Matrix Operation Instructions in RISC-V Processor

Project specification

Callum Stew

Department of Computer Science

University of Warwick

1 Introduction

Many matrix operations involve computing many simpler calculations on the components of the matrix. These calculations are often independent and can be performed in parallel. A processor would compute these sequentially but a specific hardware implementation would be able to compute them in parallel. Adding hardware-based matrix operations to a processor instruction set would allow programs to reduce the number of cycles needed to get the result and so reduce the time taken. As the average clock speed of CPUs has seen very little increase since the early 2000s moving more complicated operations to hardware allows for run time to decrease without decreasing cycle time.

This project aims to develop hardware that can perform matrix operations and integrate that into the instructions of a RISK-V processor to improve the execution time of matrix-heavy programs.

2 Background

2.1 FPGAs

As this project will use a hardware implementation, a platform is needed that can allow fast development and iterations on hardware designs. An FPGA (Field Programmable Gate Array) is a semiconductor device based around a matrix of configurable logic blocks [6]. The user can define what logic function these blocks perform and how they are connected using HDL (Hardware Description Language) programming. Hardware can be designed in HDL and constructed on the FPGA without needing to manually build the circuit.

2.2 Matrix Operations

There are four basic operations with matrices, addition, scalar multiplication, transposition and matrix multiplication. There are multiple algorithms for matrix multiplication. The naive method takes a time of $\mathcal{O}(n^3)$. More efficient

algorithms have been found such as the Refined Laser Method with a time of $\mathcal{O}(n^{2.3728596})$ [1]. However, this is a far more complicated algorithm and would only offer better efficiency for large matrices.

In the naive method, each element of the result matrices of these functions can be independently calculated allowing for parallelism. To fully utilise this all elements of the input matrices must be accessible simultaneously which limits the scalability of the input matrix size the design can handle. Block matrix multiplication can be used to split large matrices into smaller blocks that can then be calculated. This allows the hardware to be used to compute larger matrices than it can handle as long as they have a size that is a power of two. Another advantage of the naive method is that a smaller matrix can be computed as a larger matrix by filling in the extra slots with zeros and then removing them from the result.

2.3 RISK-V, Rocket Chip and Chisel

For this project, a processor is needed to integrate the matrix accelerator with. RISC-V is an open standard instruction set architecture that can be implemented in a variety of devices such as high-performance boards by SiFive and smaller microcontrollers such as Espressif's ESP32-C3. This adoption means that more software is available for the RISC-V instruction such as Debian's risk64 port. To implement a RISC-V core on an FPGA the Rocket Chip Generator will be used. This is an open-source SoC generator that produces synthesizable RTL for implementation on an FPGA. It also allows for the integration of custom accelerators as instruction set extensions [2] which is needed for this project. The FPGA being used in this project should be able to run one rocket64b core.

Rocket Chip uses the Chisel HDL based on the Scala programming language. This allows for modern programming features to be used and then generate Verilog from this code. The Verilog code can then be synthesised for an FPGA.

3 Literature Review

The following is a comparison between two papers relevant to this project, Matrix Multiplication Accelerator [4] and Implementation of a RISC-V Processor with Hardware Accelerator [3].

	Matrix Multiplication Accelerator	Implementation of a RISC-V Processor with Hardware Accelerator
Development Board Used	Cyclone V - DE1-SoC with ARM Cortex-A9	ZedBoard with Artix-7 FPGA and ARM Cortex-A9 MPCore
Processor Used	ARM Cortex-A9 HPS	PULPino (RISC-V) implemented on the FPGA
Connection Used Between Accelerator and Processor	Parallel Port on AXI bus	APB bus accessible as virtual memory
Matrix Algorithm Used	Naive method with two approaches, register-based using parallelism and 'single-threaded' memory block approach with pipelining and high clock speeds	Naive matrix multiplication
Results	Higher runtime than on HPS but significantly more uniform distribution of results	Performance improvement by a factor of 4.5 with hardware using just 124 clock cycles compared to 603 in software

Implementation of a RISC-V Processor with Hardware Accelerator provided a more in-depth explanation of the project and more useful results. An instruction set extension implementation was not used due to the added complexity so a comparison with the results found in this paper may be useful. Matrix Multiplication Accelerator provided less valuable results as runtime is not as comparable when the hardware and software approaches are running a different clock speeds, 100MHz and 925MHz. It did provide an alternative design for the hardware implementation to improve scalability but block matrix multiplication can allow for the original design to computer larger matrices.

4 Objectives

4.1 Hardware-Based Matrix Operations

(Must) Design and implement matrix multiplication hardware that can calculate results more efficiently than software. It should work for matrices containing integers and should be able to scale to larger matrices.

(Could) Add addition and scalar multiplication functions to the design.

(Could) Add support for floating point values to the design.

4.2 Integration to RISC-V Core

(Must) Generate a RISC-V core and connect it to the matrix accelerator by adding new instructions to the cores instruction set.

(Must) Develop software to allow the use of the new instructions in programs.

4.3 Comparison of Hardware Solution Against Software

(Must) Develop a program that can benchmark the use of matrix operations through software and the hardware accelerator. Run these tests and compare.

(Could) Test and compare an implementation that uses software parallelism.

(Could) If other matrix functions are implemented compare these to software and see if these extra functions are worth implementing.

5 Methodology

To develop the matrix accelerator a plan-based methodology will be used. The structure of the accelerator will be designed before implementation and then test-driven development will be used when implementing it. It takes a long time to synthesise and generate a bitstream from HDL and can be very hard to debug hardware when it is running on an FPGA. To test more efficiently the HDL it is simulated using the tools built into Vivado for Verilog or with ChiselTest for Chisel code.

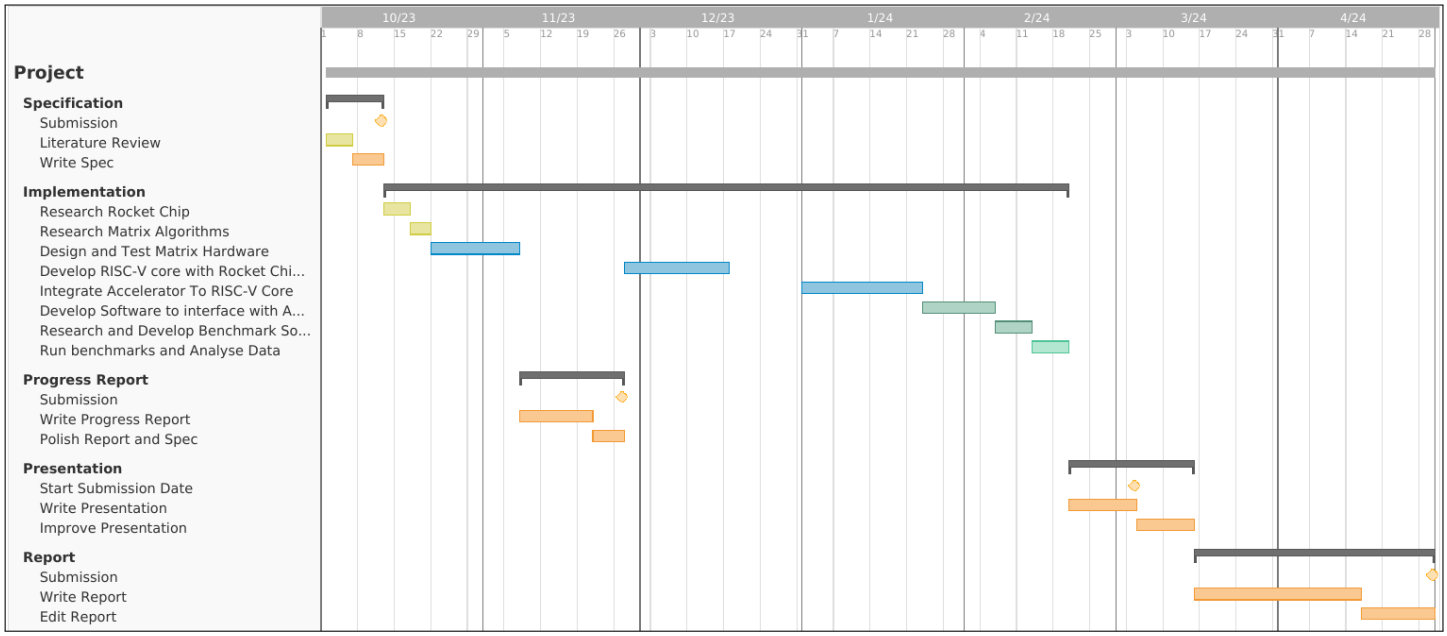
To test the performance of the accelerator a benchmark program will be needed. This should test a range of matrix sizes and if able shapes. The test should be run multiple times to find variance in results and should measure runtime and clock cycles.

6 Timeline

The planned working hours are set out in the table below giving 12 hours a week. This may change depending on what is currently being worked on.

	14:00	15:00	16:00	17:00	18:00
Monday					
Tuesday			X	X	X
Wednesday		X	X	X	
Thursday		X	X	X	
Friday			X	X	X

The Gantt chart below timetables the progress towards this project. This covers all objectives that must be completed and shows deadline submissions. Additional objectives such as adding extra functionality to the accelerator can be worked if the main objective is completed and there is still allocated time left.



7 Resources

This project will require hardware and software resources listed below as well as open-source tools such as Rocket Chip and Git.

1. **FPGA Development Board - Nexys A7-100T**
   This board will be used to synthesise the HDL generated by this project. It is on loan from the School of Engineering.

2. **Vivado Design Suit**
   Provides tools to synthesise and analyse HDL for FPGAs. Educational licence through the School of Engineering

3. **vivado-risk-v project** [5]
   An open-source project that generates RISC-V processors for FPGAs using Rocket Chip.

4. **Linux Workstation PC**
   A PC capable of running the Vivado Design Suit tools and the Ubunbtu

20.04 environment that is needed for the vivado-risk-v project. My PC fulfils these requirements.

8 Risks

Three main risks could harm this project. These are listed below along with mitigations and solutions.

1. **Damage to FPGA Development Board**
   This board is on loan from the School of Engineering so a replacement could be obtained. To reduce the chance of damage the board will only be transported when necessary and kept in its protective packaging when not in use.

2. **Damage to Workstation**
   Engineering computer labs could be used to run Vivado Design Suit and DCS labs can provide a Linux environment if needed.

3. **Data Loss**
   Git version control will be used to track changes and all data relevant to the project will be backed up to GitHub providing an easy way to recover data. A log will be kept of the process to generate data such as the use of vivado-risk-v so it can be repeated if necessary.

9 Legal, Social, Ethical and Professional Issues

This project should not present any social, ethical or professional issues and does not require working with people. There is no intent to profit from this project so no legal issues should arise.

References

[1] Alman, Josh & Williams, Virginia Vassilevska. A refined laser method and faster matrix multiplication, 2020. URL doi.org/10.48550/arXiv.2010.05846. arXiv:2010.05846 [cs.DS].

[2] Asanović, Krste & Avizienis, Rimas & Bachrach, Jonathan & Beamer, Scott & Biancolin, David & Celio, Christopher & Cook, Henry & Dabbelt, Daniel & Hauser, John & Izraelevitz, Adam & Karandikar, Sagar & Keller, Ben & Kim, Donggyu & Koenig, John & Lee, Yunsup & Love, Eric & Maas, Martin & Magyar, Albert & Mao, Howard & Moreto, Miquel & Ou, Albert & Patterson, David A. & Richards, Brian & Schmidt, Colin & Twigg, Stephen & Vo, Huy & Waterman, Andrew. The rocket chip generator. Technical Report UCB/EECS-2016-17, EECS Department, University of California, Berkeley, Apr 2016. URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html.

[3] Blomkvist, Ludvig & Oscarsson, Jonas Ibrahimoglu & Nilsson, Lucas & Stenseke, Adam & Wennerberg, Joakim. Implementation of a risc-v processor with hardware accelerator, 2019. URL https://odr.chalmers.se/server/api/core/bitstreams/8a3fa1e8-9dd1-4e4c-a598-6d4967d381bd/content. (Accessed October 2023).

[4] Dempsey, Brian & Patterson, Liam. Matrix multiplication accelerator, 2020. URL https://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/f2020/bjd86_lgp36/bjd86_lgp36/index.html. (Accessed October 2023).

[5] Tarassov, Eugene. vivado-risk-v. github.com/eugene-tarassov/vivado-risc-v.

[6] Xilinx, AMD. What is an fpga? URL xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html. (Accessed October 2023).