# Tutorial Outline

<table>
<thead>
<tr>
<th>Time</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:30 am – 9:00 am</td>
<td>Introduction</td>
</tr>
<tr>
<td>9:00 am – 10:00 am</td>
<td><strong>Pre-RTL Simulation Framework: Aladdin</strong></td>
</tr>
<tr>
<td>10:00 am – 10:30 am</td>
<td>Break</td>
</tr>
<tr>
<td>10:30 am – 11:30 am</td>
<td>CAD &amp; Benchmarks: HLS &amp; MachSuite</td>
</tr>
<tr>
<td>11:30 am – 12:00 pm</td>
<td>Aladdin Validation and Case Studies</td>
</tr>
<tr>
<td>12:00 pm – 1:30 pm</td>
<td>Lunch</td>
</tr>
<tr>
<td>1:30 pm – 2:15 pm</td>
<td>Analytical Modeling: Lumos</td>
</tr>
<tr>
<td>2:15 pm – 2:45 pm</td>
<td>Workload Characterization Tool: WIICA</td>
</tr>
<tr>
<td>2:45 pm – 3:00 pm</td>
<td>Hands-on Exercise: Set up</td>
</tr>
<tr>
<td>3:00 pm – 3:30 pm</td>
<td>Break</td>
</tr>
<tr>
<td>3:30 pm – 5:00 pm</td>
<td>Hands-on Exercise</td>
</tr>
</tbody>
</table>
ALADDIN
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

Unmodified C-Code

Accelerator Design Parameters (e.g., # FU, mem. BW)

"Accelerator Simulator" Design Accelerator-Rich SoC Fabrics and Memory Systems

Shared Memory/Interconnect Models

Power/Area

Performance
Aladdin enables pre-RTL simulation of accelerators with the rest of the SoC.
Aladdin: A pre-RTL, Power-Performance Accelerator Simulator

Unmodified C-Code

Accelerator Design Specific Parameters (e.g., # FU, mem. BW)

Shared Memory/Interconnect Models

“Accelerator Simulator” Design Accelerator-Rich SoC Fabrics and Memory Systems

↑ Flexibility

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

- **Aladdin**
  - Accelerator Specific Datapath
  - Private L1/Scratchpad

Shared Memory/Interconnect Models

- **Unmodified C-Code**
- **Accelerator Design Parameters** (e.g., # FU, mem. BW)

“Accelerator Simulator”
- Design Accelerator-Rich SoC Fabrics and Memory Systems

“Design Assistant”
- Understand Algorithmic-HW Design Space before RTL

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

↓ **Design Cost**

Power/Area

Performance
Future Accelerator-Centric Architecture

- Big Cores
- Small Cores
- Shared Resources
- Memory Interface
- Sea of Fine-Grained Accelerators

Graph showing Execution Time (μS) vs. Power (mW) with data points for HLS.
Future Accelerator-Centric Architecture

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

- **C Code**
  - Optimistic IR
  - Initial DDDG
  - Program Constrained DDDG
  - Power/Area Models

- **Acc Design Parameters**
  - Idealistic DDDG
  - Resource Constrained DDDG

**Dynamic Data Dependence Graph (DDDG)**

**Optimization Phase**

**Realization Phase**
Aladdin Overview

- **Optimization Phase**
  - C Code
  - Optimistic IR
  - Initial DDDG
  - Idealistic DDDG

- **Realization Phase**
  - Program Constrained DDDG
  - Resource Constrained DDDG
  - Power/Area Models

- **Performance**
- **Activity**
- **Power/Area**
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

Optimization Phase

C Code

Optimistic IR

Initial DDDG

Idealistic DDDG

Program Constrained DDDG

Resource Constrained DDDG

Power/Area Models

Realization Phase

Performance

Activity

Power/Area

Acc Design Parameters

Design Parameters
From C to Design Space

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

C Code

Optimization Phase

- Optimistic IR
- Initial DDDG
- Idealistic DDDG

Realization Phase

- Acc Design Parameters
- Performance
- Activity
- Power/Area Models
- Power/Area

Design Parameters

- 16
From C to Design Space

**Initial DDDDG**

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

[Diagram of the initial DDDDG graph]
Aladdin Overview

Optimization Phase

C Code

Optimistic IR → Initial DDDG → Idealistic DDDG

Acc Design Parameters

Program Constrained DDDG → Resource Constrained DDDG → Power/Area Models

Realization Phase

Performance → Activity → Power/Area
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
...

From C to Design Space
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

Optimization Phase

C Code

- Optimistic IR → Initial DDDG → Idealistic DDDG

Acc Design Parameters

- Program Constrained DDDG → Resource Constrained DDDG → Power/Area Models

Realization Phase

Performance

Activity

Power/Area
From C to Design Space

One Design

Idealistic DDDG

Acc Design Parameters:
✓ Memory BW <= 2
✓ 1 Adder

Resource Activity

Cycle
From C to Design Space

Another Design

Idealistic DDDG

Acc Design Parameters:
✓ Memory BW <= 4
✓ 2 Adders

Resource Activity

Cycle
Cycle
Power

Acc Design Parameters:
✓ Memory BW <= 4
✓ 2 Adders

Acc Design Parameters:
✓ Memory BW <= 2
✓ 1 Adder

From C to Design Space
Power-Performance per Design
From C to Design Space

Design Space of an Algorithm

![Diagram showing the relationship between Power and Cycle with data points and a trend line.](image-url)
Cycle-Level Activity

![Graph showing cycle-level activity with time in cycles on the x-axis and number of active functional units on the y-axis. The graph includes lines for active functional units and memory bandwidth.]
Power Model

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

\[
Power = \sum_{1<i<N} (activity_i \cdot P_i^{\text{switching}}) + P_i^{\text{internal}} + P_i^{\text{leakage}}
\]

• SRAM Power Model
  – Commercial register file and SRAM memory compilers with the same 40nm standard cell library
  – CACTI
From C to Design Space

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

Input:
a[0] = 1;
a[1] = 2;
a[2] = 2;
...
for(i=0; i<N; ++i)
{
    bucket[ a[i] & 0x11 ]++;
}

Input:
\[
a[0] = 1;
\]
\[
a[1] = 2;
\]
\[
a[2] = 2;
\]
...
for(i=0; i<N; ++i) {
    bucket[ a[i] & 0x11 ]++; 
}

Input:
a[0] = 1;
a[1] = 2;
a[2] = 2;
...
for(i=0; i<N; ++i)
{
    bucket[a[i] & 0x11]++;
}

Input:
a[0] = 1;
a[1] = 2;
a[2] = 2;
...
for(i=0; i<N; ++i) {
    bucket[ a[i] & 0x11 ]++; 
}

Input:

```
float a[0] = 1;
float a[1] = 2;
float a[2] = 2;
... 
```
# Tutorial Outline

<table>
<thead>
<tr>
<th>Time</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:30 am – 9:00 am</td>
<td>Introduction</td>
</tr>
<tr>
<td>9:00 am – 10:00 am</td>
<td>Pre-RTL Simulation Framework: Aladdin</td>
</tr>
<tr>
<td>10:00 am – 10:30 am</td>
<td>Break</td>
</tr>
<tr>
<td><strong>10:30 am – 11:30 am</strong></td>
<td><strong>CAD &amp; Benchmarks: HLS &amp; MachSuite</strong></td>
</tr>
<tr>
<td>11:30 am – 12:00 pm</td>
<td>Aladdin Validation and Case Studies</td>
</tr>
<tr>
<td>12:00 pm – 1:30 pm</td>
<td>Lunch</td>
</tr>
<tr>
<td>1:30 pm – 2:15 pm</td>
<td>Analytical Modeling: Lumos</td>
</tr>
<tr>
<td>2:15 pm – 2:45 pm</td>
<td>Workload Characterization Tool: WIICA</td>
</tr>
<tr>
<td>2:45 pm – 3:00 pm</td>
<td>Hands-on Exercise: Set up</td>
</tr>
<tr>
<td>3:00 pm – 3:30 pm</td>
<td>Break</td>
</tr>
<tr>
<td>3:30 pm – 5:00 pm</td>
<td>Hands-on Exercise</td>
</tr>
</tbody>
</table>
# Tutorial Outline

<table>
<thead>
<tr>
<th>Time</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:30 am – 9:00 am</td>
<td>Introduction</td>
</tr>
<tr>
<td>9:00 am – 10:00 am</td>
<td>Pre-RTL Simulation Framework: Aladdin</td>
</tr>
<tr>
<td>10:00 am – 10:30 am</td>
<td>Break</td>
</tr>
<tr>
<td>10:30 am – 11:30 am</td>
<td>CAD &amp; Benchmarks: HLS &amp; MachSuite</td>
</tr>
<tr>
<td><strong>11:30 am – 12:00 pm</strong></td>
<td><strong>Aladdin Validation and Case Studies</strong></td>
</tr>
<tr>
<td>12:00 pm – 1:30 pm</td>
<td>Lunch</td>
</tr>
<tr>
<td>1:30 pm – 2:15 pm</td>
<td>Analytical Modeling: Lumos</td>
</tr>
<tr>
<td>2:15 pm – 2:45 pm</td>
<td>Workload Characterization Tool: WIICA</td>
</tr>
<tr>
<td>2:45 pm – 3:00 pm</td>
<td>Hands-on Exercise: Set up</td>
</tr>
<tr>
<td>3:00 pm – 3:30 pm</td>
<td>Break</td>
</tr>
<tr>
<td>3:30 pm – 5:00 pm</td>
<td>Hands-on Exercise</td>
</tr>
</tbody>
</table>
Aladdin: A pre-RTL, Power-Performance Accelerator Simulator

- Unmodified C-Code
- Accelerator Design Specific Parameters (e.g., # FU, mem. BW)
- Accelerator Specific Datapath
- Private L1/Scratchpad
- Shared Memory/Interconnect Models

“Accelerator Simulator” Design Accelerator-Rich SoC Fabrics and Memory Systems

“Design Assistant” Understand Algorithmic-HW Design Space before RTL

Flexibility
Programmability

Design Cost
Aladdin Overview

Optimization Phase

C Code

Optimistic IR

Initial DDDG

Idealistic DDDG

Acc Design Parameters

Program Constrained DDDG

Resource Constrained DDDG

Power/Area Models

Performance

Activity

Power/Area

Realization Phase
Aladdin Validation

C Code

Aladdin

Power/Area

Performance

Verilog

Design Compiler

Activity

ModelSim
Aladdin Validation

C Code

RTL Designer

HLS C Tuning

Vivado HLS

Verilog

Design Compiler

ModelSim

Power/Area

Performance
## Validation Benchmarks

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

Optimized HLS Designs

Hand RTL Designs
Aladdin Validation

0.6%

6.5%

4.5%
Aladdin Validation

- Time (KCycles)
  - MD: 0.9%
  - STENCIL: 4.9%
  - FFT: 6.5%

- Power (mW)
  - NPU: 4.9%
  - HASH: 6.5%

- Area (mm²)
  - NPU: 7%
  - HASH: 9%
  - HARP: 15%
Aladdin enables rapid design space exploration for accelerators.

7 mins

52 hours
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

Without Memory Contention

With Memory Contention
Publications

• Quantifying Acceleration – ISLPED 2013
  – Reagen, Shao, Wei, Brooks

• MachSuite – IISWC 2014
  – Reagen, Adolf, Shao, Wei, Brooks

• Aladdin – ISCA 2014
  – Shao, Reagen, Wei, Brooks