LACore: A Supercomputing-Like Linear Algebra Accelerator for SoC-Based Designs

Samuel Steffl, Sherief Reda
Department of CE, Brown University, Providence, USA
{*samuel_steffl, sherief_reda*}*@brown.edu*

***Abstract*—Linear algebra operations are at the heart of scientific computing solvers, machine learning and artificial intelligence. In this paper, `LACore`, a novel, programmable accelerator architecture for general-purpose linear algebra applications, is presented. `LACore` enables many of the architectural features typically available in custom supercomputing machines in an accelerator form factor that can be deployed in System-On-a-Chip (SoC) based designs. LACore has several architectural features including heterogeneous data-streaming `LAMemUnits`, a configurable systolic datapath that supports scalar, vector and multi-stream output modes, and a decoupled architecture that overlap memory transfer and execution. To evaluate `LACore`, we implemented its architecture as an extension to the RISC-V ISA in the gem5 cycle-accurate simulator. The `LACore` ISA was implemented in gcc, and a C-programming software framework, the `LACoreAPI`, has been developed for high-level programming of the `LACore`. Using the HPCC benchmark suite, we compare our `LACore` architecture against three other platforms: an in-order RISC-V CPU, a superscalar x86 CPU with SSE2, and a scaled NVIDIA Fermi GPU. The `LACore` outperforms the superscalar x86 processor in the benchmark suite by an average of 3.43x, and outperforms the scaled Fermi GPU by an average of 12.04x, within the same or less design area.**

***Keywords*-Linear Algebra, Heterogeneous Computing, Accelerator Architectures**

I. Introduction

Modern hardware solutions to linear algebra applications include manycore processors such as GPUs, vector extensions to scalar CPUs such as Intel's SSE/AVX extensions, and custom ASIC or FPGA solutions. There are various shortcomings with each of these platforms that the `LACore` addresses through its novel architecture.

GPU Issues: One issue with GPU usage for HPC applications is that GPU architectures inherently lack the ability to reduce data without thread synchronization. This can act as a bottleneck for common kernels such as DGEMM, which is a sequence of many dot-products. An architecture that natively supports arbitrarily large vector reduction without explicit thread synchronization would have a performance advantage.

Additionally, GPUs are built for parallel processing only, and require a high-performance single-threaded CPU to handle the sequential portions of applications. Having a single, powerful thread that can switch between parallel and sequential mode can overcome the inefficiencies in the GPU fork-join model. Although x86 with AVX extensions may appear to support this model, AVX does not fully support arbitrary vector sizes and is not as parallelizable as a GPU.

Finally, GPUs use separate device memory, which requires transfer of data between the host and device before and after kernel execution. Although the latency can be partially hidden by overlapping kernel execution with data transfer, an architecture that does not require data movement between sequential and parallel execution modes would be better.

Application-Specific Accelerator Issues: Although custom ASIC/FPGA solutions typically provide the highest performance for a particular application, they are rigid, can have too narrow of scope, and have a high development cost. A better architecture would offer the same custom hardware benefits as ASICs while still being programmable for solutions to a wide range of linear algebra applications.

Traditional Supercomputer Issues: Traditional Supercomputers have configurable memory-memory linear algebra accelerator architectures similar to the `LACore`, but they are typically monolithic, warehouse-bound designs which are not suitable for the increasingly mobile-computing and Internet-of-Things trends in the industry today. A more compact design providing similar features to those supercomputers is necessary in today's computing environment.

Proposed LACore architecture: The goal of the LACore architecture is to bring many of the linear algebra computing capabilities typically reserved for supercomputers in an accelerator form factor that can be deployed in modern SoC designs. The `LACore` architecture incorporates the following novel combination of architectural features:

- **heterogeneous** data-streaming `LAMemUnits` capable of accessing scalar, vector, and sparse matrix objects in both scratchpad and memory;
- **large-format vector** support, providing an interface to work with arbitrarily-large vectors and matrices;
- **configurable systolic vector-reduction** datapath within the `LAExecUnit`, implementing up to 24 different functions on 3 mixed-precision input data-streams;
- **decoupled** architecture that overlap data-execution with memory-access during complex memory-memory instructions;
- **multi-stream**, vector, and scalar output modes, with Multi-Stream mode reducing multiple sub-vectors within larger input vector streams for enhanced data-execution and memory-access overlap; and
- **embedded vector processing** inside a lightweight RISC-V scalar CPU which brings linear algebra supercomputing to area-constrained applications

To evaluate `LACore`, we implemented its architecture as an extension to the RISC-V ISA in the gem5 cycle-accurate

Funding: This work is partially supported by NSF grant #1438958.

1063-6404/17 $31.00 © 2017 IEEE
DOI 10.1109/ICCD.2017.29

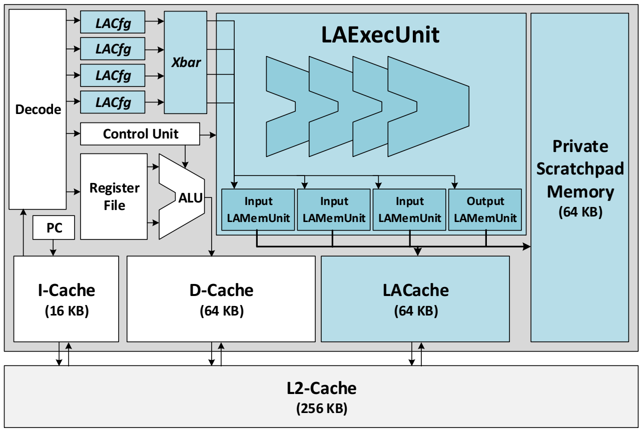

Figure 1: `LACore` architecture overview, with scalar CPU components in white and the `LACore`-specific components in blue.

simulator. The LACore ISA was implemented in gcc, and a C-programming software framework, the `LACoreAPI`, has been developed for high-level programming of the LACore. Using the HPCC benchmark suite, we compare our `LACore` architecture against three other platforms: an in-order RISC-V CPU, a superscalar x86 CPU with SSE2, and a scaled NVIDIA Fermi GPU. The LACore outperforms the superscalar x86 processor in the benchmark suite by an arithmetic average of 3.43x, and outperforms the scaled Fermi GPU by an average of 12.04x, while using less area.

II. LACore Architecture

Figure 1 illustrates the high-level microarchitecture which consists of a scalar CPU and an `LACore` acceleration unit. The gray box represents a single CPU, which contains scalar processor components highlighted in white, and `LACore` extension components highlighted in blue. In this work, a RISC-V processor was used as the scalar CPU implementation. The `LACore` accelerator consists of configuration registers (the `LACfgs`), an `LACore` execution unit (the `LAExecUnit`), a 64 kB per-core private scratchpad memory, and a high-throughput multi-banked cache (the `LACache`).

The scalar CPU's ISA is extended with `LACore`'s instructions, which are decoded by a modified scalar Decode Unit, providing configuration for the `LACfgs`, `LAExecUnit` and `LAMemUnits`. The `LACore`'s Execution instructions are 4-operand (3 inputs and 1 output) complex memory-memory operations (where memory could be main memory or scratchpad) and will execute for a variable amount of processor-cycles, concurrently accessing data in memory and scratchpad through the `LAMemUnits`, and executing arithmetic operations in the `LAExecUnit`'s datapath. The `LAMemUnits` and the datapath are connected by FIFOs.

The `LACfg` registers configure the four `LAMemUnits` for accessing data in memory or scratchpad. The `LAMemUnits` use these configurations to *stream* data between the `LAExecUnit` and the memory or scratchpad. The scratchpad is a low-latency, high-throughput memory used to store intermediate results of multi-instruction operations in the `LAExecUnit`. The `LACache` is a third L1 cache alongside the D-Cache and the I-Cache, with multiple ports and banks for the `LAMemUnits` to effectively access the main memory.

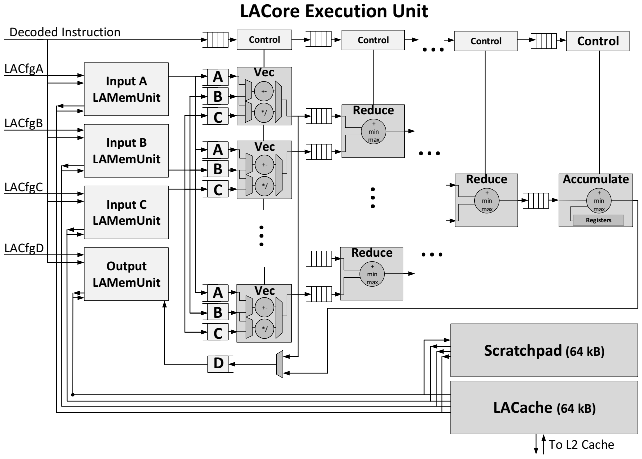

Figure 2: Scaled-Down `LAExecUnit` architecture, with the 4 `LAMemUnits`, 3 of the 8 `VecNodes`, 3 of the 7 `ReduceNodes` and the 1 `AccumulateNode` shown. The `LAMemUnits` are configured by `LACfgs`, and read or write to the memory or scratchpad.

A. The LAExecUnit

The `LAExecUnit` is a highly-configurable parallel processing unit with two major types of sub-units: the datapath, and the `LAMemUnits` as illustrated in Figure 2. There are four `LAMemUnits`, three which read large-format streams of data from the scratchpad, the memory or an `LACfg`, and write them into the datapath's input FIFOs `A`, `B` and `C`. The fourth `LAMemUnit` reads the datapath's output FIFO, `D`, and writes the resulting large-format data-stream to the scratchpad or memory. Similar to many supercomputing architectures, the `LACore` is a *Decoupled Access/Execute Architecture* [18], where the memory access and data execution are separate processing units connected by FIFOs, reminiscent of the design presented in [12].

1) The Configurable LAExecUnit Datapath

The datapath portion of the `LAExecUnit` is composed of `VecNodes` and `ReduceNodes`. Each `VecNode` operates on 8-wide double-precision or 16-wide single-precision packed SIMD buffers. Each `VecNode` reads its three input data-streams from their individual `A`, `B` and `C` FIFOs, and writes the output data stream to a FIFO going into the `ReduceNodes`. The `VecNodes` can be configured for 8 different operations given in Table I. The `ReduceNodes` read two input SIMD streams from FIFOs and produce an output SIMD stream, and can be configured for three different reduction operations, also shown in Table I. The full datapath can therefore be configured for 24 different functions. A single `AccumulateNode` is at the end of the `ReduceNode` tree, and performs similar reduction functions as the `ReduceNodes` while also accumulating stream results. The `LAExecUnit`'s datapath is composed of

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

Table I: The 8 possible VecNode configurations and the 3 possible ReduceNode configurations. There are 24 operations possible when operating in single-output or multi-stream-output mode, and only 8 operations possible for vector-output mode.

dual-precision processing elements, and also has the ability to operate on both single-precision and double-precision input streams simultaneously.

The LAExecUnit's datapath also features three configurable output-stream modes: vector, scalar, or multi-stream:

- **vector-output:** The output stream is taken directly from the outputs of the VecNodes. An example use is vector-vector addition.
- **scalar-output:** The ReduceNodes are active and produce a single output. An example use is vector dot-product.
- **multi-stream output:** Takes input streams with N elements and reduce them to an output stream with M elements, where M divides N. An example use is the DGEMM kernel.

We designed the LAExecUnit's datapath for maximum performance across a range of linear algebra applications and problem sizes using parameter sweeps in gem5. We found the optimal number of VecNodes and ReduceNodes in the datapath were 8 and 7 respsectively, and the optimal SIMD-width is 512-bytes, or 8 double-precision elements.

The LAExecUnit's peak theoretical throughput with the optimal VecNode and SIMD-width configuration is 252 FLOP/cycle for scalar-output and multi-stream output modes, and 128 FLOP/cycle for vector-output mode.

2) The LAMemUnits

The LAMemUnits are responsible for taking the wide range of data configurations and converting them into a universal stream format for the datapath. The high-level architecture of the LAMemUnit consists of 1) a FSM that determines the memory, scratchpad or LACfg register location of the next element in the stream, 2) a FIFO controller that distributes the stream data to the VecNode FIFOs accordingly, and 3) a Memory-Controller, that sends requests to the LACache and scratchpad using 128-byte line sizes for each.

A distinguishing feature of the LACore is that the LAMemUnits provide an abstracted stream interface that allows heterogeneous data sources to be used simultaneously. For example, a sparse matrix could be multiplied by a scalar and added to a regularly strided vector in memory and the result be written into a dense-matrix panel in the scratchpad. In each of these cases, the physical vectors and matrices in memory or scratchpad are converted into a unified *stream* format, which can all be handled identically by the datapath. The configuration for each of the data-streams is done by configuring LACfgs for each scalar, vector or matrix. This configuration is read by the LAMemUnit to determine how to convert the scalar, vector or matrix into the universal data-stream format.

Additionally, the LAMemUnits support *large-format* data sources, meaning that input and output vectors/matrices can be arbitrarily large, similar to archetypal vector processors [17]. The major distinction from vector architectures is that in the LACore, data sources are not limited to strided vectors.

The LAMemUnits support data-streams composed of either single-precision or double-precision floating point numbers, which can be composed into scalars, vectors or sparse-matrices, and can be located in any of the LACfgs, scratchpad or memory. Vector configurations use a stride, skip, count configuration, commonly known as a gather-scatter interface, which is a common paradign in vector-like processors such as [12] [17][5] [4].

Sparse-matrix configuration is a novel feature of the LAMemUnit that allows reading and writing to arbitrary offsets within a sparse matrix stored in a Harwell-Boeing-like format. The main distinction is that the format supported by the LAMemUnit does not assume column-major or row-major format — it can be either.

B. The LACfgs

The LACfgs are 294-bit registers that hold data-stream configuration used by the LAMemUnits. Each LACfg can hold either scalar, vector or sparse-matrix configuration. Fields within the LACfgs can generally be grouped into four categories: 1) data fields, 2) layout fields, 3) location fields, and 4) flags.

There are eight LACfgs connected by a crossbar to the four LAMemUnits. Execution and data-transfer instructions specify three data sources and one destination for the result to configure the crossbar to connect the LAMemUnits to the correct LACfg. The crossbar's location between the LACfgs and LAMemUnits can be viewed in Figure 1. The decision to use eight LACfgs with a crossbar configuration instead of directly connecting four LACfgs to four LAMemUnit enables setting up data-streams ahead-of-time, and then reuse them multiple times later without having to reconfigure that particular LACfg.

C. Scratchpad and LACache

The last major architectural features of the LACore are the 64 kB scratchpad and the 64 kB LACache. Scratchpads are low-latency, high-throughput memories and are common in embedded applications and GPUs [2]. The purpose of the LACore's scratchpad is to provide temporary storage for intermediate results in multi-instruction applications. The LACore's scratchpad is a single bank, multi-port design; it has 3 independent-reads and 1 independent-write port, with 128-byte lines per access. This constrasts the highly-banked scratchpads found in GPUs, where high banking is needed to reduce conflicts from the many lightweight threads accessing it simultaneously. In the LACore's case, there are only four LAMemUnits accessing the scratchpad simultaneously, and since they are each on independent ports, banking is not necessary.

The scratchpad has its own private address space starting at address 0x0, which contrasts the design approach presented in [2] since it gives the programmer explicit control over data movement to and from the scratchpad, which is useful for deterministic behaviour in high-performance applications.

In addition to the scratchpad, the LACore exclusively uses a 4-port, 16-bank L1-cache, called the LACache, to access the main memory. Alternative cache configurations to using the LACache are 1) routing all LAMemUnit requests through the L1 Data-cache, and 2) directly connecting the LAMemUnits to the L2-cache, similar to the approach in [12]. These alternatives were tested alongside the LACache configuration in extensive gem5 parameter sweeps across our benchmark suite, and we found the LACache to provide the best performance on average. Similarly, the sweeps were used to determine that 16 banks within the LACache provided better performance on average than fewer banks.

III. LACore Programming Model

A. The LACore ISA

The full LACore extension to the RISC-V instruction set consists of 68 new instructions using the Custom-0 extension space. RISC-V was chosen as a vehicle for the LACore architecture instead of other ISAs such as MIPS, ARM or x86, for a few reasons: 1) RISC-V is a modern, free, and open-source platform, with robust toolchains for developing both hardware and software, 2) a gem5 implementation and fully tested RISC-V RTL models exist for many CPU types, and 3) RISC-V provides a light-weight scalar CPU footprint. There are three main classes of instructions in the LACore extension: configuration, data movement, and execution.

Configuration Instructions: The configuration instructions are single-cycle instructions that update a specified destination LACfg register with contents from general-purpose integer or float registers from RISC-V scalar CPU register files. The destination data stream could be in the scratchpad or memory, or the data stream could be a scalar floating point value stored in the LACfg itself. Additionally, all scalars, vectors or sparse stream configurations can be single or double-precision.

The scratchpad and memory address spaces are non-unified, so the address 0x10 in the scratchpad refers to byte offset 0x10 in the scratchpad, while memory addresses refer to normal memory address space.

Data Transfer Instructions: Data transfer instructions copy data-streams between any combination of memory, scratchpad and an LACfg. Their functionality is similar to memcpy() but with additional mechanisms to 1) convert between single-precision and double-precision streams, 2) transform streams between scalars, vectors, and sparse matrices, and 3) move data between scratchpad and memory.

Data Execution Instructions: Execution instructions simultaneously configure the LAMemUnits to stream input and output data into the LAExecUnit's datapath, and configure the datapath to perform one of the 24 possible operations on the data streams.

Execution instructions come in three output-modes: vector-output, scalar-output, and multi-stream output. There are eight variations of vector-output instructions, and 24 variations of scalar-output and multi-stream output instructions. The variations are used to configure the various muxes and control circuitry in the LAExecUnit for the duration of the instruction. The three input data-streams and one output data-stream are specified by instruction arguments while the total number of input elements to operate on is specified by an additional instruction argument.

B. The LACoreAPI Framework

The LACore's ISA was implemented in gcc, and a C programming, header-only library called the LACoreAPI was developed to raise the abstraction of programming the LACore from the assembly level to C. All benchmarks and programs targeting the LACore thus far have been built on top of the LACoreAPI. The API provides configuration, data movement and execution function calls, similar to the LACore's instruction subsets.

IV. Benchmarks and Evaluation

A. Benchmark Methodology

HPC Challenge benchmark suite [13] was used to compare the performance of the LACore against three different architectures: a simple in-order pipelined RISC-V processor, a superscalar x86 processor with SSE2 enabled, and NVIDIA Fermi GPU with two Streaming Multiprocessors (SMs). All platforms were simulated on gem5 [3] in syscall-emulation mode, with the Fermi GPU simulated with gem5-gpu [15].

The HPCC benchmark suite was selected to evaluate our platform primarily because it is designed for high performance linear algebra computations. It contains seven benchmarks that stress computation throughput, memory bandwidth, and communication bandwidth, which cover a broad range of spatial and temporal data localities, making it an optimal suite to evaluate the LACore. The motivation for using a cycle-accurate simulator for benchmarking the x86 and GPU platforms instead of using real hardware is to provide a fairer comparison by removing process technology, clock speeds and cache configuration as variables, in order to focus on the merits of the processor architectures.

One thread was used in the LACore, RISC-V and x86 implementations in order to compare the single threaded performance of each of these three platforms. This provides the baseline performance for each processor's area and power consumption. Similarly, the GPU was run using only a single *cluster*, with 2 cores per *cluster*. A *cluster* is gpgpu-sim's equivalent of a collection of NVIDIA Streaming Multiprocessors. Only two NVIDIA SMs were modelled since this is a fair comparison to an LACore in terms of area, as well as power since area correlates directly with dynamic power through the capacitance, as discussed in Section IV-C.

LACore		x86,RISC-V		Fermi GPU	
Scalar CPU Model	In-Order	x86 CPU Model	Out-of-Order	Scalar CPU Model	Out-of-Order
Scalar CPU Clock	3 GHz	RISC-V CPU Model	In-Order	GPU Model	NVIDIA Fermi
LAEX Clock/Vecs/SIMD	1 GHz/8/8	Scalar CPU Clock	3 GHz	CPU/GPU Clock	3 GHz/1GHz
SPD Size(kB)/Line(B)	64,128	Cache Line(B)	128	GPU Clusters	1
Cache Line(B)	128	I$,D$,L2$ Sizes(kB)	16,64,256	GPU Cores-per-Cluster	2
I$,D$,L2$ Sizes(kB)	16,64,256			Warp-Size	32
LA$ Size(kB),Banks	64,16			Cache Line(B)	128
				L2$ Size (kB)	1024

Table II: gem5 and gem5-gpu configurations used for the HPCC benchmark suite. CPU clocks and caches were set to similar values. The LAExecUnit (LAEX) and Fermi SMs used a 1 GHz clock domain. The LACore's scratchpad (SPD) and LACache (LA$) are both 64 kB and use 128-byte lines. The LAEX had 8 VecNodes for 8-wide SIMD data inputs.

The gem5 configuration for the LACore, RISC-V, x86 and scaled-down Fermi GPU platforms are shown in Table II.

The HPCC benchmarks used for evaluation were DGEMM, FFT, PTRANS, HPL, and STREAM. This covers the four extreme combinations of high-and-low spatial-and-temporal locality. b_eff and Random Access were not tested since they evaluated multi-processor communication efficiency and integer workload efficiency respectively.

The standard HPCC software distribution was not used for the benchmarks. Instead, kernels tailored for each of the platforms were either hand-written, or used high-performance libraries, such as GNU Scientific Library, FFTW3, and Eigen [7][6] [8]. There are a few reasons for not using the standard distribution of HPCC: 1) gem5's syscall-implementation mode does not cover all syscalls, or all functionality of every ISA it implements, so many of the compiled x86 benchmarks cannot run; 2) LACore and CUDA are special platforms that do not have hand-tailored kernels in the HPCC distribution, so they need to be written anyways; 3) this paper evaluates single-threaded functionality for x86, RISC-V and LACore, so greater control is needed over exactly how the benchmarks are internally running; and 4) tailoring the benchmarks for each individual platform instead of using a generic benchmark allows each platform to be fully utilized and acheive maximum performance. Care was taken to follow the specification for each benchmark described in [13] as closely as possible, in order to provide reproducible results. For all benchmarks, we verified the LACore results to be correct against correct implementations.

All compiler, simulator, and benchmark source code for the LACore can be found at https://github.com/scale-lab/la-core.

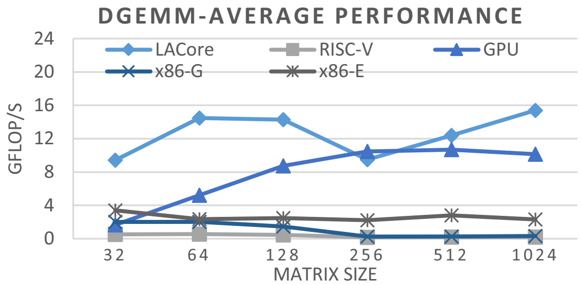

Figure 3: DGEMM Results for LACore, RISC-V, scaled Fermi GPU (GPU), x86-GSL (x86-G) and x86-Eigen (x86-E).

B. Bechmark Evaluation

DGEMM: Double-Precision Dense Matrix-Matrix multiply is at the heart of many linear algebra applications, and is a high spatial-and-temporal locality application. For this paper, all 4 variations of $C = \alpha \cdot op(A) \cdot op(B) + \beta \cdot C$ were evaluated on the platforms. The RISC-V platform used GSL, while the x86 version used GSL and Eigen library implementations. The Fermi GPU ran optimized hand-written kernels since cuBLAS was not available as a static library for NVIDIA Toolkit 3.2 and no implementations for all 4 DGEMM variations that could run on gem5-gpu were found.

The averaged performances of all four DGEMM variations are shown in Figure 3. The LACore's performance asymptotically increases with the matrix size. At a matrix dimension of 128, the LACore achieves an average 14.3 GFLOP/s, which is a speedup of 31.7x, 1.6x, 9.8x and 5.73x over RISC-V, Fermi GPU, x86-GSL and x86-Eigen implementations. At a dimension of 1024, the average speedup is 80.4x, 1.52x, 47.3x and 6.7x respectively.

The large speedup over the RISC-V and x86 implementations is expected since DGEMM is a highly computation-bound application, and the LACore can keep its LAExecUnit constantly busy since memory accesses are regularly-strided with high localities. The performance speedup over the Fermi GPU can be explained by the LACore not having to transfer data to and from device memory, while still benefiting from similar parallelism that the GPU provides. Additionally, the LACore does not require synchronization during the reduction phase of a dot-product, while the Fermi GPU requires calls to syncthreads() after each iteration.

FFT: The complex double-precision FFT tests computational throughput for low spatial-locality, high temporal-locality applications. The RISC-V cpu used GSL, and the x86 cpu used GSL and FFTW. Since cuFFT was not available as a static library for NVIDIA Toolkit 3.2, a slightly modified FFT implementation was taken from the University of Illinois' Parboil benchmark suite.

The complex double-precision FFT results are shown Figure 4, with the LACore significantly outperforming all other implementations for vector sizes up to 2^{14} (or 16384) elements. The LACore's peak double-precision throughput at a vector size of 4096 is 1.88 GFLOP/s, which is a 4.23x, 7.98x, 2.40x and 3.34x speedup over the RISC-V, Fermi GPU, x86-GSL and x86-FFTW implementations. The

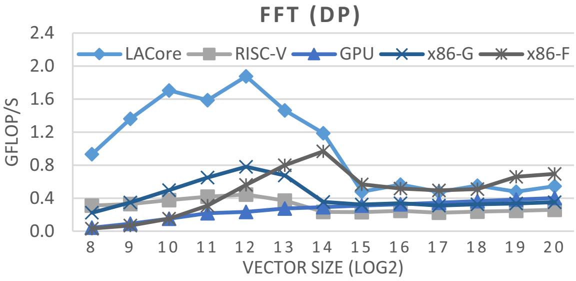

Figure 4: FFT Results for `LACore`, RISC-V, scaled Fermi GPU (GPU), x86-GSL (x86-G) and x86-FFTW (x86-F).

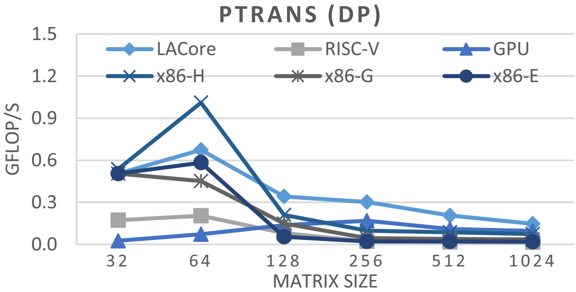

Figure 5: PTRANS Results for `LACore`, RISC-V, scaled Fermi GPU (GPU), x86-Hand-Written (x86-H), x86-GSL (x86-G) and x86-Eigen (x86-E).

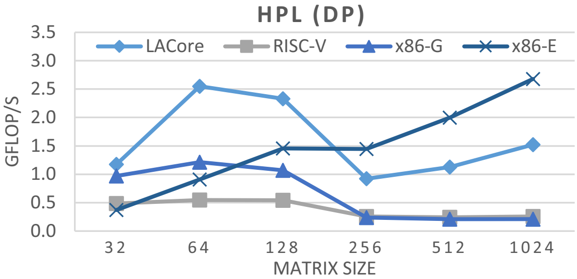

Figure 6: HPL Results for `LACore`, RISC-V, x86-GSL (x86-G) and x86-Eigen (x86-E).

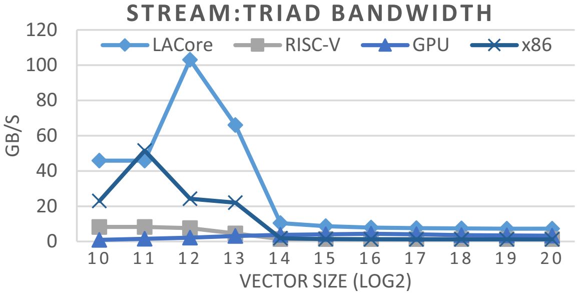

Figure 7: STREAM Triad Results for `LACore`, RISC-V, scaled Fermi GPU (GPU), and x86.

throughput asymptotically approaches 0.55 GFLOP/s, which is a 2.1x, 1.36x, 1.56x and 0.79x speedup of the RISC-V, Fermi GPU, x86-GSL and x86-FFTW implementations. So the `LACore` is the optimal architecture for small to medium-sized FFTs and nearly as good as the highly-tuned FFTW framework at large-sized FFTs, which implements higher-radix transforms than the simplest Radix-2 [6]. The small performance difference at large input vectors can be explained by the higher-sophistication of the FFTW algorithms than what the `LACore` was running.

PTRANS: Double-Precision Transpose and Add is a high spatial-locality and low temporal-locality application. The RISC-V platform used GSL, while the x86 platform used a hand-written kernel, the GSL library, and the Eigen library. The Fermi GPU ran optimized hand-written kernels, since cuBLAS was not available for NVIDIA Toolkit 3.2.

The PTRANS results are shown in Figure 5, with the `LACore` outperforming all other implementations at large matrix sizes, asymptotically reaching 150 MFLOP/s, with a speedup of 1.52x, 1.98x, 4.14x and 7.48x over the RISC-V, Fermi GPU, x86-Hand-Written, x86-GSL and x86-Eigen implementations. The `LACore` achieve modest speedups over other platforms since it can stores temporary sub-matrices in the scratchpad, while its `LAMemUnits` can effectively use consecutively strided elements in memory. Both features give this application on the `LACore` a higher effective FLOP-to-byte ratio, which allows better utilization of its datapath.

HPL: The High-Performance Linpack solves a linear system and is a high temporal and spatial locality test, similar to DGEMM. The RISC-V platform used GSL, and the x86 platform used two implementations, GSL and Eigen. Since cuSOLVER was not available as a static library for NVIDIA Toolkit 3.2, and no implementations could be found for GPU Linear-System Solve, the scaled Fermi GPU platform was not evaluated for this benchmark.

The HPL results are shown in Figure 6, with the `LACore` outperforming all other implementations for matrix sizes up to 2^8 (or 256) elements, at which point, x86-Eigen outperforms the `LACore`. The `LACore`'s peak double-precision throughput at a matrix size of 64 is 2.55 GFLOP/s, which is a 4.67x, 2.10x, and 2.81x speedup over the RISC-V, x86-GSL and x86-Eigen implementations. The `LACore`'s throughput asymptotically increases as the matrix size grows, with a throughput of 1.52 GFLOP/s at a 1024 matrix size, which is a 5.88x, 7.22x and 0.57x speedup of the RISC-V, x86-GSL and x86-Eigen implementations.

The `LACore` outperforms other platforms at small-to-medium sized HPL workloads because it directly swaps rows during the permutation steps within the LU factorization, instead of storing a permutation vector. This allows the `LACore` to better utilize its streaming hardware during the forward and backward substitution. However, this row-swapping becomes less efficient as the problem size grows, since it strains the cache hierarchy, so the `LACore`'s performance slightly suffers compared to the x86 implementation.

STREAM Triad: This benchmark is a high-bandwidth workload with high-spatial locatity and low-temporal locality, since each element in the vectors is accessed sequentially one time. The RISC-V and x86 platforms ran no libraries for STREAM Triad. The Fermi GPU implementation offloaded the vector computation as device kernels.

The STREAM Triad bandwidth results are shown in Figure 7. The GB/s for Triad is $3 \cdot VectorSize$ and the `LACore`

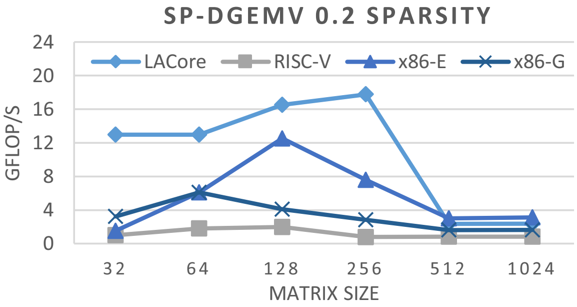

Figure 8: SpMV Results for LACore, RISC-V, x86-Eigen and x86-GSL.

peaks at 103 GB/s for a vector size of 4096, where it has 13x, 47x and 4.3x speedups over RISC-V, CUDA and x86 implementations. As the vector size grows, the LACore asymptotically approaches 7.25 GB/s, with speedups of 5.2x, 2.2x and 5.8x over RISC-V, Fermi GPU and x86 implementations.

One reason why LACore outperforms the other platforms so significantly is due to its large-format vector-like architecture with the specialized streaming LAMemUnits, which can access multiple consecutive elements in a cache-line in the same cycle. In other words, the LACore can exploit the high spatial-locality of STREAM benchmarks.

SpGEMV: Sparse DGEMV (SpMV) has an irregular memory-access pattern and a relatively high arithmetic complexity. It was chosen to evaluate the LACore because the HPCC benchmark suite does not have any sparse applications, and the sparse-matrix data-source configuration is a major feature of the LAMemUnits. The performance of SpMV on the LACore was compared to three other implementations: a RISC-V in-order CPU using the GSL library, and an x86 with SSE2 enabled using both Eigen and GSL implementations. Since cuSPARSE was not available as a static library for NVIDIA Toolkit 3.2, and no implementations could be found for GPU Sparse DGEMV, the scaled Fermi GPU platform was not evaluated for this benchmark.

At a 20% sparsity, the LACore achieves a peak performance of 17.8 GFLOP/s at a matrix size of 256, which is a 22.4x, 2.3x and 6.2x speedup over the RISC-V, x86-Eigen and x86-GSL implementations respectively, as seen in Figure 8. After a matrix size of 256, all implementations suffer a substantial drop in throughput due to the problem size exceeding cache capacity. However, the LACore continues to outperform all other platforms due its LAMemUnits being able to efficiently stream sparse-matrices from memory.

C. Design Area Estimates

Dual Mode floating point circuits for Add, Subtract, Compare, Multiply and Divide are used in the LAExecUnit's datapath. A 110 nm dual-mode Adder, 90 nm dual-mode Multiplier, and 90 nm dual-mode 2-stage Divider are presented in [1][9][10]. The total area footprint of the LAExecUnit datapath at the 32 nm technology node is found by multiplying the instance count of the Adders, Multiplier and Dividers by the scaled-down area of each from the 110 and 90 nm nodes to the 32 nm nodes, and turns out to be 4.37 mm^2.

	(mm^2)		
RISC-V Scalar	0.46		
I$,D$,L2$	4.70	GPU Area (mm^2)	21.78
Scalar Total	**5.16**	**GPU/LACore Area**	**1.67**
LACore Datapath	4.37		
LACore Control	1.46	GPU Memories (kB)	746
LACache/Scratch	2.08	LACore Mem (kB)	464
LACore Total	**13.06**	**GPU/LACore Mem**	**1.6**
LACore/Scalar	**2.53**		

Table III: RISC-V Scalar CPU, LACore and 2 NVIDIA P100 SMs area breakdown and comparison.

The other components in the LAExecUnit include 48 64-bit multiplexers, 128 4-deep 64-bit FIFOs between the components, and 4 LAMemUnits, which will all be grouped into the control portion of the LACore, and a conservative estimate of 25% of the total area and power of the LACore will be allocated to this control portion. Area estimates for the caches and scratchpad in the LACore were found using the CACTI 5.3 Web Interface [19] for the 32 nm process. The configuration for the caches and scratchpad were the same as those shown in the gem5 configuration in Table II. The scalar CPU area is taken from the design in [11], a RISC-V processor with a vector-extension unit implemented on a 28 nm node. The total area of the Scalar CPU, not including the vector extension was 0.461mm^2.

The full LACore area estimation is given in Table III, with the LACore-specific hardware taking up 60.4% of the total area. The LACore uses 2.53x the area resources as a simple RISC-V CPU, including all the caches and scratchpad.

A comparison of the LACore's area to two NVIDIA P100 Streaming Multiprocessors (SMs) is drawn using the statistics in [14] for a SM. For the global resources, such as the GPU L2 cache, the value is divided by 28, which is half the number of SMs on the GPU. The comparison of the LACore with the two SMs in Table III shows that the scaled GPU uses 1.67x the area and 1.6x the memory components as the LACore.

V. Related Work

Traditional memory-memory linear algebra supercomputers such as the Texas ASC [20], the Cray-1 Supercomputer [17] and the CDC Star-100 [16] have similar features to the LACore but are large immobile machines, unlike the smaller footprinted LACore. This means the LACore is capable of accelerating a wide range of mobile or area-constrained linear algebra applications that are untouchable by traditional linear algebra supercomputers. The large, traditional supercomputers were sold in the units of tens, while the LACore system has the potential to be deployed at a much large scale at magnitudes less cost. Additionally, the CDC Star-100 suffered poor performance to do a scalar processing bottleneck, and also suffered from inability to quickly pipeline large vector operations, which the LACore does effectively with the scratchpad. Also, the Cray-1 architecture uses vector registers instead of decoupled FIFOs, like the LACore does.

There are major differences between the LACore design and the Hwacha vector coprocessor [12]. Like the LACore, the Hwacha architecture is a modern, lightweight variation of the archetypal vector processors. However, the LACore's vector-like processing capabilities are much more flexible, including the ability to operate on mixed precision datatypes at the same time, stream arbitrarily shaped vector and sparse-matrix data elements to and from the execution unit, and supports large-format vector reduction and multi-stream reduction. Furthermore, the Hwacha coprocessor contains its own scalar CPU and executes sub-programs separate from the controlling CPU, while the LACore is integrated directly into the scalar CPU both physically and in a unified instruction stream.

RSVP vector processor is also a decoupled data-streaming architecture with *Vector stream units* (VSUs) that perform similar functions as the LAMemUnits [4]. RSVP supports similar stride, skip, count vector descriptors. However, the RSVP does not support Large-Format vectors, heterogeneous datastreams, and uses a graph-processing datapath with support for limited-sized kernels, since it is designed for smaller multimedia applications.

The MOM ISA is an matrix-oriented ISA used in multimedia applications [5], which provides instructions for operating on matrices. This provides similar functionality to the multi-stream output mode of the LACore. The MOM ISA does not support Large-Format vectors, heterogeneous data and a decoupled datastream model like the LACore though.

VI. Conclusions and Future Work

In this paper, a novel supercomputing-like linear algebra accelerator for SoCs has been presented. Its unique architectural features such as the mixed-precision, configurable vector-reduction LAExecUnit and the highly-configurable Large-Format LAMemUnits were described, and the Instruction Set and LACoreAPI software framework were presented. The LACoreAPI is demonstrated to be a programmer-friendly framework that effectively utilizes the LACore's capabilities.

The LACore performance was evaluated against a RISC-V processor, an x86 processor and a scaled NVIDIA Fermi GPU in the HPCC benchmark suite, with the LACore outperforming all of these platforms in the STREAM, FFT, HPL, DGEMM and PTRANS benchmarks for a wide range of problem sizes. The LACore is demonstrated to be an optimal architecture for both high-and-low spatial-and-temporal locality applications through the HPCC results. Finally, the LACore's area footprint was shown to be less than a scaled NVIDIA GPU, while offering higher-performing capabilities for linear algebra applications.

Future Work: will focus on the scalability of the LACore to manycore designs, and compare performance with multi-core x86 and full NVIDIA GPU designs. Additionally, FPGA implementations of single-core and many-core designs will be used to evaluate performance, area and power of the LACore.

References

[1] A. Akkaş, "Dual-mode floating-point adder architectures," vol. 54, no. 12. Elsevier, 2008, pp. 1129–1142.

[2] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, and P. Marwedel, "Scratchpad memory: design alternative for cache on-chip memory in embedded systems," in *International Symposium on Hardware/software codesign*, 2002, pp. 73–78.

[3] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[4] S. Ciricescu, R. Essick, B. Lucas, P. May, K. Moat, J. Norris, M. Schuette, and A. Saidi, "The reconfigurable streaming vector processor (rsvptm)," in *IEEE/ACM International Symposium on Microarchitecture*, 2003, p. 141.

[5] J. Corbal, R. Espasa, and M. Valero, "Mom: a matrix simd instruction set architecture for multimedia applications," in *ACM/IEEE conference on Supercomputing*, 1999, p. 15.

[6] M. Frigo and S. G. Johnson, "Fftw: An adaptive software architecture for the fft," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 3, 1998, pp. 1381–1384.

[7] B. Gough, *GNU scientific library reference manual*. Network Theory Ltd., 2009.

[8] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," 2010.

[9] M. K. Jaiswal and H. K.-H. So, "Dual-mode double precision/two-parallel single precision floating point multiplier architecture," in *IFIP/IEEE International Conference on Very Large Scale Integration*, 2015, pp. 213–218.

[10] ——, "Area-efficient architecture for dual-mode double precision floating point division," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 2, pp. 386–398, 2017.

[11] B. Keller, M. Cochet, B. Zimmer, Y. Lee, M. Blagojevic, J. Kwak, A. Puggelli, S. Bailey, P.-F. Chiu, P. Dabbelt *et al.*, "Sub-microsecond adaptive voltage scaling in a 28nm fd-soi processor soc," in *European Solid-State Circuits Conference*, 2016, pp. 269–272.

[12] Y. Lee, A. Ou, C. Schmidt, S. Karandikar, H. Mao, and K. Asanovic, "The hwacha microarchitecture manual, version 3.8." *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-263*, 2015.

[13] P. Luszczek, J. J. Dongarra, D. Koester, R. Rabenseifner, B. Lucas, J. Kepner, J. McCalpin, D. Bailey, and D. Takahashi, "Introduction to the hpc challenge benchmark suite," *Lawrence Berkeley National Laboratory*, 2005.

[14] NVIDIA. (2016) Nvidia tesla p100.

[15] J. Power, J. Hestness, M. S. Orr, M. D. Hill, and D. A. Wood, "gem5-gpu: A heterogeneous cpu-gpu simulator," *IEEE Computer Architecture Letters*, vol. 14, no. 1, pp. 34–36, 2015.

[16] C. J. Purcell, "The control data star-100: performance measurements," in *Proceedings of the May 6-10, 1974, national computer conference and exposition*. ACM, 1974, pp. 385–387.

[17] R. M. Russell, "The cray-1 computer system," *Communications of the ACM*, vol. 21, no. 1, pp. 63–72, Jan. 1978.

[18] J. E. Smith, "Decoupled access/execute computer architectures," in *ACM SIGARCH Computer Architecture News*, vol. 10, no. 3, 1982, pp. 112–119.

[19] D. Tarjan, S. Thoziyoor, and N. P. Jouppi, "Cacti 4.0," Technical Report HPL-2006-86, HP Laboratories Palo Alto, Tech. Rep., 2006.

[20] W. Watson, "The ti asc: a highly modular and flexible super computer architecture," in *Proceedings of the December 5-7, 1972, fall joint computer conference, part I*. ACM, 1972, pp. 221–228.