Instruction Scheduling and Token Allocations for Modular Time-Multiplexing on CGRAs

Mitchell Fream(mfream@andrew.cmu.edu)
Xuesi Chen(xuesic@andrew.cmu.edu)

March 2023

1 URL

https://xuesichen.github.io/ under projects section

2 Project Description

The primary goal of project is to build an instruction scheduling technique leveraging the RipTide compiler to support time-multiplexing on CGRAs with the purpose of increasing hardware resource utilization.

In the current RipTide design, instructions and processing elements (PEs) have one-on-one mappings through out the entire program, which introduces inefficiencies when executing irregular workloads. For example, hardware resources executing outer loop instructions are stalled when waiting for inner loop instructions to finish, which can be repurposed to do something else. Thus, we introduce the idea of time-multiplexing instructions and aims to build a compiler implementation that statically schedule instructions to be executed on CGRAs. Because reconfiguration can happen in a single clock cycle, proper scheduling can extract a lot of parallelism from inner loops without large overhead.

Our preliminary study on nest loop workloads has shown that dividing the DFG representation of a workload into subgraphs based on loop boundary is a good starting point for figuring out the static instruction scheduling. We call those subgraphs dataflow blocks. The scope of the project focus on statically schedule instructions within a dataflow block so that it reaches a comparable execution time if we were to do dynamic execution within a dataflow block. We will evaluate the scheduling on dataflow-sim, which a simulator that models how dataflow architecture executes on CGRAs. With the time-multiplexing support already in place in the simulator. We can easily export and collect experimental results. We will investigate various possible constraints on instruction scheduling, such as limited total processing element count, limited communication channels between, or limited configuration options for each processing element.

We will evaluate these compiler changes on small nested loop kernels, such as matrix multiplication, for performance as well as on a set of other small programs to test correctness. If we exceed our goals, we will test on more complex code with dynamic loops and conditionals.

75 percent goal: Static scheduling close to the performance of dynamic scheduling within dataflow blocks, assuming latency of all executions to be 1 and number of hardware PEs used are passed in through command line. The scope of the workload is nested for loops.

100 percent goal: Based on the static scheduling results, We write programs to find the maximum number of tokens buffered between different dataflow blocks, and within dataflow blocks. This will shed light on architecture designs needed for running time-multiplexing.

125 percent goal: Expand the static scheduling algorithm to accommodate the non-nested loop workloads. We may also expand the algorithm for cases when we want to do spatial loop unrolling and multi-tenancy.

3 Plan of Attack and Schedule

Week 1: Literature review. Reviewing existing dataflow compilation flow to identify where to optimize scheduling.
Week 2: Create list scheduling equivalent for dataflow block scheduling. Test list scheduling on small programs for correctness.
Week 3: Create token counting analysis pass.
Week 4: Use token counting analysis pass to determine maximum buffer occupancy between dataflow blocks.
Week 5: Extend the scheduling algorithm to include loop unrolling and other non nested for loops related workloads within the constraint of buffer sizes between processing elements and blocks.
Week 6: Work on poster. Profile additional applications to determine total inter-dataflow block communication and maximum required buffer sizes.

4 Milestone

By the April 14th we hope to have a optimized static scheduling algorithm implemented that would give comparable performance in comparison to dynamic scheduling.

5 Literature Search

We have background materials on both scheduling for dataflow and on RipTide, the specific dataflow architecture we are investigating. There are well-studied modulo scheduling algorithms published regarding scheduling inner loop kernels on CGRA, for example: HyCube and MRRG.

6 Resources Needed

We will use our group's existing dataflow compilation infrastructure in LLVM as a starting point for our project. This is already available to us. We will also make use of our existing simulator to test the performance implications of our optimizations.

7 Getting Started

We have started our literature search. We also have a skeleton of the scheduler implemented in the compiler.

8 References

B. R. Rau, “Iterative modulo scheduling,” Proceedings of the 27th annual international symposium on Microarchitecture - MICRO 27, 1994. Bingfeng Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, “DRESC: A retargetable compiler for coarse-grained reconfigurable architectures,” 2002 IEEE International Conference on Field-Programmable Technology, 2002. (FPT). Proceedings., 2002.

Bingfeng Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, “Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling,” 2003 Design, Automation and Test in Europe Conference and Exhibition, 2003.

G. Gobieski, S. Ghosh, M. Heule, T. Mowry, T. Nowatzki, N. Beckmann, and B. Lucia, “Riptide: A programmable, energy-minimal dataflow compiler and architecture,” 2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO), 2022.

M. Karunaratne, A. K. Mohite, T. Mitra, and L.-S. Peh, “HyCUBE,” Proceedings of the 54th Annual Design Automation Conference 2017, 2017.