EE5332 Mid-Term Project Report

Saurav Sachin Kale, EE19B141

Group:
Saurav Sachin Kale (EE19B141)
Surya Prasad S (EE19B121)
Arun Krishna AMS (EE19B001)

What are we going to do?

Kalman filter implementation for realtime control applications

Defining the problem statement

What is a Kalman Filter?

It is an algorithm which approximates the state of the system (values of the state variables) given a stream of measurements of outputs.

What can we really accelerate in this computation and what are the merits of using an FPGA here?

Myself along with AMS worked on the analysis of the operations in this algorithm.
A summary of our analysis is presented below: (a more detailed analysis presented here)
Total FLOPs are in the ~4000 per iteration range for a 6 state variable Kalman Filter.
For realtime control, we need 1 iteration to get over in the range of ~1ms.
This means we need a system capable of computing 4000FLOPs / 1 ms = 4 MFLOPS.
The data rate of a GTX 1660 Ti graphics card is 288 GBps, and throughput is ~6 TFLOPS (on paper). Likely it is capable in this workflow. This is similar to what Abhiyaan uses currently.
However, there could be a possibility to greatly simplify the workflow and reduce costs and latency immensely by a custom lightweight FPGA implementation.
The Abhiyaan Bolt driverless vehicle currently routes the sensor data via sensor -> arduino -> raspberry pi -> pc and back. This can induce latency which can be improved by an FPGA. If we are able to pack decent functionality into a low number of FPGA resources, we could reduce costs immensely. Reducing the time taken by the Kalman filter generates valuable time for other systems, thus creating another possibility of reducing costs.
An iteration interval of <0.5us has been achieved on a very cheap FPGA for 4 state variable systems according to this paper. We could also use fixed-point to further simplify our calculations within engineering limits of typical applications. This paper has demonstrated that fixed point is able to get results close to the GPU in terms of accuracy.
We could show a Pareto optimality chart of all the implementations we make.

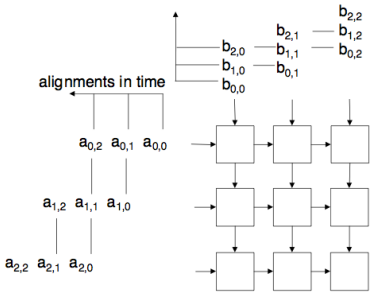
Work so far

What was the setup for the baseline?

I and AMS set up the test rig for baseline measurements. The Abhiyaan Bolt driverless vehicle currently routes the sensor data via sensor -> arduino -> raspberry pi -> pc and back. We tried to replicate this setup. The raspberry pi interfaces with the pc via SSH. We plan to make more accurate measurements with the baseline C implementation running on the PC soon. This got a bit delayed due to software and board issues.

What optimized architectures are we using to speedup the computations?

I explored in detail about various approaches to perform matrix multiplication and inversion. I wrote modules for both of these in Bluespec.



Systolic Arrays

Matrix multiplication was quite easy to implement using MAC PE units in a 2D gridlike fashion.

Gauss Jordan Inverse

I explored several algorithms and structures for Matrix inversion, like QR decomposition and LU decomposition which had several complex floating point operations like square root.

We planned on relying on the Bluespec floating point library, but the square root function is broken and produces wrong results. Therefore I chose to avoid this approach entirely and use a different algorithm. We might consider Shakti FBox if we get time later.

In the end, I went with Gauss Jordan elimination and row operations to find the inverse. Each set of row operations to convert that column to a pivot was parallelizable without any data dependency. Therefore this approach will take O(n) cycles for an n x n matrix.

This is not so bad, because the multiplication (even with systolic array approach) will also take O(n) clock cycles. So this doesn't become the bottleneck.

Future Plans

For the team as a whole, we plan to finish our self-assigned work. One thing we will do in high priority is to do better baseline measurements. The openmp and bluespec implementations are well underway. I plan to make the state machine which will do the Kalman iterations using the basic modules of mat-mul and mat-inv I made. We plan to compare the metrics between implementations towards the end.