Group 11

Aditi Darade, Alexander Archer, Daniel Ginsberg, Jarrod Grothusen, Andrew Macgillivray

**Initial Requirements Stack:**

3) Brainstorm ideas for an initial set of requirements:

In order to create a practical test for the feasibility of the proposed accelerator card, our team will emulate the described card on an FPGA -- behaving as a chip with 4 RISC-V cores with the RVV or LACore extension. Assuming that the card would be best suited for large compute-clusters involved in AI applications, we will provide software on the host machine to control the usage of the card. Specifically, we will provide a daemon that accepts requests in a specific JSON format over an always-open socket, and sends output data back to the request originator. The complete project should encompass a suite of test programs, the daemon, and an emulated chip. It should be running on a Linux-based server.

For sprint one, we will focus on the software side, defining a test program that uses linear algebra in a manner similar to common AI cases. We will compile that program and collect timing data from it when it is executed normally on the CPU, for future comparison against the FPGA implementation. Then, knowing the needs of that program, we will define an appropriate message format for the daemon, as well as a standalone function to interpret those messages (that will later be used within the daemon). On the hardware side, we will also search for an appropriate hardware device (researching the capabilities of various FPGAs as well as the resource requirements for emulating RISC cores) and determine the tools (IDEs and otherwise) that will need to be used.

After sprint 1, we alter between focusing on the development of the daemon and the FPGA implementation, ensuring that the FPGA implementation is complete enough to use for testing the daemon. If necessary, the daemon may temporarily call regular programs and execute them on the CPU if the FPGA implementation is not yet complete. We will then use the test programs we wrote to send a request to the daemon, which will process the request by running the program on the accelerator and returning output through a socket. Timing data will be collected and compared to the times collected for CPU-only execution, and possibilities for optimization will be explored.

4) Prioritize your Requirements Stack from most important to least important:

Sprint 1:

1. Find an appropriate FPGA device
2. Determine and acquire FPGA programming toolset (IDEs)
3. Write a linear algebra test program
4. Compile normally and collect CPU timings for future comparison vs card-accelerated
5. Determine message format to a daemon for running program on accelerator
6. Implement and test request handler for Daemon

Other:

1. Acquire and configure an appropriate server, with the FPGA installed
2. Program the FPGA to emulate RISC-V cores
3. Add RVV or LACore ISA extension for each core
4. Memory area on FPGA for input
5. Memory area on FPGA for output
6. Determine how to communicate with FPGA at runtime (openCL?)
7. Program the daemon to actually call the FPGA and execute user programs on the emulated RISC cores
8. Check FPGA implementation for correctness (compare linear-algebra program outputs to same program compiled and run on normal CPU)
9. Program daemon so that it tracks the users that originate each request
10. Return output of linear-algebra programs to users (from the daemon) using a socket
11. Provide C++ header file that simplifies calling / receiving actions to/from the daemon in user programs (function library)
12. Keep scoreboard of core activity within the daemon so it knows when a core is ready for a new program
13. Determine a prioritization scheme for daemon to use to change order of request handling, if not FIFO
14. Create Test Suite
15. Start the daemon process on system startup
16. Implement error handling and inform users of errors with their programs that interrupt execution on the chip
17. Find methods to improve chip performance
18. Use Gem5 to model the card and corroborate FPGA timings

5) Estimate the number of Story Points for each requirement in the Requirements Stack:

|  |  |
| --- | --- |
| **Requirement Stack Stories** | **Story Points** |
| Write a linear algebra test program | 3 |
| Compile normally and collect CPU timings for future comparison vs card-accelerated | 2 |
| Determine message format to a daemon for running program on accelerator | 5 |
| Implement and test message handler for Daemon | 5 |
| Find an appropriate FPGA device | 13 |
| Determine FPGA programming toolset | 3 |
| Expand to other tasks if time is available | 21 |

6) Requirements Stack spreadsheet:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Requirement ID** | **Description of Requirement** | **Story Points** | **Priority** | **Sprint No.** |
| 1 | Write a linear algebra test program | 3 | 3 |  |
| 2 | Compile normally and collect CPU timings for future comparison vs card-accelerated | 2 | 4 |  |
| 3 | Determine message format to a daemon for running program on accelerator | 5 | 5 |  |
| 4 | Implement and test message handler for Daemon | 5 | 6 |  |
| 5 | Find an appropriate FPGA device | 13 | 1 |  |
| 6 | Determine FPGA programming toolset | 3 | 2 |  |
| 7 | Acquire and configure an appropriate server, with the FPGA installed | 5 | 7 |  |
| 8 | Program the FPGA to emulate RISC-V cores | 13 | 8 |  |
| 9 | Add RVV or LACore ISA extension for each core | 13 | 9 |  |
| 10 | FPGA input mem | 3 | 10 |  |
| 11 | FPGA output mem | 3 | 11 |  |
| 12 | FPGA communication (OpenCL) | 8 | 12 |  |
| 13 | Program the daemon to actually call the FPGA and execute user programs on the emulated RISC cores | 8 | 13 |  |
| 14 | Check FPGA implementation for correctness (compare linear-algebra program outputs to same program compiled and run on normal CPU) | 3 | 14 |  |
| 15 | Program daemon so that it tracks the users that originate each request | 5 | 15 |  |
| 16 | Return output of linear-algebra programs to users (from the daemon) using a socket | 5 | 16 |  |
| 17 | Provide C++ header file that simplifies calling / receiving actions to/from the daemon in user programs (function library) | 5 | 17 |  |
| 18 | Keep scoreboard of core activity within the daemon so it knows when a core is ready for a new program | 8 | 18 |  |
| 19 | Determine a prioritization scheme for daemon to use to change order of request handling, if not FIFO | 5 | 19 |  |
| 20 | Create Test Suite | 8 | 20 |  |
| 21 | Start the daemon process on system startup | 3 | 21 |  |
| 22 | Implement error handling and inform users of errors with their programs that interrupt execution on the chip | 8 | 22 |  |
| 23 | Find methods to improve chip performance | 13 | 23 |  |
| 24 | Use Gem5 to model the card and corroborate FPGA timings | 13 | 24 |  |