1 Overview

Proposal Title:	**Preprocessing for Modulo Scheduling within Open-Source ARM Cortex-A8 Compiler**
Principal Investigator (PI):	Gang-Ryung Uh, Associate Professor
Co-Principal Investigator (Co-PI):	David Whalley, Professor
Mailing Address of PI:	MEC 302-N, Computer Science Department, Boise State University 1910 University Drive, Boise, ID 83725, USA
Emails of PI and Co-PI:	uh@cs.boisestate.edu *and* whalley@cs.fsu.edu
Telephone Numbers of PI and Co-PI:	(208) 426-5691 *and* (850) 644-3506
Affiliations:	Boise State University *and* Florida State University
Google contact:	Dr. Brad Chen
Google sponsor:	Dr. Stephen Hines

2 Proposal

2.1 Abstract

As communication and media applications become more complex, recent mobile devices are being built with multi-issue processors to manage increasing data quantity and instruction flow. To automatically utilize the high instruction issue rate on these processors, commercial compilers are being extended to support variants of a *modulo scheduling* algorithm. However, we recognized that, for multi-issue mobile processors, these variants make little difference in finding a better *Instruction Level Parallelism* (ILP) between loop iterations. Instead, excessively long chains of data dependence cycles of instructions, which are frequently found in signal and media processing kernels, constrain scheduling freedom. These cycles often causes the compiler to produce less satisfactory code, which fails to exploit the full potential of the multi-issue *Program Dispatch Logic* (PDL). In this research, we propose to classify excessively long data dependence cycles of instructions in various loop kernels of DSPStone and MiBench benchmark suites for the dual-issue ARM Cortex-A8 processor. **First**, we will prepare an open-source compiler using LLVM and VPO for ARM *Instruction Set Architecture* (ISA). **Second**, we will investigate an effective preprocessing strategy for modulo scheduling within the compiler to split excessively long dependence chains into pieces. The preprocessing framework, which we plan to design and implement, may greatly help Google engineers develop a high performing Android platform that matches ARM Cortex-A9 performance, but with significantly reduced power consumption.

2.2 Problem Statement and Research Goals

Modulo scheduling is an aggressive loop transformation technique that exploits ILP in inner loops between loop iterations [1]. When this compiler technique is successfully applied to a loop kernel, the critical path of the loop is drastically compressed. As a result, performance can be greatly improved when the loop kernel forms an execution hot-spot. As multi-issue low-power processors become more popular, engineers are beginning to use variants of a modulo scheduling algorithm within compilers to optimize signal and media processing kernels for multi-issue PDLs. However, unlike a traditional multi-issue general purpose processor, it is notable that most operations in low-power processors typically require low latency. As a result, different slack scheduling strategies in these variants typically make no difference in finding a better ILP between loop iterations [2, 3]. Instead, for the multi-issue processors for mobile devices, we recognized that the limiting factor to the modulo scheduling is not the scheduling algorithm, but the excessively long data dependence cycles of instructions which are frequently found in signal and media processing loop kernels.

Problem Statement: *The problem we plan to study is the classification of excessively long data dependence cycles of instructions in loop kernels for the dual-issue ARM Cortex-A8 processor.* Once we correctly understand the nature of these data dependence cycles, we believe we might be able to split the chains into pieces.

Research Goals: *Our goal is to design and implement a practical preprocessing strategy that can enable modulo scheduling to better exploit ILP spanning more loop iterations by splitting long data dependence cycles of instructions.* In particular, by statically utilizing the dual-issue ARM cortex-A8 PDL via a compilation strategy that we plan to study, we may achieve the performance of ARM Cortex-A9 with much less power as it dynamically selects instructions to be issued for the PDL.

2.3 Proposed Research and Expected Outcome

How Modulo Scheduling is done in a Commercial Compiler for a Multi-Issue Low-Power Processor: Consider *freescale* SC140 processor since the alpha-2.6 SC140 commercial compiler was available to PIs. This processor supports a 6-issue PDL with *Variable Length Execution Set* (VLES) [4]. To statically utilize this multi-issue logic, the commercial compiler is comprised of three main stages – front-end, middle-end, and back-end. For a C source code, the front-end produces an intermediate code in the form of Abstract Syntax Tree (AST). The middle-end generates a sequential SC140

machine code by walking the ASTs multiple times. The back-end compacts a sequential code from the middle-end into VLESs for the 6-issue PDL. To maximize the use of VLES for an execution hot-spot, the back-end applies various transformations to a loop kernel with special emphasis on modulo scheduling.

Although the modulo scheduling in the back end is highly effective, excessively long chains of data dependence cycles of instructions, which are frequently observed in signal processing kernels, restrict scheduling freedom. As an

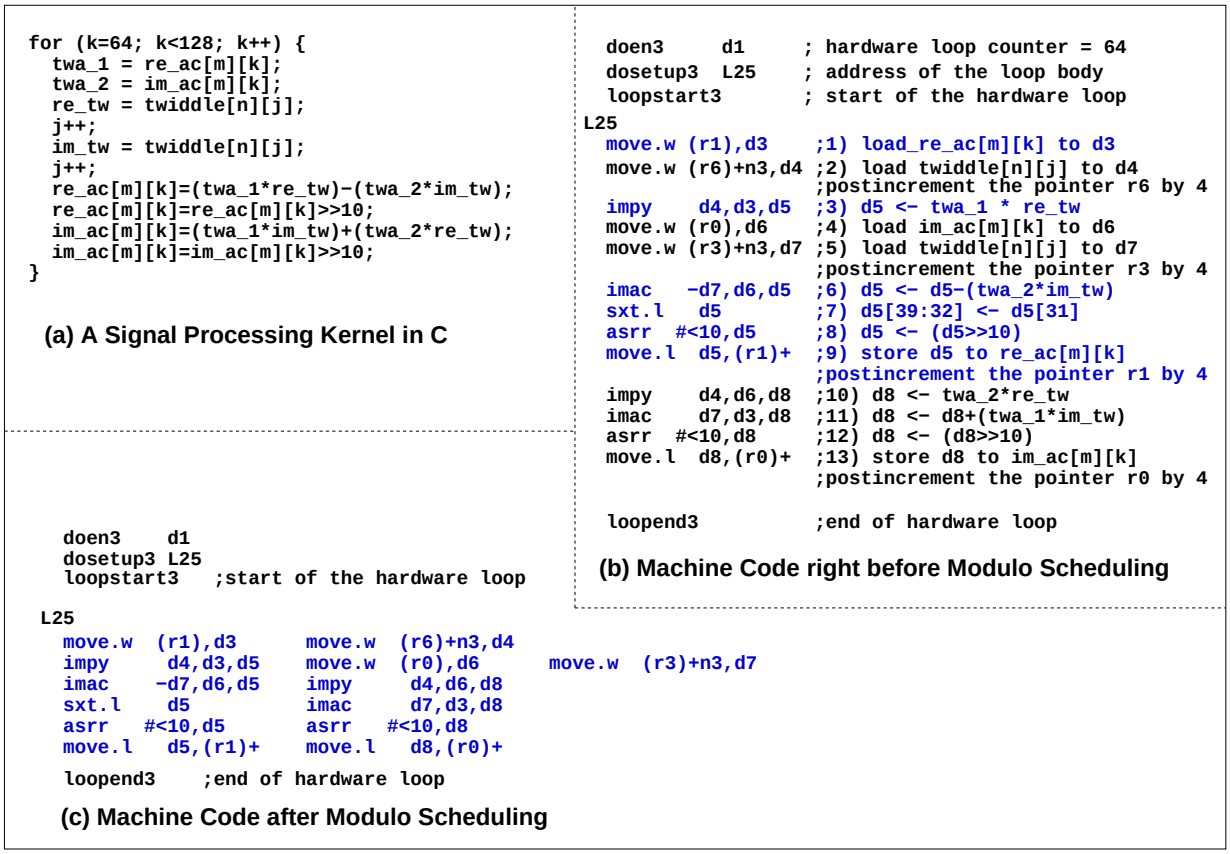

Figure 1: FFT Loop Kernel in C and SC140 Compiler Code Generation

illustration, for the *Fast Fourier Transform* (FFT) loop kernel shown in Figure 1(a), the alpha-2.6 SC140 DSP compiler generates the machine code shown in Figure 1(b) for modulo scheduling. To aggressively compress the loop height, modulo scheduling in the back end attempts to expose ILP between loop iterations. However, due to the long data dependence cycle of instructions formed by [1)] $\xrightarrow{1/0}$ [3)] $\xrightarrow{1/0}$ [6)] $\xrightarrow{1/0}$ [7)] $\xrightarrow{1/0}$ [8)] $\xrightarrow{1/0}$ [9)] $\xrightarrow{1/1}$ [1)] [1] in Figure 1(b), modulo scheduling fails to find ILP between loop iterations, and produces a less satisfactory schedule as shown in Figure 1(c). The resulting code requires 6 machine cycles per loop iteration and uses only 30% of the SC140 6-issue PDL.

Expected Benefits by Splitting the Data Dependence Cycles of Instructions: The length of this dependence cycle of instructions appears irreducible since the dependence from [9)] $\xrightarrow{1/1}$ [1)] is a loop-carried flow (true) dependence [2] [3]. However, there is no memory dependence from [9)] $\xrightarrow{1/1}$ [1)], and therefore, the loop-carried dependence with `r1` can be safely removed by creating an additional induction variable `r10` that replicates the behavior of `r1`. After repeating this preprocessing, this excessively long data dependence cycle of instructions is split into pieces. As a result, modulo scheduling in the back end is able to find ILP that spans three loop iterations, and achieves a better loop schedule requiring only four machine cycles per loop iteration, and using 54% of the 6-issue PDL. This is a significant performance improvement over the schedule shown in Figure 1(c).

Research Challenges and Plan: First, to mirror the performance improvement rate by splitting long data dependence cycles to the dual-issue ARM Cortex-A8 processor, we need an open-source compiler to design and implement various preprocessing strategies for modulo scheduling. However, there exists no such open-source compiler to enable our planned research. Second, to the best of our knowledge, no existing preprocessing strategy for dual-issue ARM Cortex-A8 has been

[1] [i)] $\xrightarrow{m/d}$ [j)] : m denotes delay cycles and d represents loop iteration distance from instructions i to j.

[2] `r1` in instruction [9)] is post-incremented and the updated value of `r1` is read by instruction [1)] in the next loop iteration.

[3] The ARM ISA also supports post/pre increment/decrement addressing modes, i.e., `STMDA` and `LDMDA`.

developed for modulo scheduling. Therefore, we need to explore practical preprocessing strategies, which can (1) spot excessively long data dependence cycles of instructions in polynomial time for analysis, and (2) split these dependence cycles into pieces by using functional resources which are otherwise left unused. Our plan is detailed in the following.

First, to properly execute the planned research for modulo scheduling, we will prepare an open-source compiler for ARM Cortex-A8 ISA using two open-source compilers, LLVM and VPO [5, 6]. LLVM compiler is library-based and therefore, it is highly reusable and extensible. The most important design aspect of LLVM is the Intermediate Representation (IR), which retains source level information that is vital to perform mid-level loop analysis and transformations. However, some LLVM IRs can be lowered to multiple machine effects during target code generation, which can potentially invalidate the effects from modulo scheduling. On the contrary, VPO is a machine level global optimizer, where each machine-dependent code improving transformation is performed in a machine-independent way. Therefore, VPO is an ideal framework to implement modulo scheduling and loop preprocessing to split data dependence cycles of instructions. However, VPO uses LCC as a front end, which has no mid-level code improving transformations [7]. By streamlining LLVM and VPO, the LLVM-VPO tool chain will be an ideal open-source compiler framework for the planned research, where LLVM will prepare highly optimized machine code for signal and media processing kernels, and VPO will perform modulo scheduling with various loop preprocessing strategies for the ARM Cortex-A8 processor.

Second, once the LLVM-VPO is prepared for the ARM Cortex-A8 processor, we will implement modulo scheduling and develop an algorithm to enumerate all excessively long data dependence cycles of instructions in various loop kernels from the DSPStone and MiBench benchmark suites [8, 9]. From this research activity, we will identify the nature of these data dependence cycles, and explore a practical preprocessing strategy to reduce the length of the dependence chains.

Relevance to Google: This new open-source LLVM-VPO compiler may enable Google engineers to develop a highly performing Android platform on low-power ARM Cortex-A8 that matches the performance of ARM Cortex-A9 while requiring significantly less power. We look forward to potentially collaborating with Google system tools engineers on this research.

3 Data Policy

We clearly understand the project may take more than one year. However, our intent is to build the LLVM-VPO compiler tool chain for ARM Cortex-A8 and implement modulo scheduling during the first year. We will release the compiler to the public to involve more researchers/developers with planned research beyond year one.

4 Budget

We are asking Google for $37,407 that covers (1) one year graduate research assistant support which includes salary, fringe, and tuition ($33,907), (2) student conference travel ($1,500), (3) ARM Cortex-A development boards ($1,000), and (4) ARM software tools ($1,000).

References

[1] B. Rau: Iterative modulo scheduling. In *HP Laboratories Technical Report, HPL94115*, Nov 1995.

[2] R. Huff: Lifetime-Sensitive Modulo Scheduling. In *Proceedings of the SIGPLAN'93 Conference on Programming Language Design and Implementation*, June, 1993.

[3] I. Finlayson, G. Uh, D. Whalley, and G. Tyson: An overview of Static Pipelining. In *IEEE Computer Architecture Letters*, 2011.

[4] Freescale semiconductor: Freescale SC140 DSP Core Reference Manual. Revision 4.1, Chapter 2, pages 32 – 100, September 2005

[5] C. Lattner and V. Adve: LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the 2004 International Symposium on Code Generation and Optimization*, March, 2004.

[6] M. Benitez and J. Davidson: A Portable Global Optimizer and Linker. In *Proceedings of the SIGPLAN'88 Conference on Programming Language Design and Implementation*, June, 1988.

[7] C. Fraser and W. Hanson and R. David: A Retargetable C Compiler: Design and Implementation. *ISBN 0-8053-1670-1*, Addison-Wesley, 1995.

[8] V. Zivojnovic, J. Velarde, C. Schager, and H. Meyr: DSPStone - A DSP oriented Benchmarking Methodology. In *Proceedings of International Conference on Signal Processing Applications and Technology*, 1994.

[9] C. Lee, M. Potkonjak, and W. Smith: MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems. In *Proceedings of the 30th Annual IEEE/ACM International Symposium on Microarchitecture*, Nov 1997.