UVA-CS-2013-01, University of Virginia.

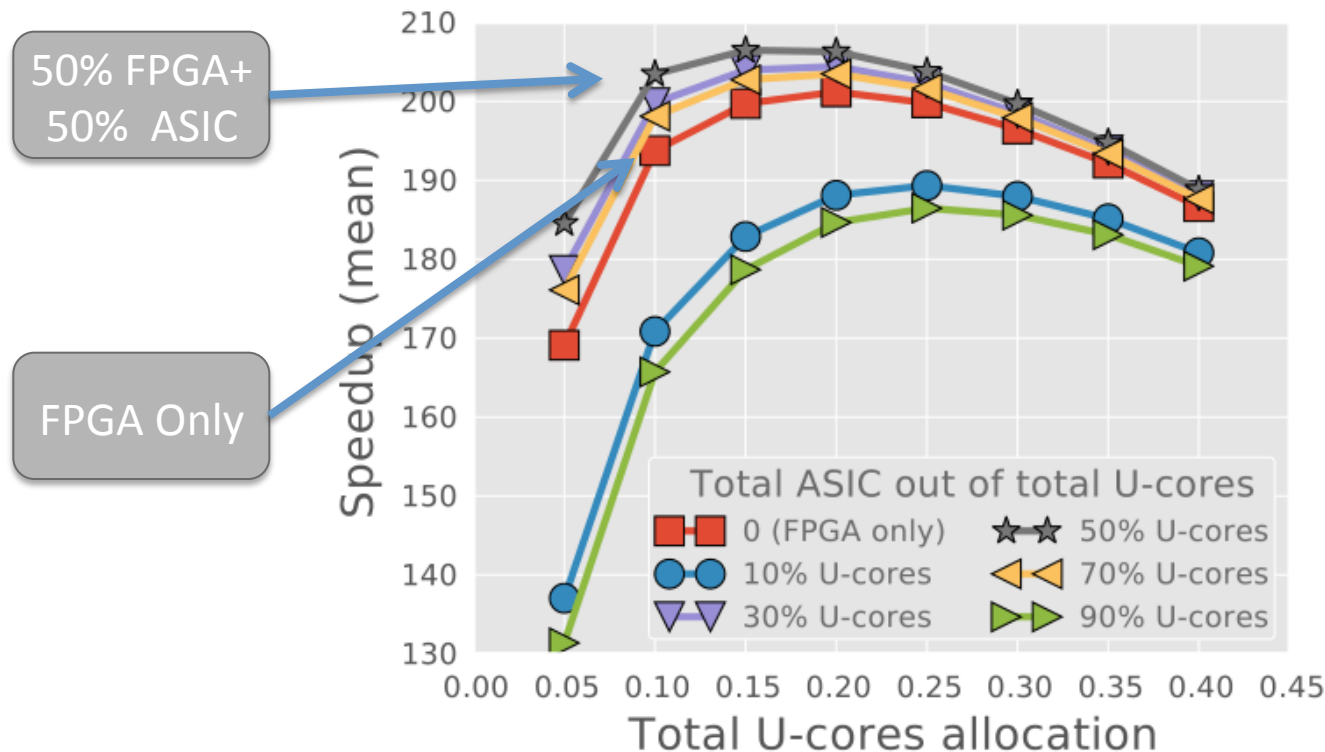
BACKUP

U-core area for Case III

Ratio \ cov.	10%	20%	30%	40%
5x	10	15	20	20
10x	10	15	20	20
50x	10	10	15	20

Benefit of ASIC Accelerators (detailed)

- An ASIC accelerator is beneficial only when its targeted kernel covers significant portion of the application (e.g. more than the total coverage of all other kernels)



Kernel coverage: 30% targeted kernel, 10% all other kernels

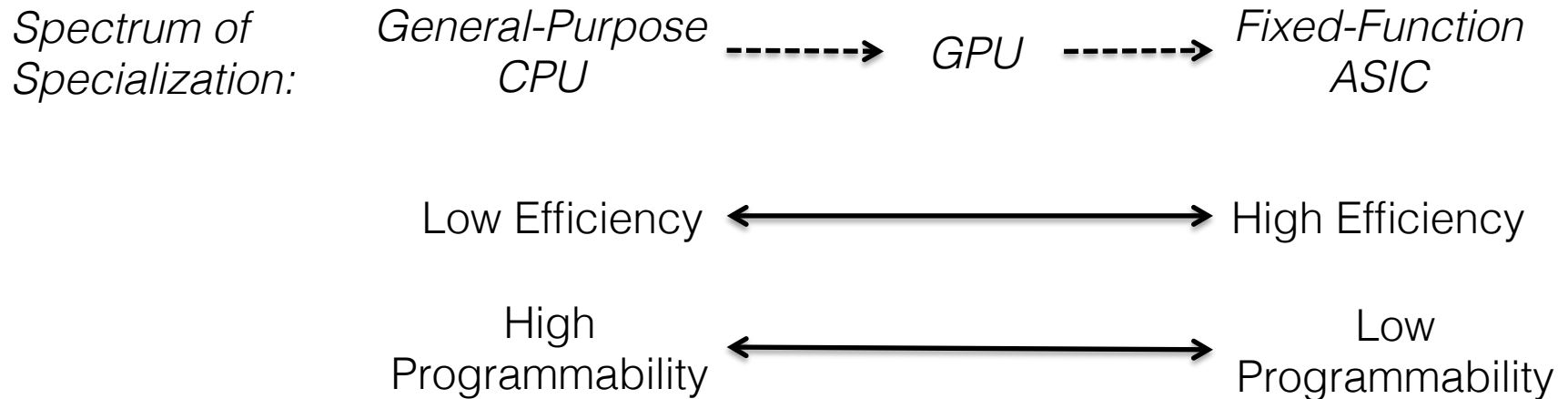
WIICA

Workload ISA-Independent Characterization for Applications

Yakun Sophia Shao, Emma Wang,
Gu-Yeon Wei, David Brooks
Harvard University



Specialized architectures are decoupled from legacy ISAs.



Specialized architectures are decoupled from legacy ISAs.

*Spectrum of
Specialization:*

*General-Purpose
CPU*



GPU



*Fixed-Function
ASIC*

Low Efficiency



High Efficiency

High
Programmability



Low
Programmability

Tied to a
Specific ISA



No ISA

Performance-Counter Based Workload Characterization

- Metrics
 - IPC
 - Cache miss rates
 - Branch mis-prediction rates
 - ...
- Microarchitecture-dependent
 - What if there is a bigger cache/a better branch predictor?
 - Not program intrinsic characteristics

ISA impacts program behaviors.

Stack Overhead

- Limited Registers
- Additional Load/Store

ISA impacts program behaviors.

Stack Overhead

- Limited Registers
- Additional Load/Store

Complex Operations

- Memory Operands
- Vector Operations

ISA impacts program behaviors.

Stack Overhead

- Limited Registers
- Additional Load/Store

Complex Operations

- Memory Operands
- Vector Operations

Calling Conventions

WIICA Summary

Goal:

- An analysis tool to characterize workloads ISA-independent characteristics for specialized architectures

WIICA Summary

Goal:

- An analysis tool to characterize workloads ISA-Independent characteristics for specialized architectures

Methods:

- Leverage compiler's intermediate representation (IR)
- Categorize characteristics into compute, memory, and control

WIICA Summary

Goal:

- An analysis tool to characterize workloads ISA-Independent characteristics for specialized architectures

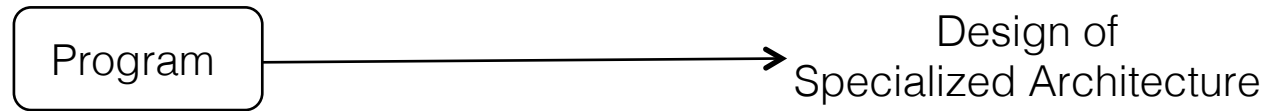
Methods:

- Leverage compiler's intermediate representation (IR)
- Categorize characteristics into compute, memory, and control

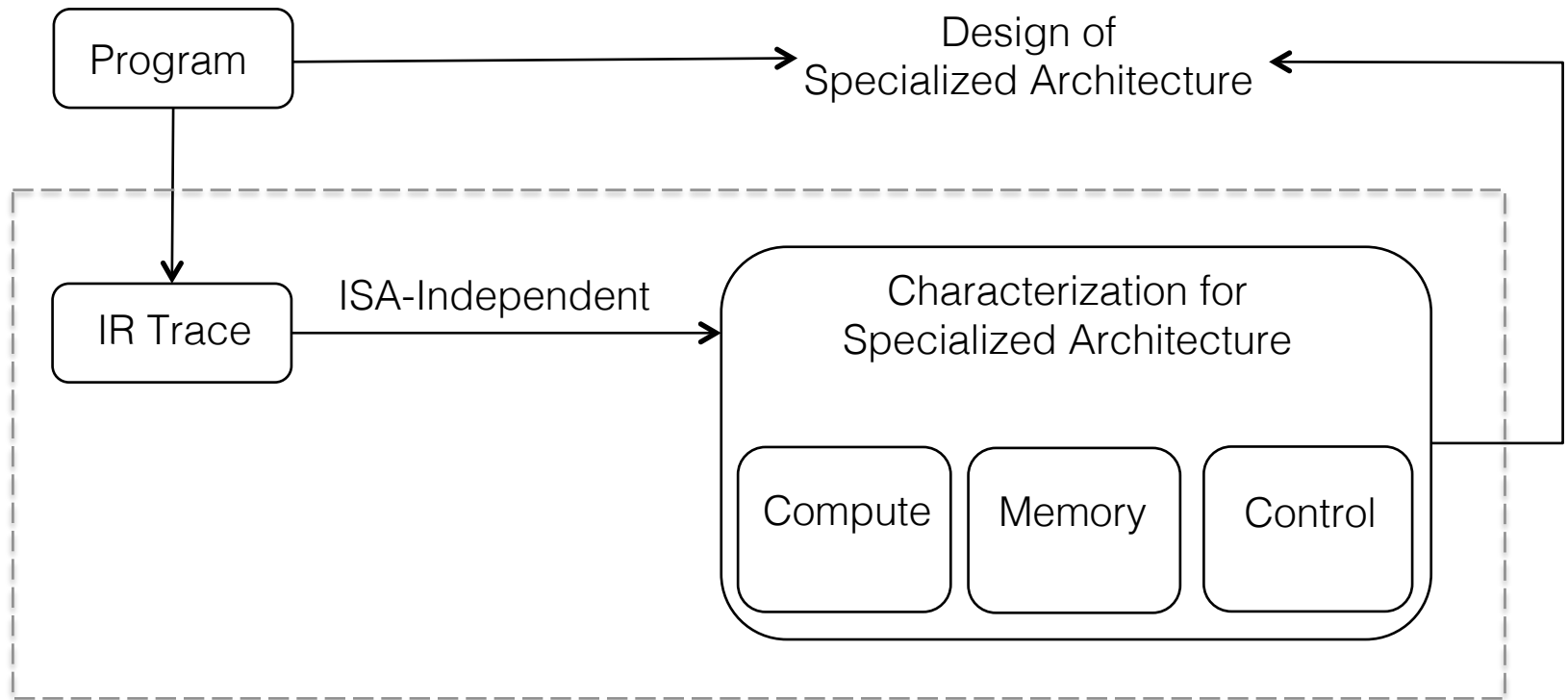
Takeaways:

- ISA-dependent characterization is misleading for specialization.
- ISA-independent characterization allows designers to quickly identify opportunities for specialization.

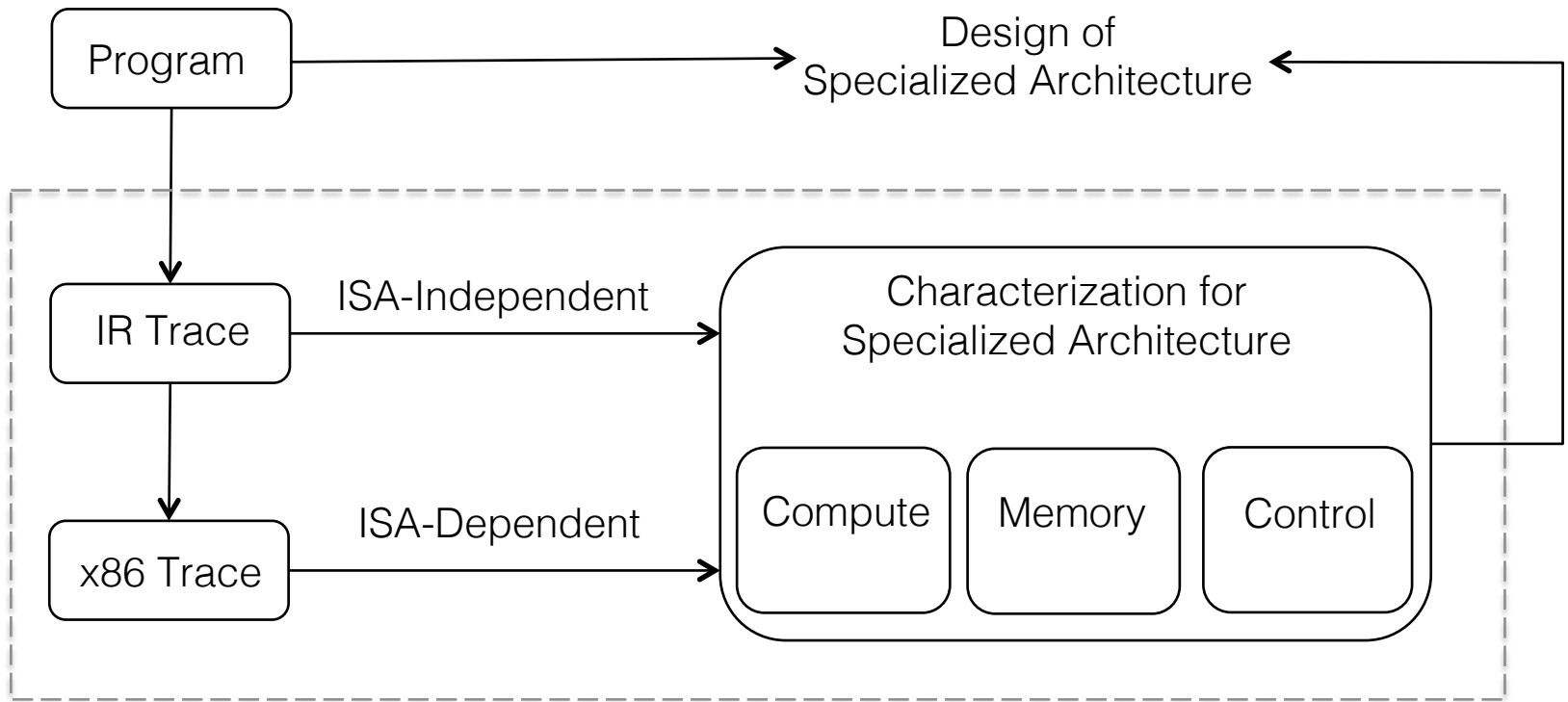
Tool Overview



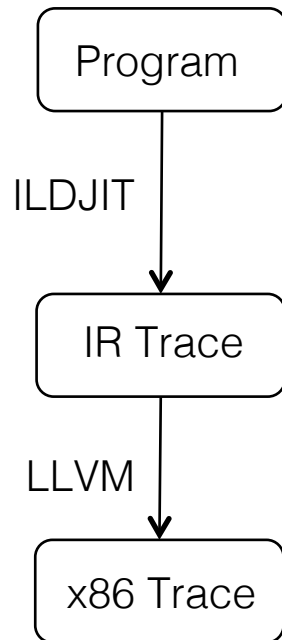
Tool Overview



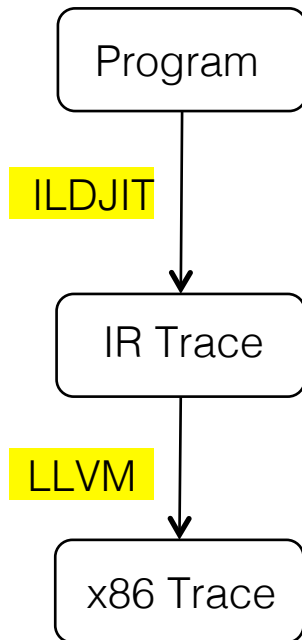
Tool Overview



Program Representations



Program Representations

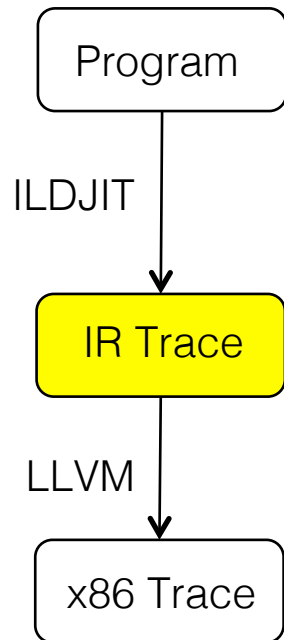


ILDJIT

- A modular compilation framework
- Performs machine-independent classical optimizations at the IR level
- Uses LLVM's back end to
 - Do machine-dependent optimizations
 - Generate machine code

Campanoni, et al., A Highly Flexible, Parallel Virtual Machine: Design and Experience of ILDJIT, Software Practice Experience, 2010

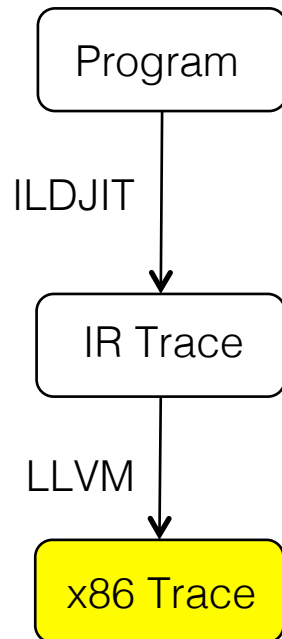
Program Representations



ILDJIT IR

- High-level IR
- Machine-, ISA-, and system-library-independent
- Features:
 - 80 instructions
 - Unlimited registers
 - Only loads/stores access memory
 - No vector operations
 - Parameters are passed by variables

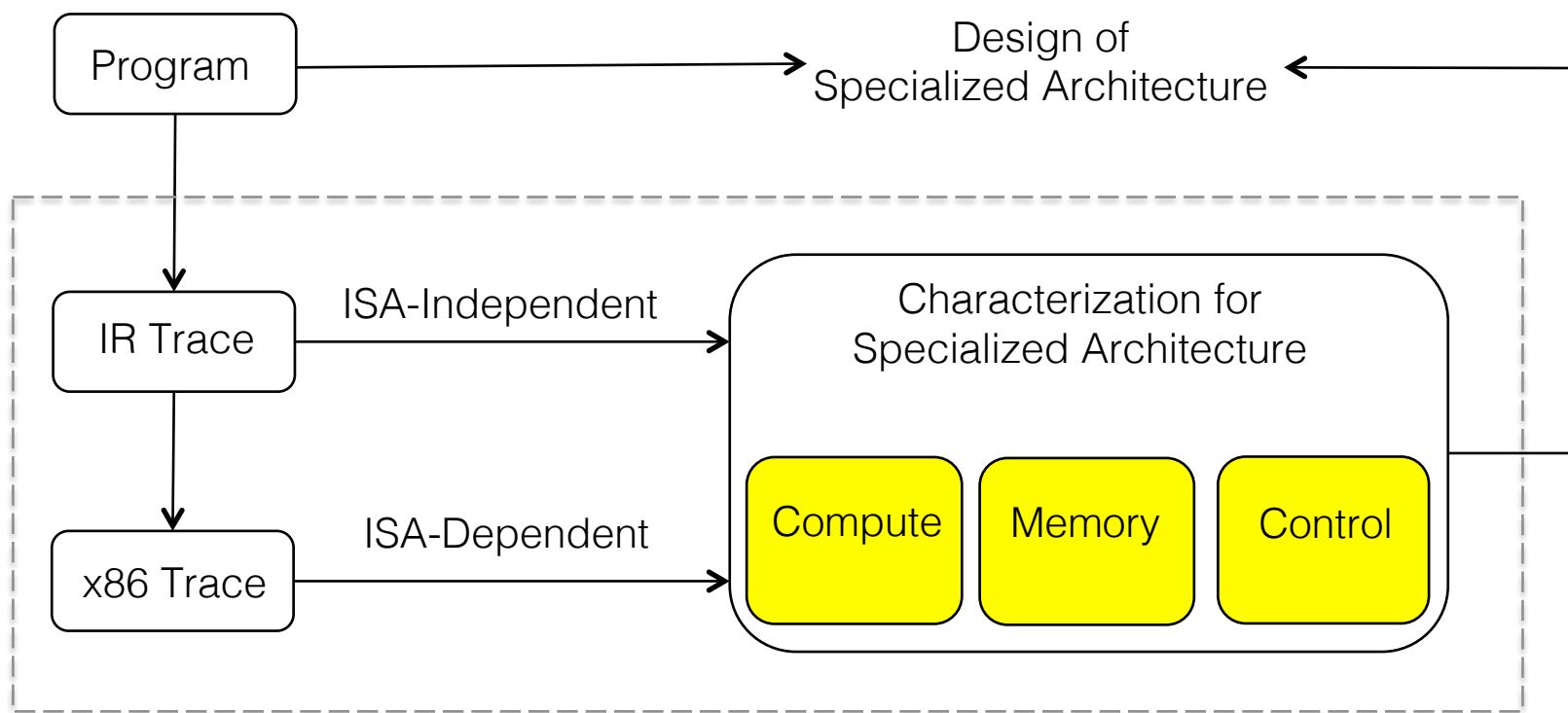
Program Representations



x86 Trace

- Used for ISA-dependent analysis
- Semantically equivalent to the IR code
- Collected with Pin instrumentation

Tool Overview



ISA-Independent Workload Characteristics

Compute

- Opcode Diversity
- Static Instructions (I-MEM)

Memory

- Memory Footprint (D-MEM)
- Memory Entropy
- Locality Score

Control

- Branch Instruction Counts
- Branch Entropy

ISA-Independent Workload Characteristics

Compute

- Opcode Diversity
- **Static Instructions (I-MEM)**

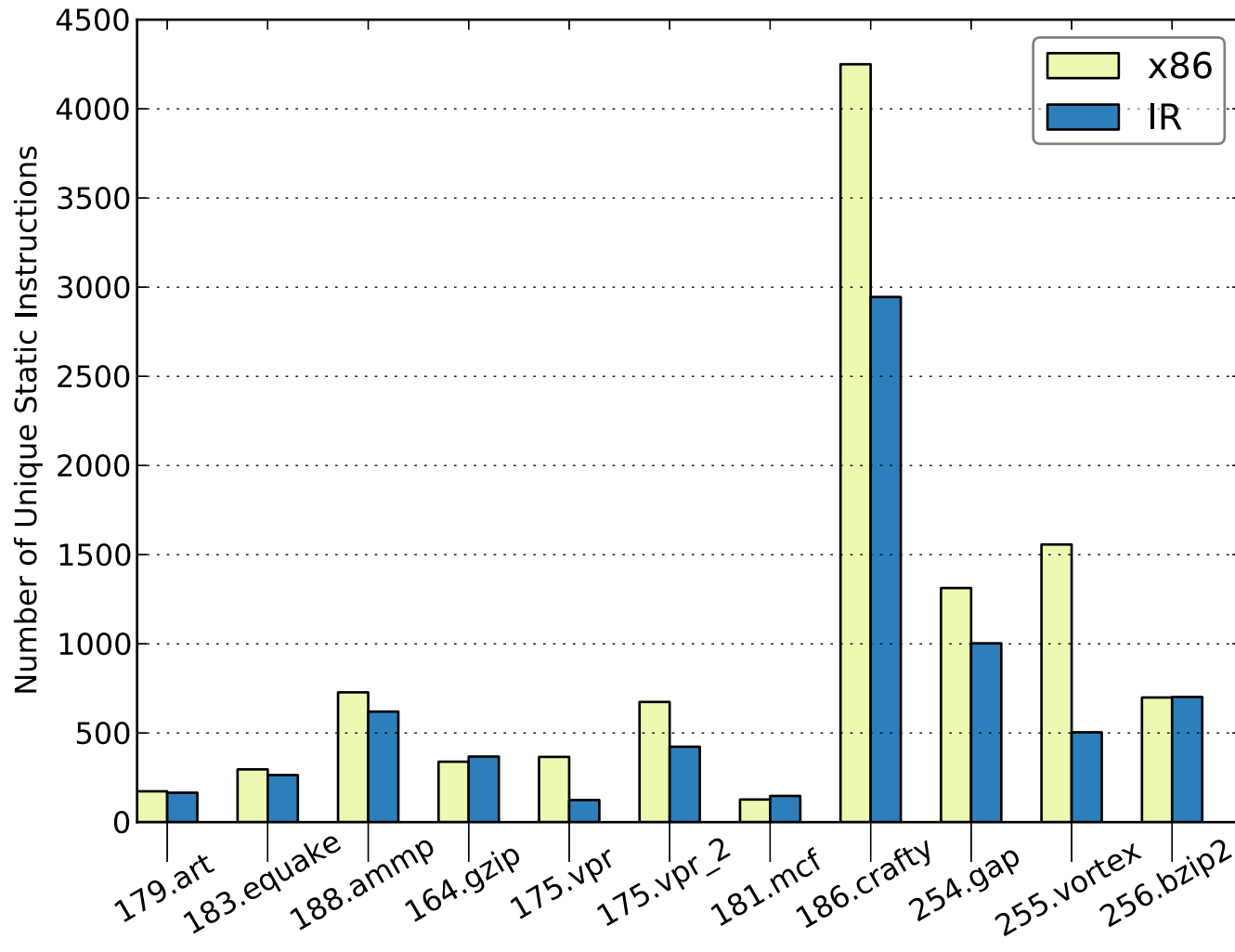
Memory

- Memory Footprint (D-MEM)
- Memory Entropy
- Locality Score

Control

- Branch Instruction Counts
- Branch Entropy

Compute::Static Instructions



ISA-Independent Workload Characteristics

Compute

- Opcode Diversity
- Static Instructions (I-MEM)

Memory

- Memory Footprint (D-MEM)
- **Memory Entropy**
- Locality Score

Control

- Branch Instruction Counts
- Branch Entropy

Memory::Entropy

Entropy: a measure of the randomness

$$Entropy = - \sum_{i=1}^N p(x_i) * \log_2 p(x_i)$$

Memory::Entropy

Entropy: a measure of the randomness

$$Entropy = - \sum_{i=1}^N p(x_i) * \log_2 p(x_i)$$

Case 1:

X is always a constant.

$$p(X) = 1$$

$$\log_2 p(X) = 0$$

$$Entropy = 0$$

Case 2:

N possible outcomes of X occur equally.

$$p(X) = \frac{1}{N}$$

$$\log_2 p(X) = \log_2 N^{-1}$$

$$Entropy = -N * \frac{1}{N} * \log_2 N^{-1}$$

$$Entropy = \log_2 N$$

Memory::Global Address Entropy

Temporal Locality

Address Stream A
(less temporal locality)

0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1

Entropy = 2

Address Stream B
(more temporal locality)

0 0 1 1
0 0 1 1
0 0 1 1
0 0 1 1

Entropy = 0

Yen, Draper, and Hill. Notary: Hardware Techniques to Enhance Signatures. MICRO 08



Memory::Global Address Entropy

Temporal Locality

Address Stream A
(less temporal locality)

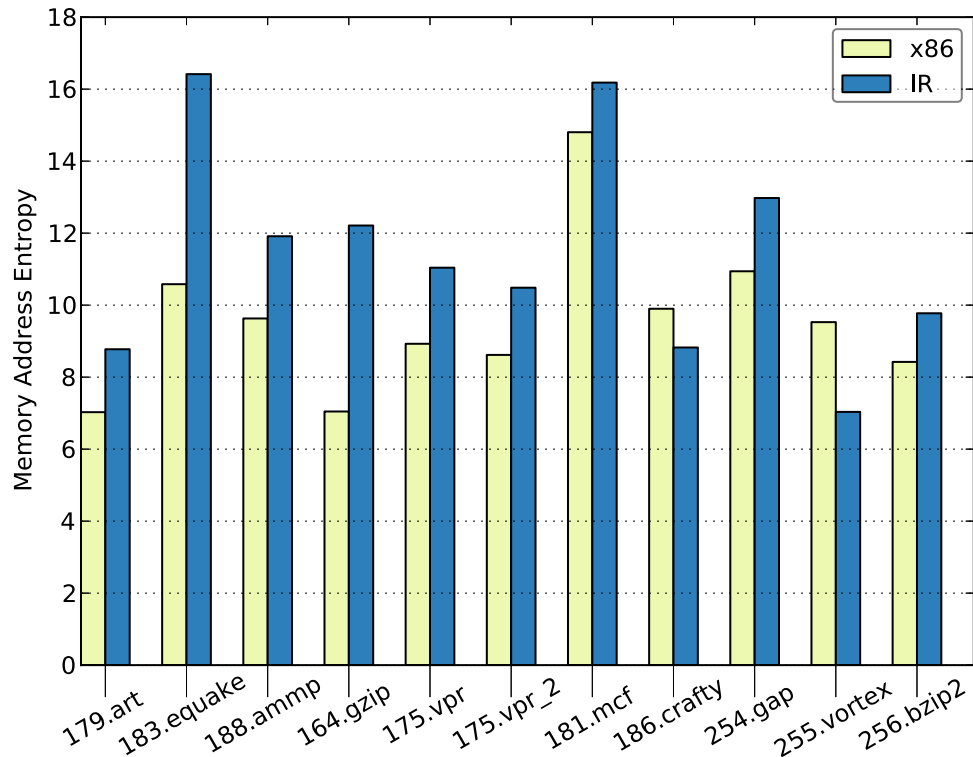
0000
0001
0010
0011

Entropy = 2

Address Stream B
(more temporal locality)

0011
0011
0011
0011

Entropy = 0



Yen, Draper, and Hill. Notary: Hardware Techniques to Enhance Signatures. MICRO 08



Memory::Global Address Entropy

Temporal Locality

Address Stream A
(less temporal locality)

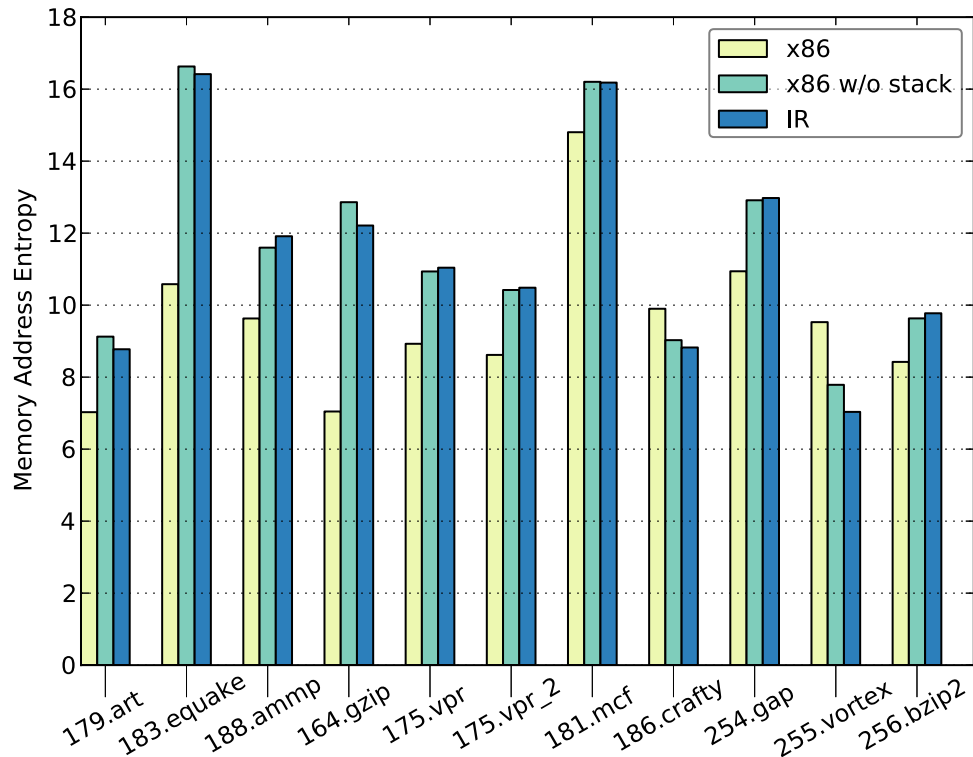
0000
0001
0010
0011

Entropy = 2

Address Stream B
(more temporal locality)

0011
0011
0011
0011

Entropy = 0



Yen, Draper, and Hill. Notary: Hardware Techniques to Enhance Signatures. MICRO 08



Memory::Local Address Entropy

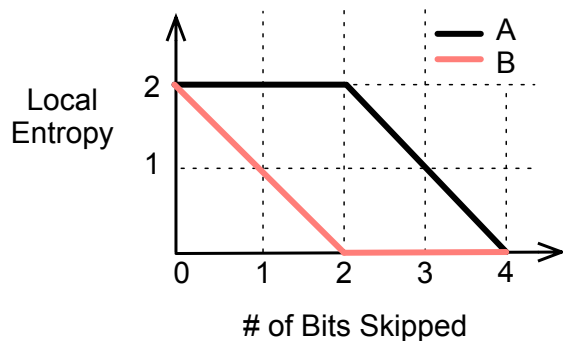
Spatial Locality

Address Stream A
(less spatial locality)

0 0 0 0
0 1 0 0
1 0 0 0
1 1 0 0

Address Stream B
(more spatial locality)

0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1



Memory::Local Address Entropy

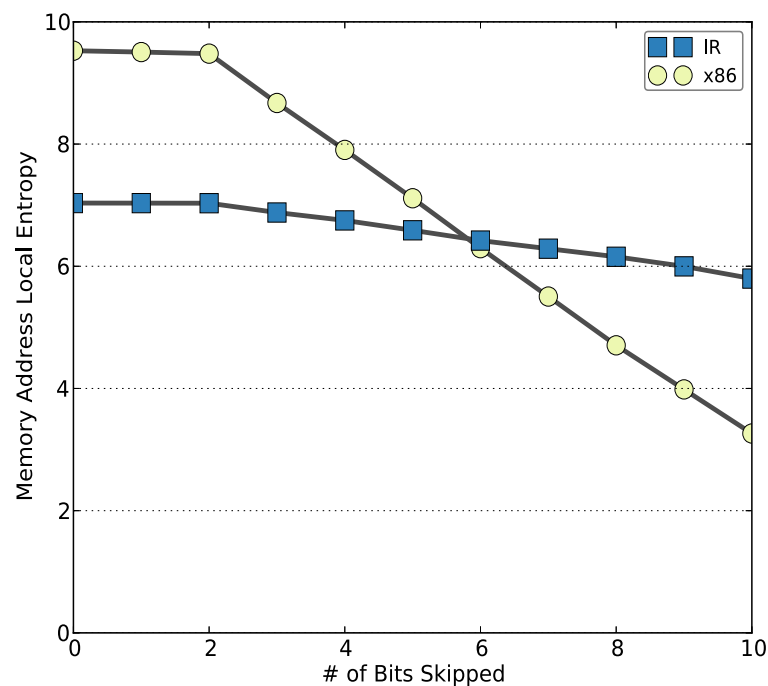
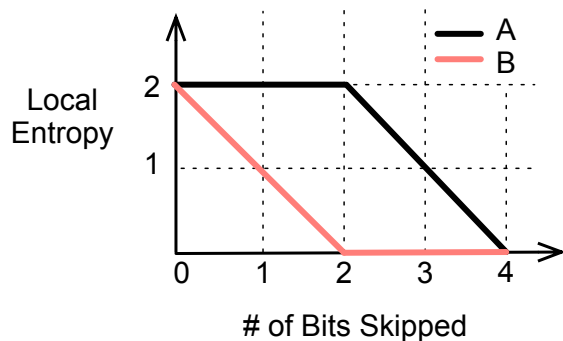
Spatial Locality

Address Stream A
(less spatial locality)

0 0 0 0
0 1 0 0
1 0 0 0
1 1 0 0

Address Stream B
(more spatial locality)

0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1



Memory::Spatial Locality Score

Address Stream A
(less spatial locality)

0x0000

0x0004

0x0008

0x000C

Address Stream B
(more spatial locality)

0x0000

0x0001

0x0002

0x0003

$$L = \frac{P(\text{stride} = 4)}{\text{stride} = 4}$$
$$= \frac{1}{4}$$

$$L = \frac{P(\text{stride} = 1)}{\text{stride} = 1}$$
$$= 1$$

$$L_{\text{spatial}} = \sum_{\text{stride}=1}^{\text{stride} < \infty} \frac{P(\text{stride})}{\text{stride}}$$

Weinberg, et al, Quantifying Locality in the memory access patterns of HPC applications,
In SC, 2005



ISA-Independent Workload Characteristics

Compute

- Opcode Diversity
- Static Instructions (I-MEM)

Memory

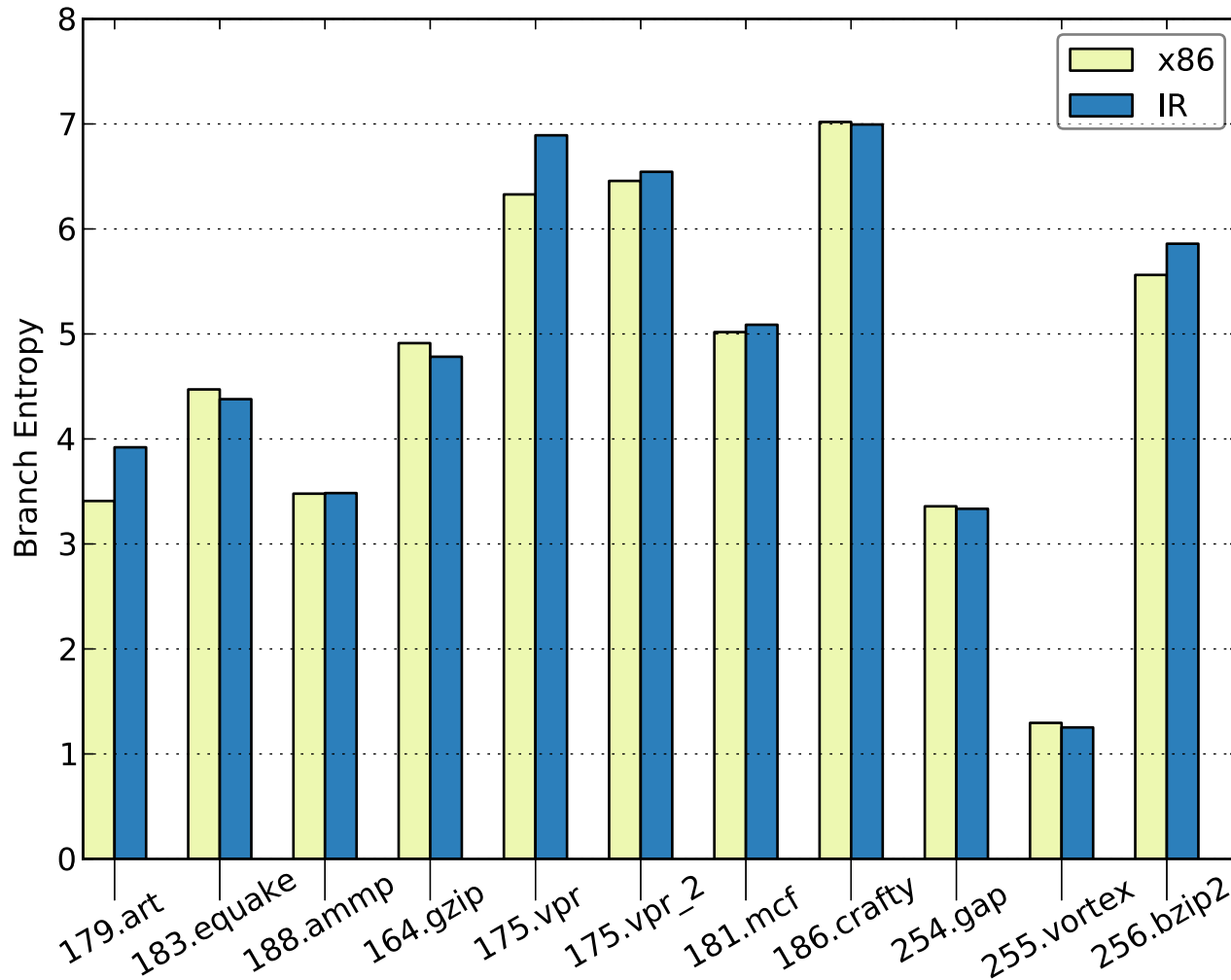
- Memory Footprint (D-MEM)
- Memory Entropy
- Locality Score

Control

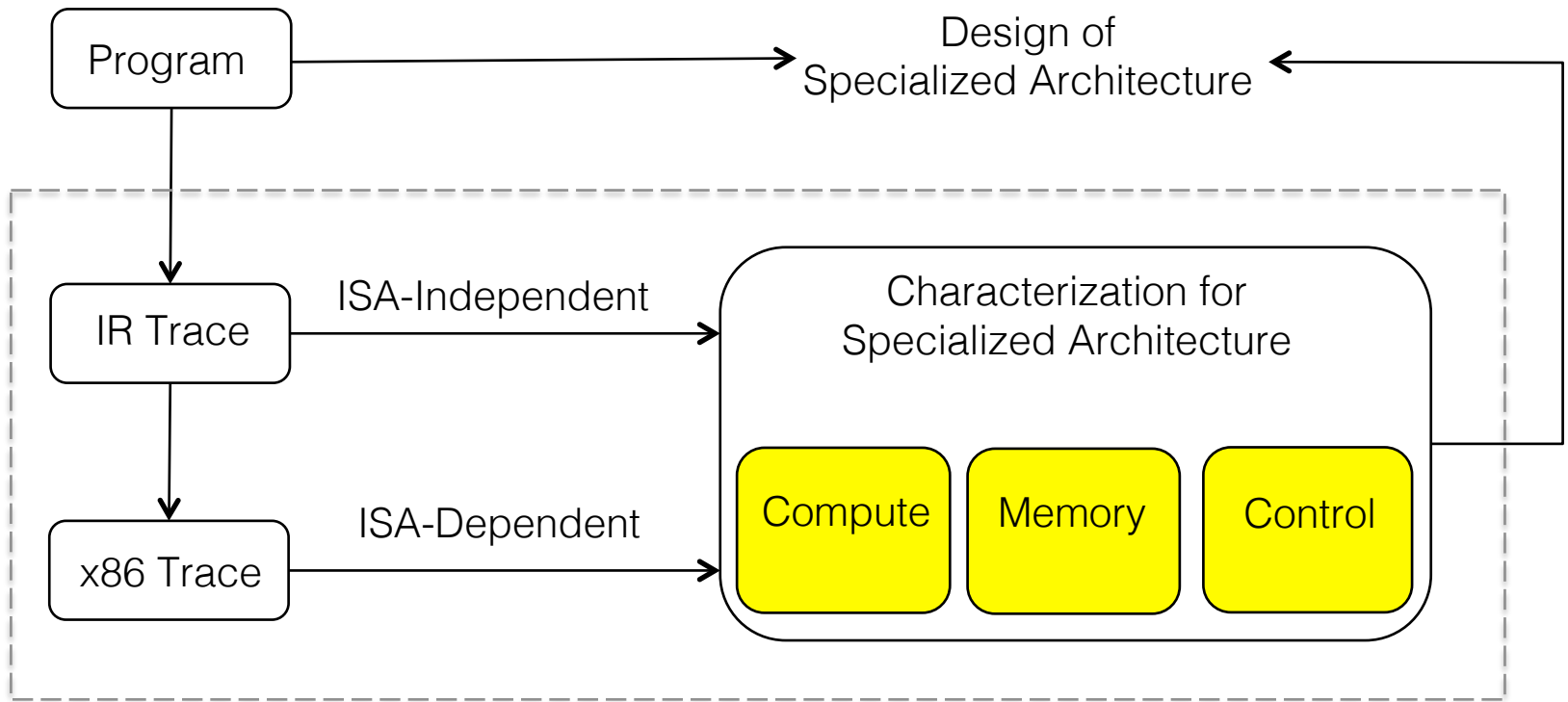
- Branch Instruction Counts
- **Branch Entropy**

Yokota, et al, Introducing Entropies for Representing Program Behavior and Branch Predictor Performance, 07

Control::Branch Entropy



Tool Overview



Publications

- Implications of the Power Wall: Dim Cores and Reconfigurable Logic
 - Wang and Skadron, IEEE Micro, 2013
- ISA-Independent Workload Characterization and its Implications for Specialized Architectures
 - Shao and Brooks, ISPASS, 2013