Initial Architecture Document

Team Number: 11

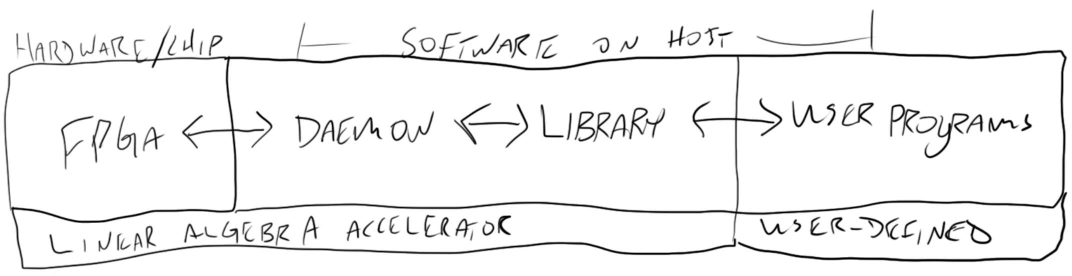
Team Members: Alexander Archer, Aditi Darade, Daniel Ginsberg, Jarrod Grothusen, Andrew Macgillivray

Project Name: Linear Algebra Accelerator

Project Synopsis: FPGA-based accelerator-card design for linear algebra operations using RISC-V with the RVV extension, packaged with a daemon-style access controller/scheduler for the host system.

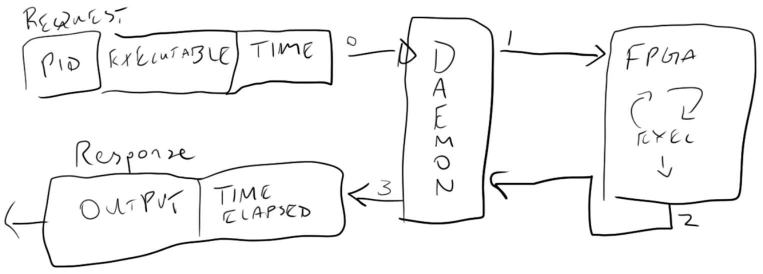
Architecture:

Our project consists of four components: the RISC-V based accelerator card, implemented on an FPGA; a "Daemon" that acts as an access controller by taking requests into a queue and running them on the accelerator; a library of helper functions that user-defined programs can include and use to send requests to the daemon and receive output; and a number of user-defined programs examples, used as a test suite. Moving through these components in reverse helps to explain the purpose of our project. The diagram below shows the general outline of our project, and the separation between what we call the "Linear Algebra Accelerator" (the FPGA, daemon, and library) versus the user-defined programs that will request to use the Linear Algebra Accelerator. Note that "Linear Algebra Accelerator" may be abbreviated as "LA2".



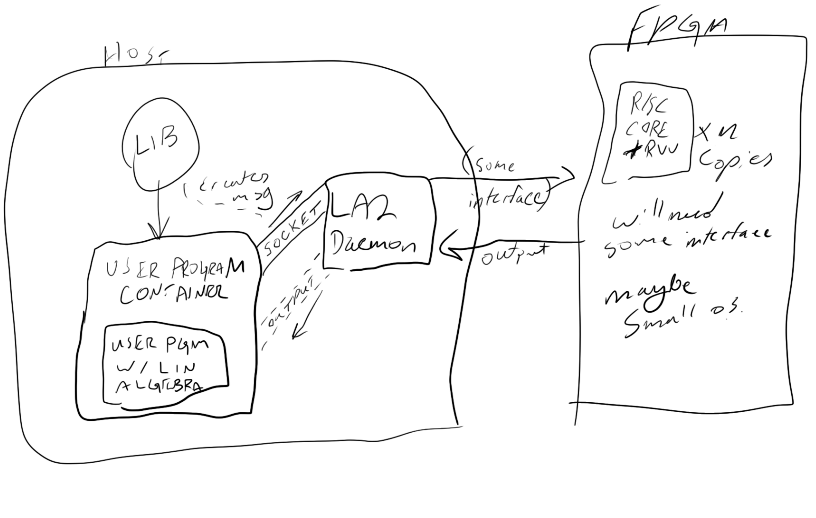
We begin with user-defined programs, which are any programs written by a (hypothetical) third party. This splits further into two sets of user programs: 1) those that use linear algebra operations and are compiled to run on our RISC-V architecture, and 2) any 3rd party programs that send requests to run a RISC-V compatible program on the accelerator. The finer details of both are going to depend on the format that we require programs to be in when sent, which will vary based on our final FPGA implementation. In general, a container program (type 2) will include the Linear Algebra Accelerator library, which it will use to send information about a type-1 program to the Daemon. The Type 2 program will then wait for its request to be handled, and collect the output of the type-1 program when it has been calculated on the FPGA and sent back via the daemon.

Next, we have the user library. We plan for the user library to contain a small number of functions (perhaps even a single function) that, when called, generate an appropriately formatted message and send it to the Daemon process through a specific socket or queue. The daemon will ingest the message, add it to the queue, execute it on the accelerator in as soon as possible, and return the results through an output queue or a socket with a predetermined naming convention that the user program can anticipate at runtime. Once the output is returned by the function call, the data is ready to use. The library may be expanded to support more than just requests (by adding debugging functions, requests for FPGA information, etc).



The Daemon layer, accessed using the functions in the user library, is somewhat complex. It keeps a virtual representation of the accelerator card and its status (idle, in use, etc), to help schedule requested jobs, and tracks all of the information necessary to start a job and return its results to the process that sent the request. Like a standard daemon, this will be a compiled program that is called every time the host system starts up, and runs as long as the FPGA is connected. If we are able, we may also allow it to handle different types of requests, such as requests for information about the FPGA (e.g. maximum ram available to a program, number of cores, etc).

Finally, the FPGA layer is where the accelerator will be implemented. We will use VHDL to describe each component of our core, then lay it together so that it supports basic RISC-V instructions as well as those from the RVV ISA extension. The FPGA will be programmed to have as many of these cores as we can possibly fit (limited by the resources of the FPGA). Then, we will have to figure out the best way to load programs into memory for execution - we would prefer to find a method for bare-metal execution, but may resort to using a small OS or similar solution tool.



The resulting architecture will look something like the diagram to the right. “Lib” is a function library and/or single header that includes a file that users can access and use to interact with the Daemon. The functions in lib abstract away the complexity of formatting messages and sending/receiving through the Daemon's sockets. Below the "Lib", the diagram shows two user programs: a container program - the one that uses "Lib" to talk to the daemon - as well as the program compiled for our RISC-V architecture that includes the linear algebra operations to be executed on the accelerator. The container program uses Lib to forward the details of the linear-algebra program to the Daemon. The daemon ingests the message and, when ready, forwards the program to the FPGA, where it executes. The output is returned from the FPGA to the Daemon, which uses some tracking object or struct to identify the request to which the data belongs to and calculate the time elapsed from the request was first received until the output was returned to the user program. Logs of the requests and output may also be stored for future analysis.

Once all of the parts are completed, we plan to collect timing information so that we can compare our accelerator card to execution on a typical consumer-grade or workstation CPU. Using this timing information, we will look for opportunities to improve our chip design and software architecture to maximize performance. The results of that analysis and any optimizations will be recorded within the project's documentation. Ideally, we will be able to show that the chip provides a noticeable improvement in the execution of linear-algebra related programs compared to standard hardware. We may also rank our project against existing products such as the SiFive x280, if performance metrics for those products are readily available. While we wouldn't expect to outperform a polished, market-ready product, it would provide interesting insight into the quality of our work and ways that our project could be improved in the future. The project will also be published on GitHub under an open-source license, to help others learn the basics of FPGA implementations and allow any of our group members to extend the project in the future if we so desire.