Summer Research Reports

Vincent Zhao
vincentzhaorz@pku.edu.cn

August 3, 2015

Contents

`

1 Introduction

2 Reports Archive

2.1 Report of Week 3 (Aug. 3, 2015)

Basically I've finished running and testing the real hardware implementation of my previous SpMV design, and improved it by enabling large matrix calculation. Also, I've written a simple multi-pumped design by using MaxJ. This one has been checked about its accuracy and time frequency.

2.1.1 SpMV Implementation

CSRP format The format that I've devised and implemented could be called as CSRP (*CSR format with Padding zeros*). This format enables multiple rows of elements to be calculated at the same time, and when those simultaneous caculated rows don't have the same number of non-zeros, this format will automatically add padding zeros to ensure the result is correct.

The main advantage of this format is its simplicity in implementation, but it will also face these deflects:

1. *Redundant calculation*: For those cases when different rows have quite different number of non-zeros, the total number of calculations will depend on the maximum number of non-zeros among these rows.

2. *Resource duplication*: We need to replicate same kind of resources to enable parallelized calculation.

Implementation The implementation of CSRP format is quite simple. We set 2 input streams, one is for value and the other is for column index of each non-zero element. All these input streams are interleaved and zero-padded. Each of the calculation components has one a cyclic-loop which will sum the result for current row, and one ROM which stores the values of vector. We take Matrix Market Format of matrix as input, and generate interleaved data from the original data.

Optimization The detailed statistics will be updated by next week, but here I'll list several ideas of optimization:

1. *Reduce redundant calculation*: Currently the number of redundant calculation depends on the maximal number of non-zeros of all rows in the matrix. We could reorganize our output data and let number of padding zeros depend on the maximal number of non-zeros of all rows in each simultaneously calculated groups. After that, we could also reorder the matrix rows to get the best placement of rows into different groups, and get a globally optimizated result.

2. *Reduce resource usage*: We could do this by using dual-port of ROM, and if possible, using multi-pumping.

TODO

1. Finish the fully implemented baseline by Thursday. Fully implemented means, it could take MMF matrix data as input, and has reasonable space of memory.
2. Finish the first kind of optimization and the dual-port version by the end of this week.
3. If possible, try a multi-pumped version. (Only after the first 2 tasks finished)

2.1.2 Multi-Pumping

Simple implementation This simple implementation will do square on 2 input streams. Those input streams will be configured at 1x clock rate, and the inner multiplier will be configured at 2x clock rate. It could be implemented by using `ManagerClock` in MaxJ. The real hardware implementation has also been checked. The result is correct but the frequency is not accurate.

TODO

1. Finish checking this simple implementation. Must get a reasonable clock frequence.
2. Finish a Ruby implementation of multi-pumping by Thursday. Also, generate its MaxJ code.

2.1.3 Ruby Compiler

I haven't checked the compiler last week, but I'll start it as soon as possible. Basically, I should first use ruby to do some basic design, compile it into MaxJ and run it. After that, try to do some loop in Ruby. And test its MaxJ result (Which may not be correct).