LACORE: A RISC-V BASED LINEAR ALGEBRA ACCELERATOR FOR SOC DESIGNS

Samuel Steffl and Sherief Reda

Brown University, Department of Computer Engineering

Partially funded by NSF grant 1438958

Published as "LACore: A Supercomputing-Like Linear Algebra Accelerator for SoC-Based Designs" at IEEE Conference on Computer Design, 2017

Lightweight Linear Algebra in HPC

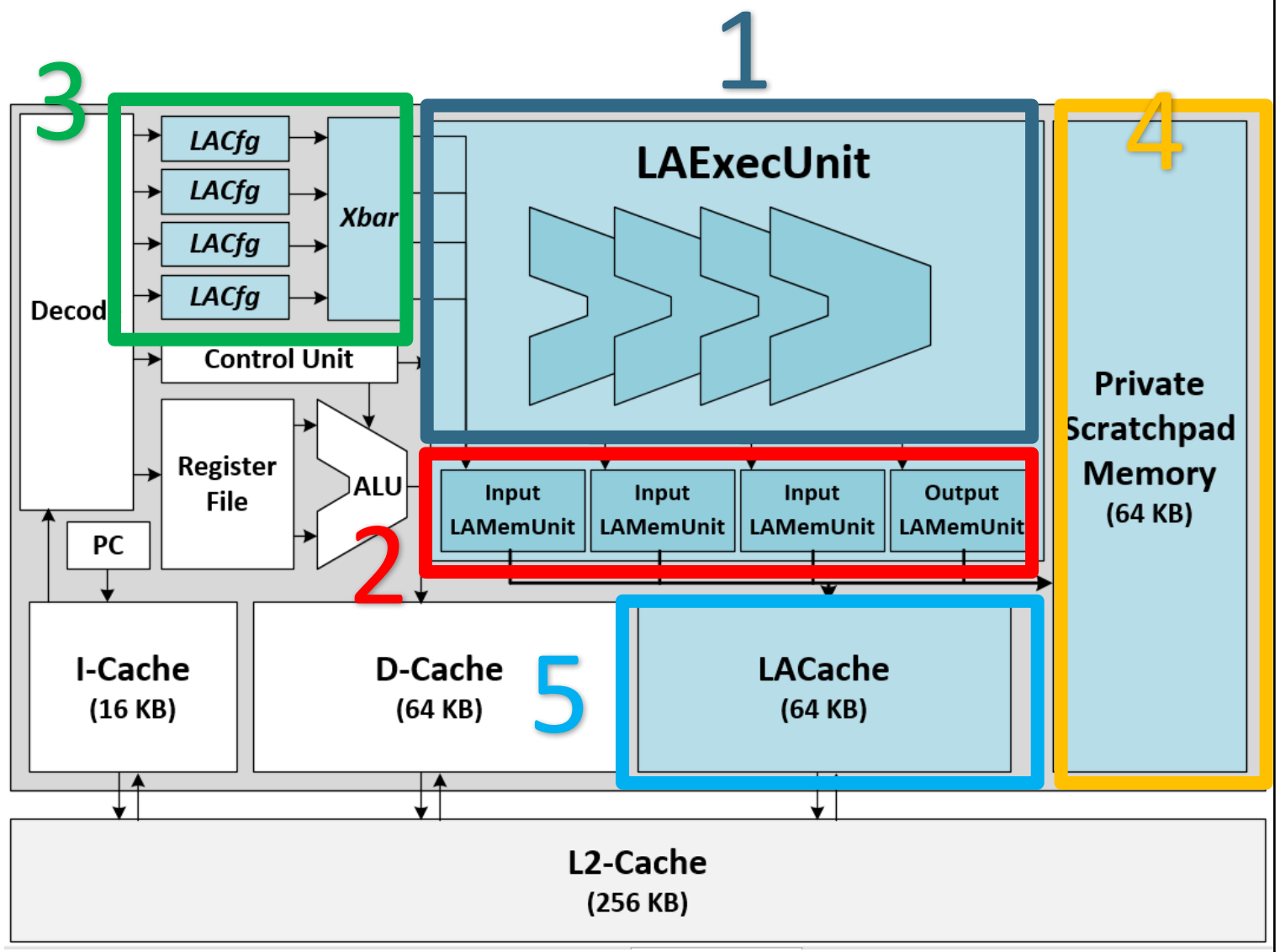
- Linear Algebra is a foundation of High Performance Computing (HPC)
- Most HPC apps can be reduced to a handful of computation classes
- Modern Accelerators have numerous shortcomings
  - **Lightweight Manycore (GPU):** memory transfers, poor sequential mode, synchronization bottleneck
  - **Fixed-Function Accelerators (FPGA/ASIC):** costly and limited applicability

- ✓ Sparse Linear Algebra
- ✓ Dense Linear Algebra
- ✓ FFT
- ✓ Structured Grids
- ✓ Unstructured Grids
- ✓ Machine Learning
- ✓ Artificial Intelligence

- Proposed Architecture: LACore
  - targets wide range of applications and overcome current hardware issues
- Design and Development of the LACore consisted of:
  1. **Architecture** and microarchitecture design
  2. **Instruction Set** (ISA) design and implementation in gcc
  3. Creation of C-Programming Framework, called **LACoreAPI**
  4. Implementation of the LACore in the **gem5** C++ simulator
  5. Evaluating the LACore with the **HPCC Benchmark Suite**

LACore Architecture: Overview

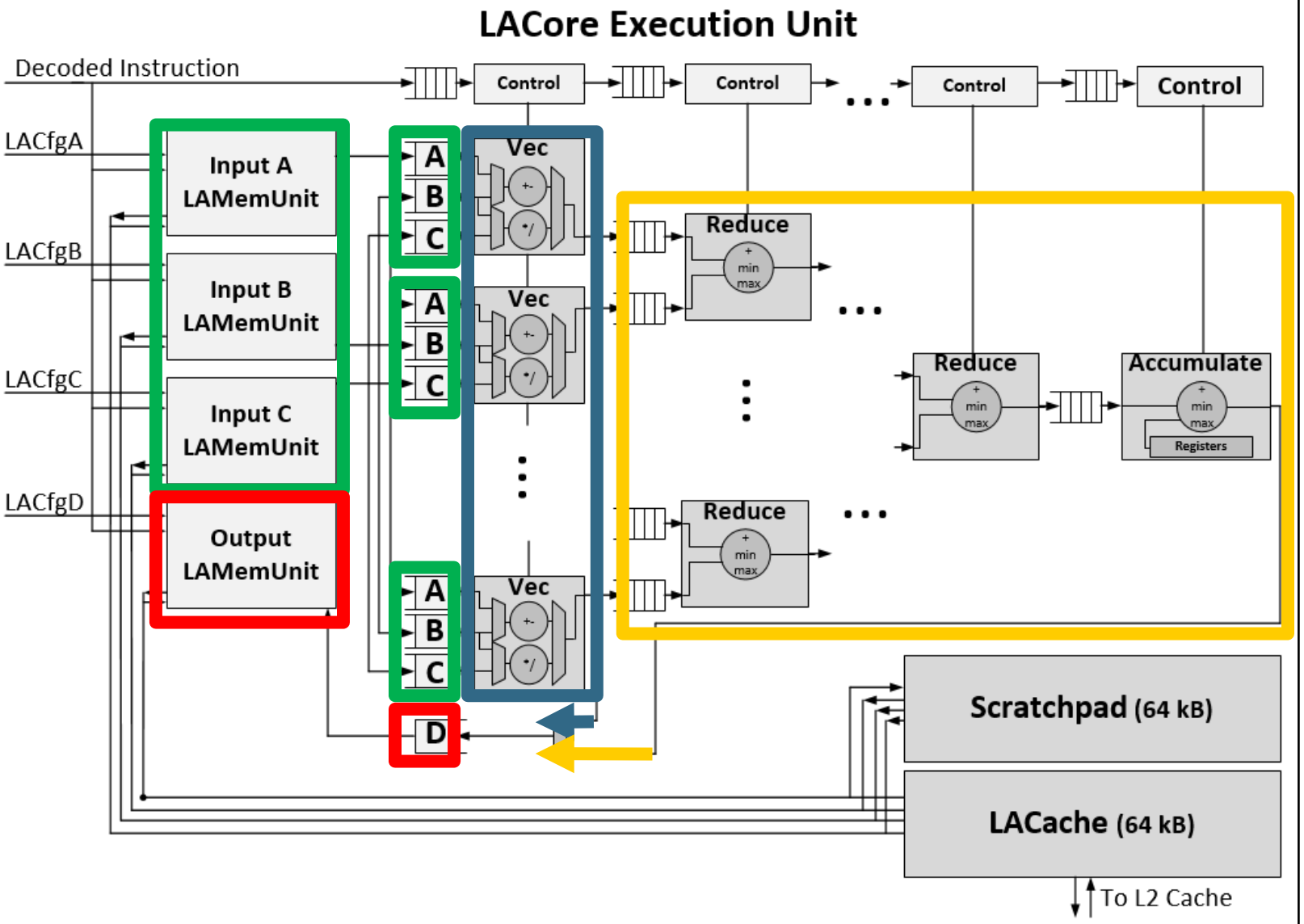
- LACore extension to scalar CPU
- Scalar CPU blocks in white
- LACore Pieces in blue:
  1. **LAExecUnit** is a mixed-precision systolic datapath connected to LAMemUnits with FIFOs
  2. **LAMemUnits** read and write data-streams to the datapath FIFOs and LACache/Scratchpad
  3. **LACfgs** provide configuration to LAMemUnits
  4. **64 kB Scratchpad** provides low-latency high-throughput temporary storage
  5. **64 kB LACache** with 4-ports and 16-banks is interface to the memory hierarchy



LACore Architecture: LAExecUnit

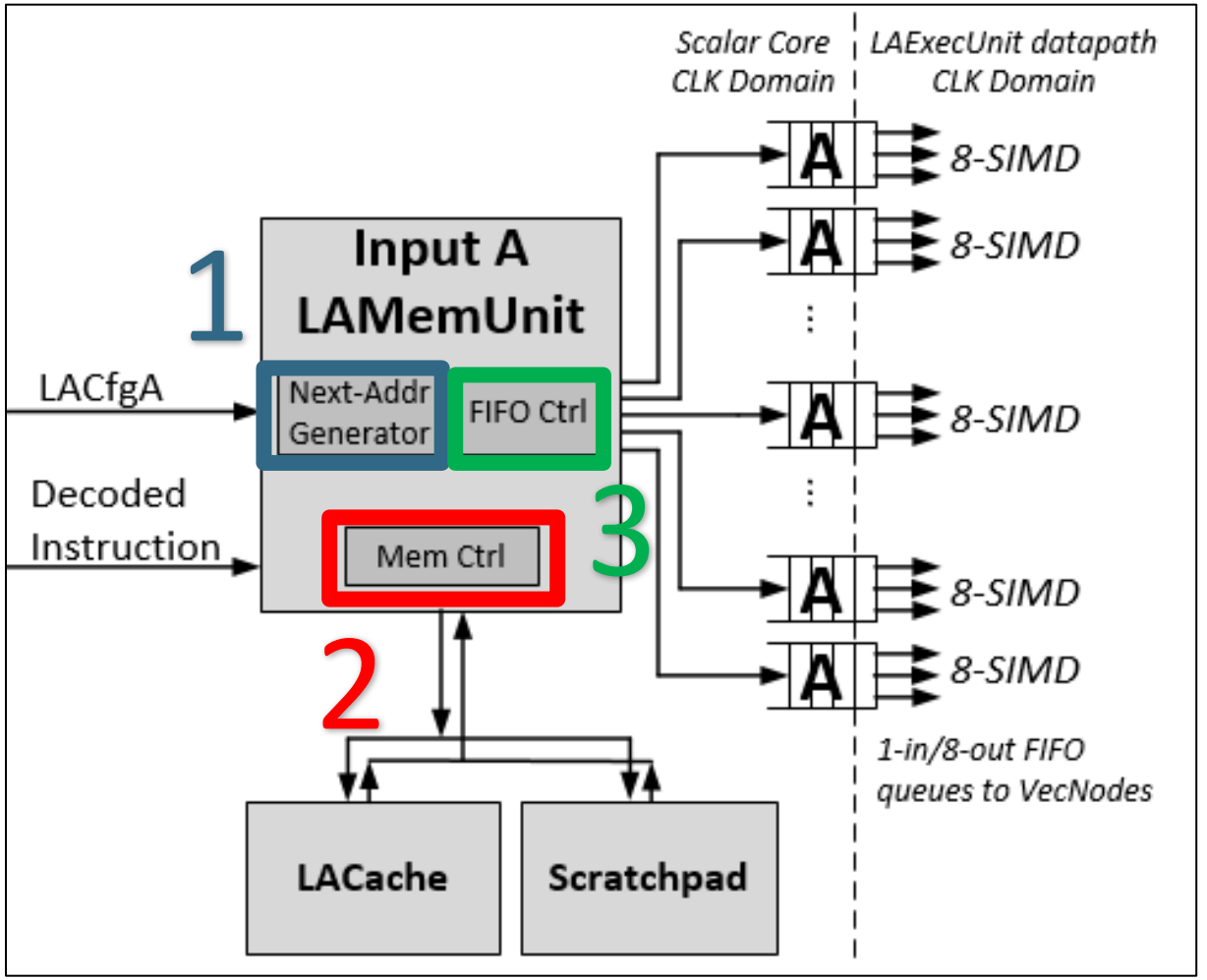
- 4-operands: 3 inputs (A,B,C) and 1 output (D)
- Dual Precision (32 and 64 bit)
- Systolic Datapath: connected to LAMemUnits through FIFOs
- **Vector Unit**: 8 Parallel VecNodes with 8 different configurations
- **Reduction Unit**: Binary tree with 7 ReduceNodes and 1 AccumulateNode, with 3 different configurations
- 24 datapath configurations

VecNode Ops		ReduceNode Ops
(A+B)*C	(A+B)/C	sum(X,Y)
(A-B)*C	(A-B)/C	min(X,Y)
(A*B)+C	(A*B)-C	max(X,Y)
(A/B)+C	(A/B)-C	



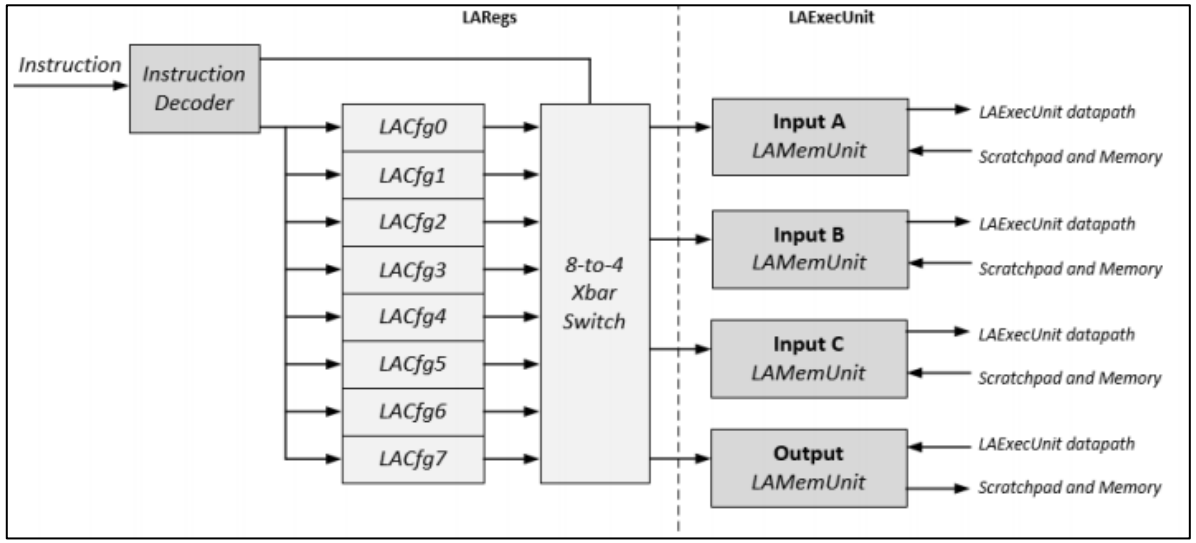
LACore Architecture: LAMemUnit

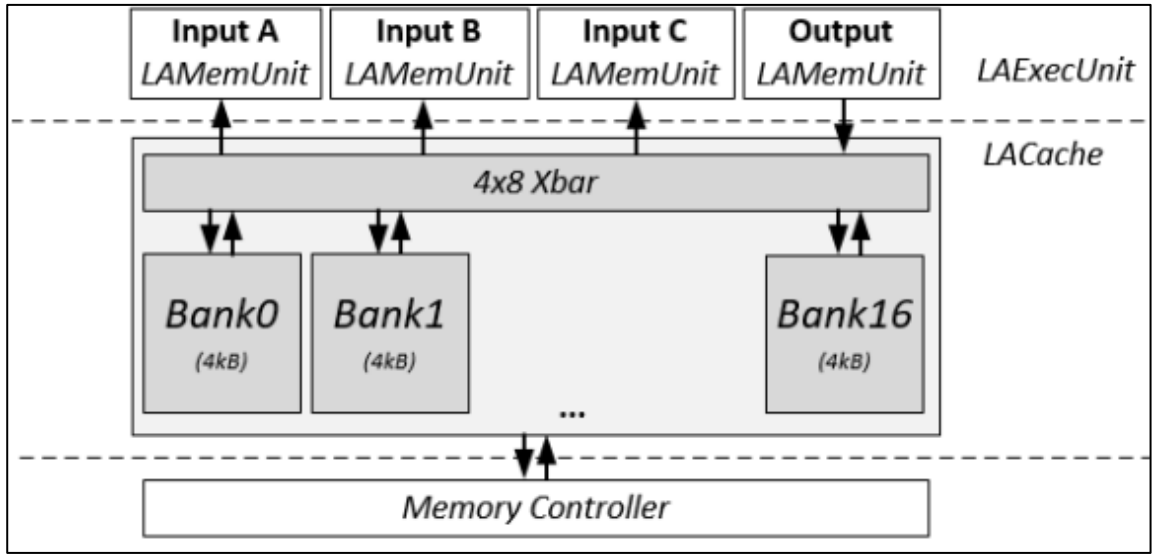
- Handle all memory/scratchpad interactions
- *streams* data into the LAExecUnit's datapath
- Can read or write scalars, vectors, matrices, and sparse-matrices
- Handles 32-bit and 64-bit floating point
- 3 main components:
  - **Next-Addr Generator** determines the location of the next item in the stream
  - **Memory Controller** handles the cache and scratchpad communication protocol
  - **FIFO Controller** reads and writes data to the FIFOs attached to the datapath



LACore Architecture: LACfg, LACache, Scratchpad

- 8 LACfg configuration registers
  - 294 bits per LACfg
  - Connected by crossbar switch to LAMemUnits to provide data-stream configuration
  - Hold all info about data-stream type, precision, location, etc…





- LACache
  - 64 kB with 128 Byte line size
  - 3 read ports and 1 write port
  - 16 line-interleaved banks
- Scratchpad
  - 64 kB with 128 Byte line size
  - Separate address space
  - 3 read ports and 1 write port

LACore Instruction Set and LACoreAPI

- RISC-V ISA Extension
  - 68 new 32-bit LACore instructions
  - full gcc toolchain support
  - 3 classes of instructions
    - configuration
    - data-movement
    - execution

- Programming Framework: LACoreAPI
  - C-programming framework
  - Contains 3 classes of APIs:
    - **Configuration**
    - **Data-Movement**
    - **Execution**

```
double *value;
LAAddr addr = (LAAddr)value;

// set scalar in an LACfg, scratch, or mem
la_set_scalar_dp_reg(0, *value);
la_set_scalar_dp_sch(1, 0x10);
la_set_scalar_dp_mem(2, addr);

// set vector in mem or scratch
la_set_vec_dp_mem(0, addr, stride, count, skip);
la_set_vec_adr_dp_mem(0, addr);
la_set_vec_adrcnt_dp_sch(0, addr, count);
```

```
// copies src->dst, convert double->float
void copy_d2s(double *src , float *dst , int cnt)
{
  la_vec_adr_dp_mem(0, (LAAddr)src);
  la_vec_adr_sp_mem(1, (LAAddr)dst);
  la_copy(1, 0, cnt);
}
```

```
//D = (A+B)*C, vector output
la_AaddBmulC(D, A, B, C, count);

//D = (A/B)-C, scalar output
la_AdivBsubC_sum(D, A, B, C, count);

//D = (A*B)+C, multi - stream output
la_AmulBaddC_sum_multi(D, A, B, C, count);
```

HPCC Evaluation in gem5

- Implemented LACore in gem5
  - industry standard cycle-accurate c++ system simulator
  - ~25,000 lines of C++/Python
- Implemented HPC Challenge (HPCC) Benchmark
  - DGEMM, FFT, PTRANS, STREAM, HPL and Random Access
- 3 other architectures evaluated for comparison
  - In-order pipelined RISC-V core
  - Superscalar x86 core with SSE2 extensions
  - Equivalent Fermi GPU with 2 Streaming Multiprocessors

Overall HPCC Benchmarks Suite

- LACore Outperforms x86 by average of **3.43x**
- LACore Outperforms RISC-V by average of **10.72x**
- LACore Outperforms GPU by average of **12.04x**

LACore has competitive area utilization

- Only **2.53x** the area of a single RISC-V scalar CPU
- Only **0.60x** the area of the equivalent GPU

	DGEMM	FFT	PTRANS	HPL	Random Access	STREAM - Triad
x86	5.73x	2.4x	4.14x	2.10x	0.7x	4.3x
RISC-V	31.7x	4.23x	1.52x	4.67x	1.0x	13x
GPU	1.6x	7.98x	1.98	-	0.3x	47x

LACore is Freely Available

- https://github.com/scale-lab/la-core
- In spirit of RISC-V open hardware and open-source gem5
- Installation guide
  - riscv-tools with LACore extension
  - cross-compiling GSL for RISC-V
  - building the HPCC benchmarks for LACore, RISC-V, x86 and CUDA
  - installing and building gem5 models for LACore, RISC-V, x86 and CUDA
  - How to extend LACore codebase
- Developer Docs
  - emphasis on effectively programming for the LACore ISA
  - provides full LaCoreAPI usage examples

LACore simulators and benchmarks

- https://github.com/scale-lab/la-core
- Multiple simulators available for LACore
  - available as reference for other RISC-V-based accelerators
  - riscv-isa-sim (aka ‘spike’): gold-standard RISC-V functional simulator
  - gem5 64-bit LACore models
    - AtomicLACoreSimpleCPU: purely functional
    - TimingLACoreSimpleCPU: single-cycle instructions + cycle-accurate memory access
    - MinorLACoreCPU: in-order pipelined CPU + cycle accurate memory access
  - other gem5 models
    - gem5-gpu CUDA model, in-order pipelined RISC-V model, out-of-order x86 model
- Full HPCC Benchmarks for all platforms available
  - full DGEMM, FFT, PTRANS, STREAM, HPL and Random Access kernels
  - easily reproduce the benchmark results in this presentation
  - use as a reference for other applications targeting the LACore

Conclusions and Future Work

- Conclusions
  - LACore is a Large-Format vector accelerator for a broad range of Linear Algebra applications
  - LACore has novel architectural features including as the:
    - configurable, data-streaming LAMemUnits
    - dual-precision, configurable, systolic LAExecUnit
  - A compiler toolchain, programming framework and architectural simulator were all implemented for the LACore
  - The LACore significantly outperforms the x86, RISC-V and GPU for linear algebra applications
  - LACore is open-source
- Future work
  - LACore multi-core design and evaluation
  - LACore ASIC implementation and eventual tapeout

Thank You

Questions?