ICIC Express Letters
Part B: Applications
Volume **7**, Number **2**, February **2016**

ICIC International ©2016 ISSN 2185-2766
pp. **363–368**

A PARALLEL LOOP SCHEDULING SCHEME ON FIELD PROGRAMMABLE GATE ARRAYS

Zhijian Lu

Postdoctoral Scientific Research Workstation
Shanghai Futures Exchange
No. 500, Pudian Road, Shanghai 200122, P. R. China
lu.zhijian@shfe.com.cn

Received August 2015; accepted November 2015

Abstract. *The designers in reconfigurable computing fields always require considerable knowledge in both software and hardware to build hybrid applications. The main challenges and barriers that hamper the wide adoption of reconfigurable computing systems are the lack of high-level design and development tools. This paper presents a compilation framework for IR2HDL mapping for Field Programmable Gate Arrays. The modulo scheduling schemes with a constant initiation interval in most compilers schedule the iterations and generate pipelines by using a pipeline division method. In order to reduce the pipeline depth and increase throughput, a parallel loop scheduling scheme and improved modulo scheduling method are presented. The test case on selected kernels shows the method improves the delay stage and the scheme shows significant performance improvement.*
Keywords: Reconfigurable compiling, Loop scheduling, IR2HDL

1. **Introduction.** Reconfigurable systems have shown the great importance in both computation-extensive and data-extensive applications, which combine the sequential and the spatial computing models. The applications should be partitioned to software and hardware parts. The software programs run in the general purpose processors such as CPUs. The hardware partitions are translated into hardware description language, such as Verilog and VHDL, and then synthesized to bit streams by vendor-specific tool chain, such as Quartus II by Altera. The translation process of high-level languages to HDLs is not trivial and extensive design knowledge of digital circuit and platform-specific hardware specifications are needed. Time is also an important factor. However, developers with software background are using high-level synthesis tools to generate HDL code [1-3]. For general purpose, automatic compilation is the key to the success of wide adoption of reconfigurable computing systems.

Handel-C [4] is a subset of C and a high-level programming language which targets reconfigurable hardware. It is mainly oriented towards the hardware developers and can be compiled to HDL before synthesizing to the corresponding hardware. All hardware details are exposed to developers by extending C syntax. SA-C [5] and Streams-C [6] are also subset of C programming language, which are designed to be directly and intuitively translated into hardware, including FPGAs. Hardware details are hidden for software designers by using C variations. This means the existing applications need to be rewritten. Many C-like languages have been proposed, such as SPARK [7], ROCCC [8,9], SPC [10], DEFACTO [11], Garp [12], Nimble [13], CHIMPS [14], which consider C subset as input languages and compiled C subset into HDL and will eventually be synthesized using commercial synthesis tools. Many high-level synthesis techniques have been proposed. However, none of them seems to have the potential to replace the hardware description

languages like Verilog and VHDL. These techniques focused on specific reconfigurable architectures emphasizing parallelizing schemes and memory optimizations. Loop pipeline can provide efficient schedules by overlapping the operations of different iterations sequentially, and thus can fit effectively on reconfigurable architectures. Modulo scheduling as a loop pipelining technique can generate operation arrangements in loops, and there are no data dependence violations or resource conflicts. Modulo scheduling methods [8,10,12,13] with constant initiation interval are used to generate loop pipelining architectures on reconfigurable hardware. However, these methods lead to more clock cycles and lower pipelining throughput. The ASAP scheduling and directed pipelining division method are used to insert nodes in the loop data dependence graph to these pipelines. Although the directed pipelining division method puts nodes with the same height in the loop data dependence graph into the same pipelining stage, more inter-stage registers and more resources are needed [10,12,13]. This paper presents a parallel loop scheduling scheme and improved modulo scheduling method. The test case on selected kernels shows the method improves the delay stage and the scheme shows significant performance improvement.

The rest of the paper is organized as follows. Section 2 presents an overview of the compilation framework. Section 3 shows the architecture of parallel loop pipeline. An improved pipelining division method and modulo scheduling with the variable initiation interval are also presented to improve the performance. Section 4 presents the result of test cases. Section 5 concludes the paper.

2. **Compilation Framework.** The framework uses LLVM front-end to convert C code to Intermediate Representations (IR) with standard data and control optimizations and could extract computation-extensive loops of C programs and compiles its subset into HDL running on a reconfigurable hardware. The selected kernel can be mapped by IR2HDL module, which can map the selected kernel to reconfigurable function units. The output files of the framework consist of C program, HDL and the interface files of software and hardware. The target architecture consists of a general-purpose processor and FPGA, which can raise the performance of the target applications. As shown in Figure 1, the process of each module in the framework is as follows.

Software/Hardware Partitioning uses the feedback profiling with computation and memory templates, which can be adapted to the applications. This module can select the kernels that make trade-offs between performance and resource allocations in IR for reconfigurable hardware. Preprocessing module transforms the selected kernels composed of a serial of basic blocks to function and is responsible for generating the software/hardware

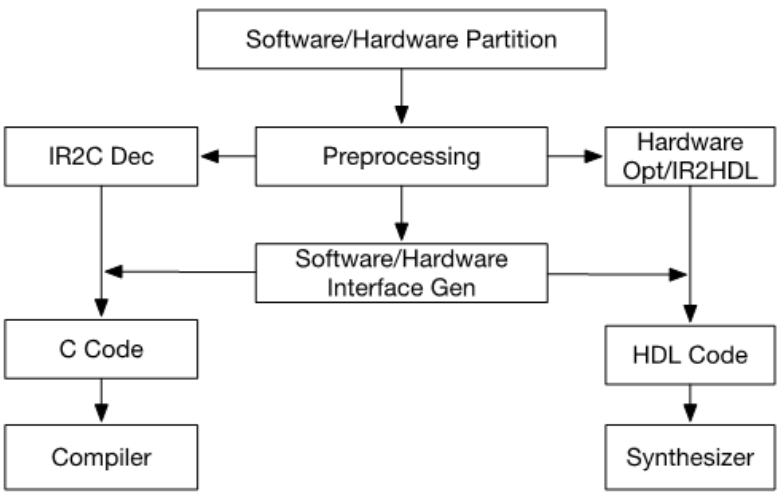

FIGURE 1. Compilation framework

interface file and the communication functions. IR2C Dec module transforms the modified IR to C program. LLVM IR is a static single assignment language. Its optimizations are for software and do not consider reconfigurable hardware features. It is necessary to apply dedicated optimizations to LLVM IR in order to fit in reconfigurable hardware. Operation parallelization, memory access optimization, basic block partitioning, bit-width analysis are the Hardware Opt module options. IR2HDL module can transform the IR functions into HDL. According to the target program characteristics, the IR2HDL module transforms the program as hardware partition into HDL with loop pipelining. Software/Hardware Interface Gen module generates the interface code in H/S interface information file.

3. **Parallel Pipelining Architecture.** Parallel pipelining architecture schedules instructions derived from different iterations of loop in parallel for hardware acceleration. Loop pipelining architecture can be suitable candidate for mapping on reconfigurable systems.

3.1. **Parallel pipelining framework.** Figure 2 shows the parallel pipelining framework generated for loop pipelining architectures.

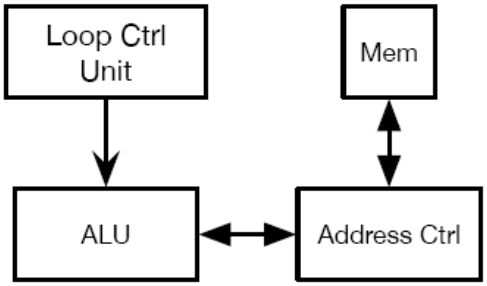

FIGURE 2. The framework of loop pipelining

The Loop Ctrl Unit controls the start, stepping, wait and termination of the loop pipeline. Because of the data dependence, a two-dimensional vector algorithm is used to deal with the iteration postponement. ALU is in charge of executing arithmetic operations under control of the Control Unit which is responsible for the pipeline schedule according to the Loop Data Dependence Graph (LDDG) after H/S partitioning. A process statement is used to describe the parallel behavior of ALU, and the communication between different processes. Each pipeline stage corresponds to a process of HDL design. Mem Interaction Unit is used to generate address and enable signals. By using the self-loop pipelining technique, a distributed control is used to generate indexes of arrays independently in the Address Ctrl module.

3.2. **The division of pipelining.** Pipelining division means adding registers to the proper place of the data path. Registers for output signals will be updated by the system after the completion of each stage operation. In order to generate loop pipelining architecture, division is the first step. Reconfigurable compilers use the ASAP pipelining schedule algorithm and directed division of pipelining method to put nodes into the same stage. However, it needs to insert more registers between stages and demands more resources. The time delay of each pipeline stage is different in the loop pipelining. The frequency of the design on FPGA depends on the circuit delay.

Figure 3 shows the results of pipeline stages using directed pipelining division method that puts nodes that have the same height in the LDDG into the same pipelining stage. The nodes represent operations and edges represent data dependence. Iteration of the loop body can be executed in pipeline. The logic delays of pipeline stages s1-s3 are shorter

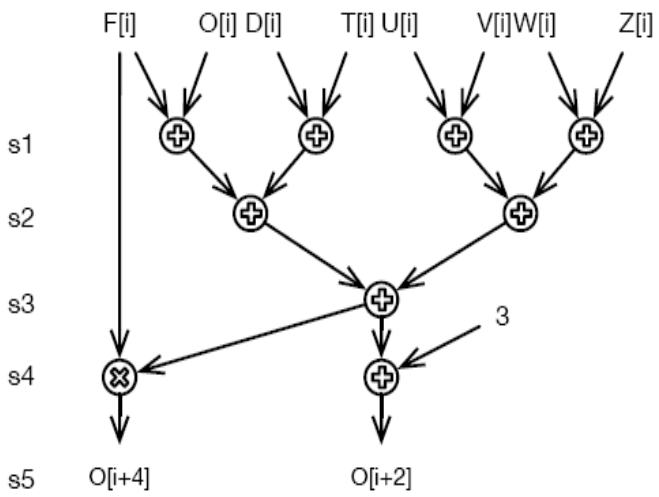

FIGURE 3. The loop LDDG and results

than stage s4. If s1-s3 stages are combined into one stage, the maximum frequency of hardware is the same, which can reduce the depth of the pipeline and the number of registers and increase the throughput of pipelining. Therefore, the improved method merges the shorter logic delay of pipeline stages without changing the maximum frequency. The combinational logic delay of the pipeline stages is estimated.

3.3. **Modulo scheduling.** Module scheduling methods are used by many reconfigurable compilers to generate loop pipelining architecture on FPGAs. Instructions from successive loop iterations are initiated at a constant interval, which defines the number of cycles between the beginning of two consecutive iterations. To determine the minimum Initiation Interval (II), the resource and data dependency constraints should be taken into account carefully. The resource constraints are determined by total resource requirements of the operations in the loop. Reconfigurable hardware like FPGAs has abundant resources in Configurable Logic Blocks, a large amount of LUT as on-chip memory, and other special-purpose resources, such as MAC. Data dependency constraint is also a critical factor in FPGAs. Minimum initiation interval preserves the data dependence and is the max number of cycles between the start time of the two successive iterations. The constant initiation interval is usually used in compilers when generating loop pipelining architecture.

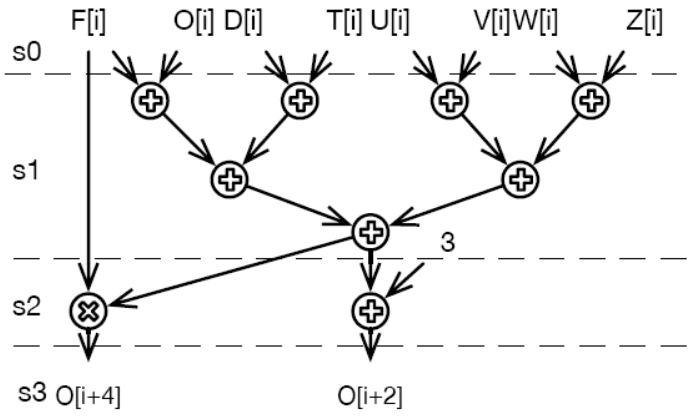

FIGURE 4. The combined pipeline stages

To characterize the dependences, an edge is presented with (ϕ, τ) pair. ϕ indicates the number of iterations that the dependence spans, and τ indicates the elapsed time between the time when the first and the second operation is issued. For cyclic dependence, a cycle λ contains a series of edges; we use ϕ_λ to present the sum of ϕ and τ_λ to present the sum of τ on λ. As shown in Figure 4, by the formula $II = \max(\forall\lambda)\lceil\tau_\lambda/\phi_\lambda\rceil$, $II = \max(\lceil 4/2\rceil, \lceil 1/2\rceil, \lceil 5/2\rceil, \lceil 5/4\rceil) = 3$. However, the loop iteration is also scheduled by the variable $II(1,4)$. In Figure 4, by the previous formula, $II = \max(\lceil 3/4\rceil, \lceil 3/2\rceil, \lceil 1/2\rceil) = 2$. However, this loop iteration is also scheduled by the variable $II(1,2)$. The modulo scheduling with constant II reduces the pipelining throughput.

4. **Test Case.** In this paper, test cases are used for experiment. The C program and software interface file are compiled to executable files. HDL and hardware interface files are synthesized into bit-stream file by ISE. The improved pipelining division method reduces the pipeline depth and the number of registers and does not reduce the maximum frequency.

In Table 1, DR means the directed pipelining division method and IM means the improved pipelining division method. There is not loop-carried dependency in test case 2. Because the other test cases have loop-carried dependency, they scheduled by the module scheduling with variable II are better than scheduled by the module scheduling with constant II. The module scheduling with variable II reduces the initiation interval. The number of loop iterations is 64 in the first three test cases. The loop execution cycles are shown by the scheme, which uses the improved pipelining division method and module scheduling with variable II to generate the loop pipelining architecture. The test case 4 is nested loop. The number of the inner loop iterations and outer loop iterations is 64.

Table 1. Different pipelining division II

Test case	Test 1		Test 2		Test 3		Test 4	
	DR	IM	DR	IM	DR	IM	DR	IM
CII	1	1	3	3	3	2	2	2
VII	1	1	3	(1, 4)	(1, 4)	(1, 2)	(1, 2)	(1, 2)

The loop pipelining has three components: a prolog, a steady state, and an epilog. In the steady state, a result is available every II cycle. The prolog and epilog are the instruction schedules that respectively set up and drain the execution of the loop kernel. Because both constant II and variable II are 1 in test case 1, the performance is improved only by the prolog component. Because the other cases have loop-carried, the performance is significantly improved.

5. **Conclusions.** In this paper, a parallel loop scheduling scheme for FPGA-based computing is presented, which applies compilation technology not only to accelerating software, but also to improving the hardware implementation. IR2HDL module is used to map selected kernels to the reconfigurable units, which can efficiently map SSA based IR to HDL. When generating loop pipelining architecture, previous compilation technologies use module scheduling with constant initiation interval and directed pipelining division method. In order to increase throughput, the improved pipelining division scheme and module scheduling with variable II are presented. In particular loop with loop-carried dependence, the scheme shows significant performance improvement. In the future, more high level languages will be added in the scheme such as Java.

REFERENCES

[1] A. Banaiyan, H. Esmaeilzadeh and S. Safari, Co-evolutionary scheduling and mapping for high-level synthesis, *Proc. of the IEEE International Conference on the Engineering of Intelligent Systems*, pp.1-5, 2006.

[2] A. Canis, J. Choi, M. Aldham et al., LegUp: An open-source high-level synthesis tool for FPGA-based processor/accelerator systems, *ACM Trans. Embedded Computing Systems*, pp.27-28, 2013.

[3] J. Curreri, G. Stitt and A. D. George, High-level synthesis of in-circuit assertions for verification, debugging, and timing analysis, *International Journal of Reconfigurable Computing*, 2011.

[4] L. Celoxica, *Handel-C Language Reference Manual for DK2.0*, Celoxica Ltd, pp.206-207, 2003.

[5] W. A. Najjar, W. Böhm, B. A. Draper et al., High-level language abstraction for reconfigurable computing, *Computer*, pp.63-69, 2003.

[6] J. Frigo, M. Gokhale and D. Lavenier, Evaluation of the streams-C C-to-FPGA compiler: An applications perspective, *Proc. of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp.134-140, 2001.

[7] S. Gupta, N. Dutt, R. Gupta et al., SPARK: A high-level synthesis framework for applying parallelizing compiler transformations, *Proc. of the International Conference on VLSI Design*, 2003.

[8] Z. Guo and W. Najjar, A compiler intermediate representation for reconfigurable fabrics, *Proc. of the International Conference on Field Programmable Logic and Applications*, 2006.

[9] B. Buyukkurt, J. Cortes, J. Villarreal et al., Impact of high-level transformations within the ROCCC framework, *ACM Trans. Architecture & Code Optimization*, pp.110-143, 2010.

[10] M. Weinhardt and W. Luk, Pipeline vectorization, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, pp.234-248, 2006.

[11] K. Bondalapati, P. C. Diniz, P. Duncan et al., DEFACTO: A design environment for adaptive computing technology, *Proc. of the 11th IPPS/SPDP'99 Workshops Held in Conjunction with the 13th International Parallel Processing Symposium and the 10th Symposium on Parallel and Distributed Processing*, 1999.

[12] T. J. Callahan, *Automatic Compilation of C for Hybrid Reconfigurable Architectures*, University of California Berkeley, 2002.

[13] Y. Li, T. Callahan, E. Darnell et al., Hardware-software co-design of embedded reconfigurable architectures, *Proc. of the 37th Annual Design Automation Conference*, 2000.

[14] A. R. Putnam, D. Bennett, E. Dellinger et al., CHiMPS: A high-level compilation flow for hybrid CPU-FPGA architectures, *FPGA*, 2008.