<table>
<thead>
<tr>
<th>Time</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>9:00 am – 9:30 am</td>
<td>Introduction</td>
</tr>
<tr>
<td>9:30 am – 10:10 am</td>
<td>Standalone Accelerator Simulation: Aladdin</td>
</tr>
<tr>
<td>10:10 am – 10:30 am</td>
<td>Standalone Accelerator Generation: High-Level Synthesis</td>
</tr>
<tr>
<td>10:30 am – 11:00 am</td>
<td>HLS-Based Accelerator-Rich Architecture Simulation: PARADE</td>
</tr>
<tr>
<td>11:00 am – 11:30 am</td>
<td>Break</td>
</tr>
<tr>
<td>11:30 am – 12:00 pm</td>
<td>Pre-RTL SoC Simulation: gem5-Aladdin</td>
</tr>
<tr>
<td>12:00 pm – 12:30 pm</td>
<td>FPGA Prototyping: ARACompiler</td>
</tr>
<tr>
<td>12:30 pm – 2:00 pm</td>
<td>Lunch</td>
</tr>
<tr>
<td>2:00 pm – 3:00 pm</td>
<td>Panel on Accelerator Research</td>
</tr>
<tr>
<td>3:00 pm – 3:30 pm</td>
<td>Accelerator Benchmarks and Workload Characterization</td>
</tr>
<tr>
<td>3:30 pm – 4:00 pm</td>
<td>Break</td>
</tr>
<tr>
<td>4:00 pm – 5:00 pm</td>
<td>Hands-on Exercise</td>
</tr>
</tbody>
</table>

Zhenman Fang, Michael Gill
Jason Cong, Glenn Reinman

Center for Domain-Specific Computing
Center for Future Architectures Research
Computer Science Department, UCLA

[ICCAD 15]
The Power Wall and Customized Computing

Adapt the architecture to application domain

Parallelization
Customization

Source: Shekhar Borkar, Intel
The Trend of Accelerator-Rich Architecture (ARA)

Global Accelerator Manager (GAM) with shared TLB

From ARC [DAC 12] & CHARM [DAC 14]
Our Motivation and Goal

A stack of research tools for accelerator-rich architecture

- Standalone accelerator simulation: Aladdin
- Standalone accelerator generation: HLS
- **System-level HLS-based ARA simulation: PARADE**
- System-level pre-RTL SoC simulation: gem5 + Aladdin
- ARA FPGA prototyping: ARACompiler

Spare the community the difficulties we have encountered

Accelerate the adoption of accelerator-rich architecture (ARA)
**PARADE: Platform for Accelerator-Rich Architectural Design & Exploration [ICCAD 15]**

- Extended m5 (McPAT) for X86 CPU, with OS
- Auto-generated accelerators based on HLS (AutoPilot)
- Added SPM, DMA, GAM & TLB model
- Extended Garnet (DSENT) for NoC
- Extended Ruby (CACTI) for coherent cache hierarchy
- gem5 memory model [ISPASS 14]
HLS-based Automatic Accelerator Generation

- **High-Level Synthesis**
  - RTL Synthesis
  - RTL model
  - C function to accelerate

- **Simulation Module**
  - Timing info e.g., II, clk
  - Simulation module info

- **Accelerator Source Code**
  - Handles accelerator communication, task buffer, interrupts, ...

- **Application Dataflow**
  - Accelerators chaining info

- **Program Generator**

- **Simulation Module Generator**

- **Generated Program**

- **Tool**

- **Input**

- **Output**
According to the component’s utilization, assign a different color. The dark red indicates a potential bottleneck!
Tutorial Agenda

Building the PARADE simulator
Creating a benchmark
Running a benchmark on PARADE
Performance and energy analysis using PARADE
  ▪ Performance breakdown
  ▪ Energy breakdown
Simulation speed of PARADE
Summary of PARADE features
Building PARADE

Use an existing accelerator module: VectorAddSample

- `#include "VectorAddSample.hh"` in `LCAccOperatingModeInclude.hh`
  - `VectorAddSample.hh` contains accelerator modeling details
- `REGISTER_OPMODE(OperatingMode_VectorAddSample)` in `LCAccOperatingModeListing.hh`
- `g_LCAccInterface->AddOperatingMode(g_LCAccDeviceHandle[0], "VectorAddSample");` in `startup().mem/ruby/system/System.cc`

The same compiling command as gem5 (32nm @2GHz)

- `scons PROTOCOL=MESI_Two_Level_Trace build/X86/gem5.opt`
Adding an Accelerator Simulation Module

User high-level description: VectorAddSample.type

```plaintext
1. MODULE VectorAddSample
2. OPCODE 14001
3. POWER 1.0 1390 uw
4. AREA 1.0 52806 um²
5. DEPTH 1 10
6. CYCLE 1 2 (1GHz)
7. II 1 1
8. ARGUMENT INPUT float input_a
9. ARGUMENT INPUT float input_b
10. ARGUMENT OUTPUT float output
11. BODY
12. output = input_a + input_b;
13. END
```

Specify an unique accelerator module name and opcode

Replace timing by our HLS/RTL tool
`./run.autopilot.sh VectorAddSample.type`

Specify accelerator inputs and outputs

Specify accelerator computation body: one iteration in the loop
Auto-Generated Accelerator Simulation Module

Auto-generated accelerator module: VectorAddSample.hh

each accelerator inherits the LCAccOperationMode class

accelerator module name and opcode

accelerator timing info from HLS/RTL

auto-generated SPM address mapping

auto-generated SPM/DMA timing, and computation latency

```
class OperatingMode_VectorAddSample
  : public LCAccOperatingMode
{
  ...

  static std::string GetModeName()
  {
    return "VectorAddSample";
  }
  static int GetOpCode(){return 14001;}

  virtual int PipelineDepth(){return 10;}
  virtual int CycleTime(){return 2;}
  virtual int InitiationInterval(){return 1;}
  virtual void GetSPMReadIndexSet(...) {...}
  virtual void GetSPMWriteIndexSet(...) {...}
  virtual void Compute(...) {
    ...
    input_a = ReadSPMFLt(0, addr_input_a, 0);
    input_b = ReadSPMFLt(1, addr_input_b, 0);
    output = input_a + input_b;
    WriteSPMFLt(2, addr_output, 0, output);
  }
  ...
};
```
mudo AccGen.exe -path:src/modules/LCAcc VectorAddSample.type
Creating a Benchmark

To use existing accelerators, just call accelerator API

```c++
#include "BenchmarkNode.h"
#include "VectorAddSampleLCacc.h"

class Sample : public BenchmarkNode {

...;

virtual void Initialize(int threadID, int procID) {
    CreateBuffer_VectorAddSampleLCacc(&buf, &bufSize, 
        thread, input_1, input_2, output, dataSize, tileSize);
}

virtual void Run() {
    VectorAddSampleLCacc_buf(buf, bufSize, thread);
}
}

BENCHDECL(Sample);
```

Create a BenchmarkNode, call Initialize() and Run().
Within the Accelerator Library (for Benchmarks)

1. Request available accelerators (lcacc-req)
2. Response available ones & waiting time
3. Request reservation (lcacc-rsv) and wait
4. Reserve accelerator, send it the core ID
5. The core shares a task description and start the accelerator (lcacc-cmd)
6. Read task & start work
7. Work done, notify the core
8. Free accelerators (lcacc-free)

New ISA

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lcacc-req</td>
<td>type</td>
</tr>
<tr>
<td>lcacc-rsv</td>
<td>id, time</td>
</tr>
<tr>
<td>lcacc-cmd</td>
<td>id, cmd, addr</td>
</tr>
<tr>
<td>lcacc-free</td>
<td>id</td>
</tr>
</tbody>
</table>

Users don’t have to worry about these, we provide a dataflow language and tool to automatically generate the library
Creating an Accelerator Library (for Benchmarks)

To create an accelerator library, specify accelerator dataflow

```
1  VARIABLE STREAM IN float in_a [size]
2  VARIABLE STREAM IN float in_b [size]
3  VARIABLE STREAM OUT float out [size]
4
5  VARIABLE int size
6  VARIABLE int chunk
7
8  TASK [0..size:chunk]
9  SPM_WINDOWS 2
10
11 DECLARE VectorAddSample vAccel []
12
13 CREATE_SPM vAccel float region_a [chunk]
14 CREATE_SPM vAccel float region_b [chunk]
15 CREATE_SPM vAccel float region_c [chunk]
16
17 in_a[0..size] => vAccel.region_a[0..chunk]
18 in_b[0..size] => vAccel.region_b[0..chunk]
19 vAccel.region_c[0..chunk] => out[0..size]
20
21 vAccel.region_a[0..chunk] => vAccel.input_a
22 vAccel.region_b[0..chunk] => vAccel.input_b
23 vAccel.output => vAccel.region_c[0..chunk]
```

input and output
whole data size and tile size
task based on tile size (chunk)
use double SPM buffer
declare the accelerator
create SPM for input/output
data transfer based on tile:
input LLC/DRAM -> SPM & output SPM -> LLC/DRAM
trigger accelerator within tile:
input SPM -> Register & output Register -> SPM

(\textit{mono ApGen.exe VectorAddSample.txt VectorAddSampleLCacc.h})
Running a Benchmark on PARADE

Similar gem5 command to run benchmarks

- ./gem5.opt --outdir=./TDLCA_BlackScholes/configs/example/fs.py full-system config
- --checkpoint-dir=./ckpt-1core/ --restore-with-cpu=timing -r 1 -n 1 restore checkpoint
- -s 16565183 -W 16565183 timing warmup initialization, then switch to OoO
- --ruby --l2_size=64kB --num-l2caches=32 32-banked 2MB LLC with Ruby
- --mem-size=2GB --num-dirs=4 2GB memory with 4 DDR3 controllers
- --garnet=fixed --topology=Mesh --mesh-rows=4 4*8 mesh with Garnet
- --lcacc --accelerators=1 1 copy of accelerator
- --script=./configs/boot/BlackScholes.td.rcS BlackScholes boot script
- >& TDLCA_BlackScholes/result.txt redirect output to result.txt

Output statistics of PARADE

- Stats.txt, result.txt, and visual.txt (visualization trace)
Execution Cycles for BlackScholes

- BlackScholes-SW
- BlackScholes-dedicated-ARA

<table>
<thead>
<tr>
<th>Category</th>
<th>BlackScholes-SW</th>
<th>BlackScholes-dedicated-ARA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>1,000,000,000</td>
<td>10,000,000,000</td>
</tr>
<tr>
<td>Computation</td>
<td>100,000,000</td>
<td>1,550,000,000</td>
</tr>
<tr>
<td>Non-overlapped</td>
<td>10,000,000</td>
<td>640,000,000</td>
</tr>
</tbody>
</table>

Execution cycles:
- Total: 94X
- Computation: 155X
- Non-overlapped Communication: 64X
Execution Cycles for BlackScholes (cont.)

Total cycles: check stats.txt
- `system.switch_cpus_1.numCycles` 3158763

CPU/SW computation time (assume perfect cache)
- Change configuration to use 20MB L2 cache with 1 cycle latency

ARA computation/communication time
- `Result.txt` contains start, and end time for each task computation and data transfer
- Postprocessing “ResultParser result.txt”, it will generate

<table>
<thead>
<tr>
<th>Read</th>
<th>Write</th>
<th>Compute</th>
<th>R + C</th>
<th>R + W</th>
<th>W + C</th>
<th>R + C + W</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,923,952</td>
<td>1,567,370</td>
<td>1,048,576</td>
<td>421,202</td>
<td>839,828</td>
<td>657,081</td>
<td>198,306</td>
</tr>
</tbody>
</table>
Issuing IPC Breakdown for BlackScholes

# of issued instructions per cycle on CPU (BlackScholes)

- No issuing instruction: 62%
- Other issue counts: 15%, 7%, 5%, 5%, 3%, 1%, 1%

For the accelerator version, it’s customized to a fully-utilized 234-stage deep pipeline
Issuing IPC Breakdown for BlackScholes (cont.)

Check stats.txt

| system.switch_cpus_1.iq.issued_per_cycle::0 | 183230654 | 61.68% |
| system.switch_cpus_1.iq.issued_per_cycle::1 | 43824558  | 14.75% |
| system.switch_cpus_1.iq.issued_per_cycle::2 | 20315383  | 6.84%  |
| system.switch_cpus_1.iq.issued_per_cycle::3 | 15288101  | 5.15%  |
| system.switch_cpus_1.iq.issued_per_cycle::4 | 14263448  | 4.80%  |
| system.switch_cpus_1.iq.issued_per_cycle::5 | 9026183   | 3.04%  |
| system.switch_cpus_1.iq.issued_per_cycle::6 | 3865093   | 1.30%  |
| system.switch_cpus_1.iq.issued_per_cycle::7 | 3818686   | 1.29%  |
| system.switch_cpus_1.iq.issued_per_cycle::8 | 3453974   | 1.16%  |

ARA version based on HLS

- `./run.autopilot.sh BlackScholes.type`
# of Cache/Memory Access for BlackScholes

- **BlackScholes-SW**
- **BlackScholes-dedicated-ARA**

### L1D/SPM
- **42X L1D reduction by removing register spilling**

### L1I
- **removed instructions**
- **no notable reduction**

### LLC

### Memory
# of Cache/Memory Access for BlackScholes (cont.)

Check stats.txt

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>system.ruby.l1_cntrl0.L1Dcache.demand_hits</td>
<td>8004741</td>
</tr>
<tr>
<td>system.ruby.l1_cntrl0.L1Dcache.demand_misses</td>
<td>330835</td>
</tr>
<tr>
<td>system.ruby.num_spm_reads</td>
<td>1572864</td>
</tr>
<tr>
<td>system.ruby.num_spm_writes</td>
<td>262144</td>
</tr>
<tr>
<td>system.ruby.l1_cntrl0.L1Icache.demand_hits</td>
<td>22606088</td>
</tr>
<tr>
<td>system.ruby.l1_cntrl0.L1Icache.demand_misses</td>
<td>87604</td>
</tr>
<tr>
<td>system.ruby.l2_cntrl0.L2cache.demand_hits</td>
<td>26161</td>
</tr>
<tr>
<td>system.ruby.l2_cntrl0.L2cache.demand_misses</td>
<td>10561</td>
</tr>
<tr>
<td>system.mem_ctrls0.num_reads::total</td>
<td>71657</td>
</tr>
<tr>
<td>system.mem_ctrls0.num_writes::total</td>
<td>43666</td>
</tr>
</tbody>
</table>
Cache/Memory Bandwidth for BlackScholes

Bandwidth = Access * 64B * 10^-6/(Cycles/2GHz) (MB/s)

- BlackScholes-SW
- BlackScholes-dedicated-ARA

Bandwidth =

\[ \text{Access} \times 64B \times 10^{-6}/(\text{Cycles}/2GHz) \times (\text{MB/s}) \]

- L1D/SPM
- LLC
- Memory

improved LLC/DRAM effective bandwidth
Energy Breakdown for Deblur

Energy breakdown for dedicated ARA (Deblur)

- DRAM: 67%
- Core_Acc: 14%
- LLC: 12%
- NoC: 7%
Energy Breakdown for Deblur (cont.)

Energy = power * time, we measure power directly

Accelerator power: given by run.autopilot.sh

DRAM power (gem5 integrated Micron model, ISPASS 14)
- system.memCtrls0.averagePower::0 753.996167 in stats.txt

Core power (McPAT)
- generate.mcpat.xml.sh convert gem5 statistics to mcpat xml
- generate.mcpat.energy.energy.sh generate power numbers

Total Cores: 1 cores
Device Type= ITRS high performance device type
  Area = 14.6922 mm^2
  Peak Dynamic = 8.9637 W
  Subthreshold Leakage = 0.86593 W
  Gate Leakage = 0.299078 W
  Runtime Dynamic = 1.6622 W
Energy Breakdown for Deblur (cont.)

LLC power (CACTI, integrated with McPAT)

Total L2s:
Device Type= ITRS high performance data
Area = 9.24735 mm^2
Peak Dynamic = 433.643 W
Subthreshold Leakage = 0.707949 W
Gate Leakage = 0.328609 W
Runtime Dynamic = 0.00525153 W

NoC power (DSENT)
- run generate.dsent.sh

system.ruby.network.int_links00.nls0
Link:
  Dynamic power: 1.74243e-05
  Leakage power: 2.18209e-05
Link_total_power: 3.92451e-05
Simulation Speed of PARADE

KIPS: Kilo Instructions simulated Per Second

- gem5-SW_KIPS
- PARADE-dedicated-ARA-speedup

### PARADE speedup

- Deblur
- Denoise
- Registration
- Segmentation
- BlackScholes
- StreamCluster
- Swaptions
- LPCIP_Desc
- Texture_Synthesis
- Robot_Localization
- Disparity_Map
- EKF_SLAM

### Medical Imaging

- Commercial
- Vision
- Navigation

### gem5 KIPS

- 0
- 10
- 20
- 30
- 40
- 50
- 60
- 70

### PARADE speedup

- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7

15X
PARADE: Platform for ARA Design & Exploration

- Based on widely-used gem5 and support full-system X86
- Global Accelerator Management (GAM)
- Coherent cache/SPM with shared memory (Ruby)
- Customizable Network-on-Chip simulation (Garnet)
- Power/area simulation
  - McPAT for CPUs; high-level synthesis (AutoPilot) for accelerators
  - CACTI for caches; DSENT for NoC, Micron model for DRAM
- Visualization to assist design space exploration
Thank You!

For more information, please contact Zhenman Fang, zhenman@cs.ucla.edu
Tutorial website: http://accelerator.eecs.harvard.edu/isca15tutorial/