Path Selection Based Branching for Coarse Grained Reconfigurable Arrays

by

Shri Hari Rajendran Radhika

A Thesis Presented in Partial Fulfillment
of the Requirement for the Degree
Master of Science

Approved September 2014 by the
Graduate Supervisory Committee:

Aviral Shrivastava, Chair
Jennifer Blain Christen
Yu Cao

ARIZONA STATE UNIVERSITY

December 2014

ABSTRACT

Coarse Grain Reconfigurable Arrays (CGRAs) are promising accelerators capable of achieving high performance at low power consumption. While CGRAs can efficiently accelerate loop kernels, accelerating loops with control flow (loops with if-then-else structures) is quite challenging. Techniques that handle control flow execution in CGRAs generally use predication. Such techniques execute both branches of an if-then-else structure and select outcome of either branch to commit based on the result of the conditional. This results in poor utilization of CGRA′s computational resources. Dual-issue scheme which is the state of the art technique for control flow fetches instructions from both paths of the branch and selects one to execute at runtime based on the result of the conditional. This technique has an overhead in instruction fetch bandwidth. In this thesis, to improve performance of control flow execution in CGRAs, I propose a solution in which the result of the conditional expression that decides the branch outcome is communicated to the instruction fetch unit to selectively issue instructions from the path taken by the branch at run time. Experimental results show that my solution can achieve 34.6% better performance and 52.1% improvement in energy efficiency on an average compared to state of the art dual issue scheme without imposing any overhead in instruction fetch bandwidth.

DEDICATION

To my parents, for their unconditional love and support.

ACKNOWLEDGEMENTS

I am thankful to Prof. Aviral Shrivastava for giving me an opportunity to work on this thesis. I would like to express my sincere gratitude to him for his pin point criticisms and probing questions that enabled me to develop a solid understanding of the subject and instilled in me an attitude of not to settle for mediocrity. I am grateful for the valuable feedback, and the expertise provided by him.

I am thankful to Prof. Sarma Vrudhula for being friendly and approachable and letting me use his lab resources required for the experimental results of this thesis.

I am thankful to my colleagues Mahdi Hamzeh, Dipal Saluja, Reiley Jeyapaul and Bryce Holton for the technical discussions we had and for providing constructive criticism of my work in good faith since the initiation of this work. I would also like to thank my friends Sivaseetharaman and Sathish Kumar for their support for the completion of this thesis.

TABLE OF CONTENTS

Page

LIST OF FIGURES v

CHAPTER

1 INTRODUCTION 1

2 BACKGROUND AND RELATED WORK 5

2.1 Partial Predication 7

2.2 Full Predication 8

2.3 Dual Issue 9

3 LIMITATIONS OF EXISTING TECHNIQUES 12

4 PROPOSED APPROACH: PSB 14

4.1 What Must the Compiler Do? 16

4.2 Problem Formulation 19

4.3 My Heuristic 22

5 EXPERIMENTAL RESULTS 26

5.1 Experimental Setup 26

5.2 PSB Achieves Lower II Compared to Existing Techniques to Accelerate Control Flow 26

5.3 PSB Architecture has Comparable Area and Frequency with Existing Solutions 30

5.4 PSB Achieves Higher Energy Efficiency Compared to Existing Techniques 30

5.5 Instruction Memory Overhead in PSB is Tolerable 32

6 SUMMARY 34

REFERENCES 35

LIST OF FIGURES

Figure Page

1.1 A 4 × 4 CGRA. A PE Consists of an ALU and Two Register Files, Data Register File to Hold Data and a Predicate Register File Store Predicate Values (Result of Conditional Expressions) 1

2.1 (a) Shows a Simple Loop Body, (b) Shows a 2 × 2 CGRA with Torus Interconnection, (c) Shows the DFG for a Loop Body in (a), (d) A Valid Mapping for the DFG in (c) 5

2.2 (a) Shows a Loop Body with Control Flow, (b) Shows the Loop Body after SSA Transformation. 6

2.3 (a) Code Transformation for Partial Predication Scheme, (b) Shows Corresponding DFG for the Partial Predication Transformation, (c) Shows a Valid Mapping Obtained for the DFG on a 2 × 2 CGRA, the II Obtained is 3. 7

2.4 (a) Shows the Transformation for Full Predication Scheme, (b) Shows Corresponding DFG for the Full Predication Transformation, (c) Shows a Valid Mapping Obtained for the DFG on a 2×2 CGRA. The II Obtained is 5 8

2.5 PE Architectural Template for Full Predication Scheme 9

2.6 (a) Shows the DFG after Packed Node Formation (b) Shows a Valid Mapping Obtained for the DFG on a 2 × 2 CGRA. The II Obtained is 3. 9

2.7 PE Architectural Template for Dual Issue Scheme 10

4.1 A Valid Instruction Arrangement for PSB 15

Figure Page

4.2 Architectural Support for the Proposed Approach. The Branch Parameters and Outcome is Communicated to the Instruction Fetch Unit (IFU) to Issue Instructions only from the Path Taken at Run Time. 16

4.3 Selective Instruction Issuing Without Pairing of If-Path and Else-Path Operation a) Shows Instruction Arrangement b) Shows Mapping of the Kernel with Poor Resource Utilization of PEs 18

4.4 Arrangement of Instructions for the Loop Kernel after Modulo Scheduling . . . 19

4.5 (a) (b) (c) Shows a Valid Pairing of Operations from the If and Else-Path. (d) Shows an Invalid Pairing since such a Pairing Fails to Meet the Criteria for Validity and a Feasible Schedule for such a Pairing Does Not Exist 20

4.6 (a) (b) (c) Shows Elimination of Eligible PHI/Select Operation with Inputs from If-Path and Else-Path, (d) Shows an Example of a PHI that Cannot be Eliminated to Form a Fused Node since One of its Input Does not Belong to the Set of If or Else-Path Operations 21

4.7 Shows Construction of DFG with Fused Nodes from an Input DFG 24

5.1 Performance of Compiled Loops Using I) Partial Predication Malhke et al. [1992], ii) Full Predication Han et al. [2013] , iii) Dual-Issue Han et al. [2010], iv) PSB on a 4x4 CGRA 27

5.2 % of Conditional Instructions in a Loop Kernel Prior to DFG Formation. 28

5.3 % Reduction of Input DFG Size in Terms of Nodes and Edges for PSB Compared to other Approaches. On Average 40% or More Reduction in Edge Size

Figure Page

and 20% or More Reduction in Node Size Translates to Achieving Lower II Compared to Other Techniques. 29

5.4 Hardware Overhead of Supporting Existing Acceleration Techniques Used While Executing Loops with Control Flow on 4 × 4 CGRA with Torus Inter-Connection Network 30

5.5 Estimation of Relative Energy Efficiency Normalised with Respect to Full Predication Technique for Executing the Kernel of Each Benchmark 31

5.6 Required IMEM Size in Kb for the Benchmarks Used 32

Chapter 1

INTRODUCTION

Improving performance and energy efficiency simultaneously has been always the goal in micro-electronics industry. Quest for high performance and low power consumption has resulted in novel architectural solutions such as accelerators. Special purpose, custom hardware accelerators have been shown to achieve the highest performance with the least power consumption Chung et al. [2010]. However, they are not programmable and incur a high design cost. On the other hand Graphics Processing Units (GPUs), although programmable, they are limited to accelerating only *parallel loops* Betkaoui et al. [2010]. Field Programmable Gate Arrays (FPGAs) have some of the advantages of hardware accelerators and are also programmable Che et al. [2008]. However, their *fine-grain* reconfigurability incurs a very high cost in terms of power and energy efficiency Poon et al. [2005], Hartenstein [2001].

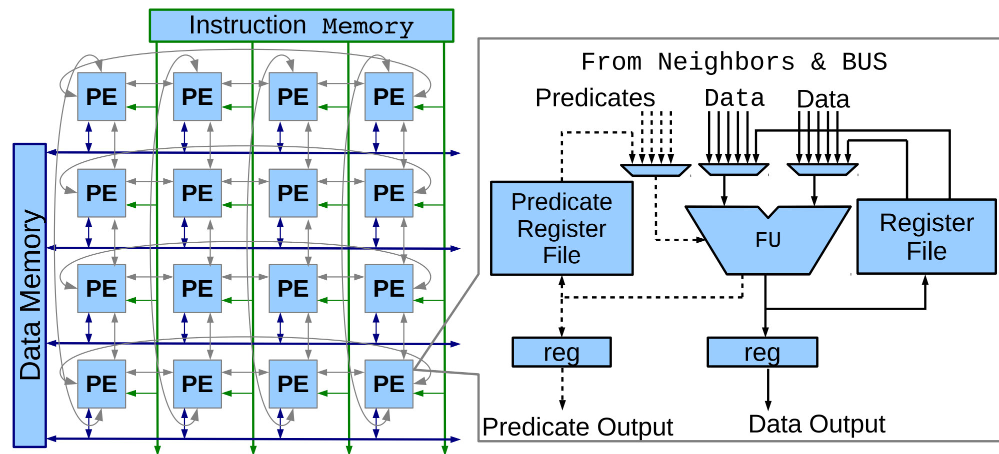


Figure 1.1: A 4×4 CGRA. A PE Consists of an ALU and Two Register Files, a Data Register File to Hold Data and a Predicate Register File Stores Predicate Values (Result of Conditional Expressions).

Coarse Grain Reconfigurable Arrays(CGRAs) are programmable accelerators that promise high performance at low power consumption Carroll et al. [2007]. For instance ADRES CGRA Bouwens et al. [2008] has been shown to achieve performance and power efficiency of upto 60 GOPS/W. CGRA is an array of processing elements(PE) which are connected with each other through an interconnection network as shown in Figure 1.1. Each PE consist of a functional unit, local register files and output register. The functional unit typically can perform arithmetic, logic, shift and comparison operations. The operands for each PE can be obtained from neighbouring PEs, it's output from previous cycle, data bus or the local register file. Every cycle, instructions are issued to all PEs specifying the operation type and position of input operands. CGRAs are programmable at a coarser granularity at a level of arithmetic operations in contrast to FPGAs which are programmable at bit level. Since CGRAs are capable of pipe-lining and executing iterations simultaneously they can accelerate both parallel and non-parallel loops De Sutter et al. [2013], Hartenstein [2001].

The majority of an application′s execution time is spent on loops Rau [1994]. Acceleration of such loops results in lower execution time and hence an improvement in performance. CGRAs have been used to accelerate such loops Galanis et al. [2005]. Software pipelining Allan et al. [1995] is a classic technique to accelerate loops and modulo scheduling Hatanaka and Bagherzadeh [2007] is a form of software pipelining that is widely used. Compiler solutions to accelerate both parallel and non-parallel loops on CGRAs are presented in Hamzeh et al. [2012], Park et al. [2008], Chen and Mitra [2012], Hamzeh et al. [2013]. The performance metric of these techniques is measured by Initiation Interval(II), which is the number of cycles after which the next iteration of the loop can be initiated. Lower II results in faster execution of the loop and hence better performance.

One of the major challenges associated with CGRA accelerators is accelerating loops with if-then-else structures. The need for supporting conditionals in loops is presented in Hamzeh et al. [2014]. Since the result of the conditional is known only at run time, existing solutions handle control in CGRAs by predication Mahlke et al. [1992],Mahlke et al. [1995],Han et al. [2013b],Chang and Choi [2008]. These techniques execute operations from both the paths of an if-then-else structure and chose the result based on the evaluation of the conditional. State of the art Dual-Issue scheme Han et al. [2010], Han et al. [2013a], Hamzeh et al. [2014] aims to improve performance by fetching 2 instructions per PE in the same cycle, one from the if-then path and the other from the else path, then by selectively executing one instruction, based on the result of the conditional. Predication based techniques result in poor utilization of CGRA′s computational resources (PEs), since operations from both the if-then path and else path are executed unconditionally. Moreover, there is mapping overhead to communicate the result of the conditional, to operations belonging to the if-then-else path. Dual issue scheme alleviates the problem of poor resource utilization by conditionally executing instructions in a PE. However, this comes at an overhead in instruction fetch bandwidth, since 2 instructions have to be fetched per PE. Moreover, there is hardware overhead additional circuitry required to select an instruction at each PE.

In this thesis, I aim to improve the state of the art to accelerate loop kernels with if-then-else structures in CGRAs by leveraging on the fact that only one of the paths of the conditional branch is required to execute at run time. My solution communicates the result of the conditional expression that decides the branch outcome, to the Instruction Fetch Unit (IFU) of the CGRA. The IFU issues only instructions that belong to the path taken by that branch at run time. Experimental results on accelerating loop kernels, with if-then-else structures from biobench Albayraktaroglu et al.

[2005] and SPEC Albayraktaroglu et al. [2005] benchmark by my solution demonstrates the following: a performance improvement (lower II) of 34.6% on an average and energy efficiency (CGRA power and power spent on instruction fetch operation) of 52.1% on an average compared to state of the art Dual issue technique Hamzeh et al. [2014], a performance improvement of 36% and improvement of 35.5% in energy efficiency compared to partial predication scheme presented in Mahlke et al. [1992] and 59.4% performance improvement and improvement of 53.9% in energy efficiency compared to full predication scheme presented in Han et al. [2013b], can be achieved.

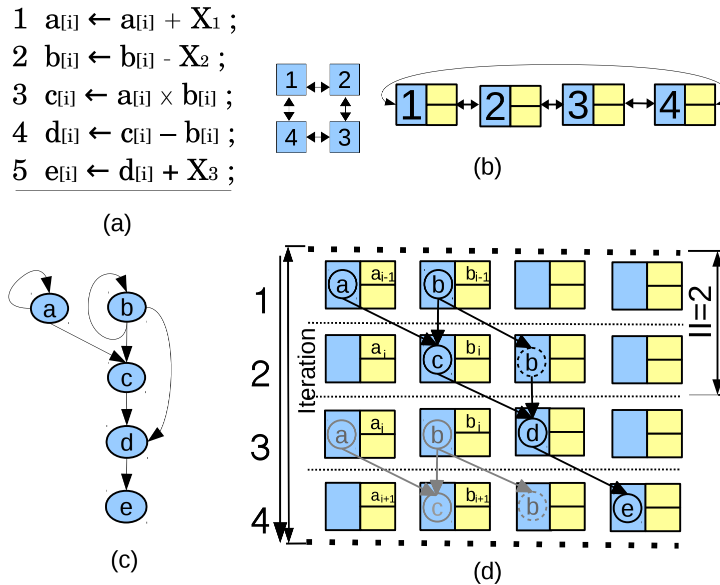


Figure 2.1: (a) Shows a Simple Loop Body, (b) Shows a 2×2 CGRA with Torus Interconnection, (c) Shows the DFG for a Loop Body in (a), (d) A Valid Mapping for the DFG in (c).

Chapter 2

BACKGROUND AND RELATED WORK

Loop kernels are the most desirable parts of the program to be accelerated in a CGRA Rau [1994]. Existing compiler techniques use modulo scheduling scheme to efficiently map the loop kernels on a CGRA Hamzeh et al. [2012], Park et al. [2008], Chen and Mitra [2012], Hamzeh et al. [2013]. Consider a simple loop kernel in figure 2.1(a) which has five instructions to update the variables a[i],b[i],c[i],d[i] and e[i]. X1, X2 and X3 in code represent constants obtained from the immediate field of the instruction to a PE. An attempt to map this kernel onto a 2x2 CGRA is shown in figure 2.1(d). As a first step a Data Flow Graph(DFG) of the loop kernel is created as shown in figure 2.1(c). Each node in the DFG represents an operation and the edges represent the data dependencies for those operations. A valid mapping

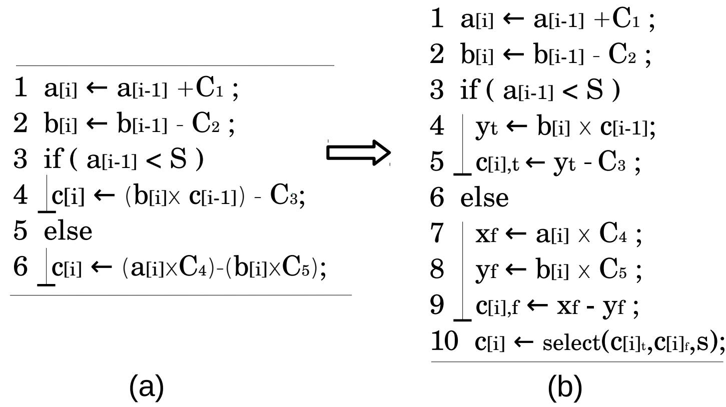


Figure 2.2: (a) Shows a Loop Body with Control Flow, (b) Shows the Loop Body After SSA Transformation.

for the DFG on a 2x2 CGRA is shown in figure 2.1(d). Nodes enclosed in solid lines represent computation and nodes enclosed dashed lines represent routing operation. The number of cycles required to execute an iteration of the kernel is schedule length, which is 4 for the assumed example. The performance metric, Initiation Interval (II) is 2, since the operations of the next iteration can be started 2 cycles after the start of the current iteration.

Consider a loop kernel with If-Then-Else as shown in figure 2.2(a). The resulting SSA transformation of the loop kernel is shown in figure 2.2(b). The kernel has 5 predicate based instructions, two in *if* block and three in the *else* block. The variable c[i] could be updated in both the blocks so it's updation must be conditional depending on the branch taken at run time. Variable yt is an intermediate variable used for the computation of the final value of ct in the *if* block. Variable xf, yf are intermediate values used for the computation of cf in the *else* block. C represents constant values that can be obtained from the immediate field of the instruction. There are three commonly used techniques to execute such kernels with if-else structures in CGRAs.

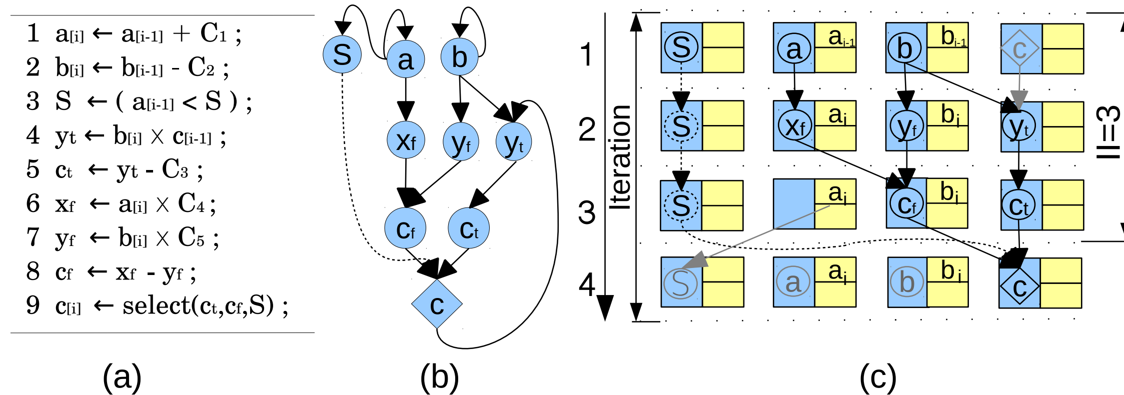


Figure 2.3: (a)Code Transformation for Partial Predication Scheme, (b) Shows Corresponding DFG for the Partial Predication Transformation, (c) Shows a Valid Mapping Obtained for the DFG on a 2×2 CGRA, II Obtained is 3.

2.1 Partial Predication

In partial predication method, the if-path instructions (true path) and the else-path instructions (false path) of a conditional branch are executed in parallel in different PE resources. When the result of computation becomes available at both paths, the final result of an output is selected between outputs of two paths based on conditional operations outcome (predicate value) as shown in figure 2.3(b). This is accomplished by a select instruction which acts like a hardware multiplexer. The diamond shaped nodes represent the select instruction for variable c[i]. If a variable is updated in only one path, a select instruction is still required to choose between old value and new value generated after executing conditional path. Details about architectural support for partial predication scheme is studied in Han et al. [2013a].

Architectural support for partial predication scheme is presented in figure 1.1. There is a predicate mux selecting a predicate value available from the neighbouring PEs or from the PEs register file or the predicate value generated by the PE in previous cycle. This predicate value is communicated to other PEs through a output predicate register and a predicate network.The overhead of this predication scheme

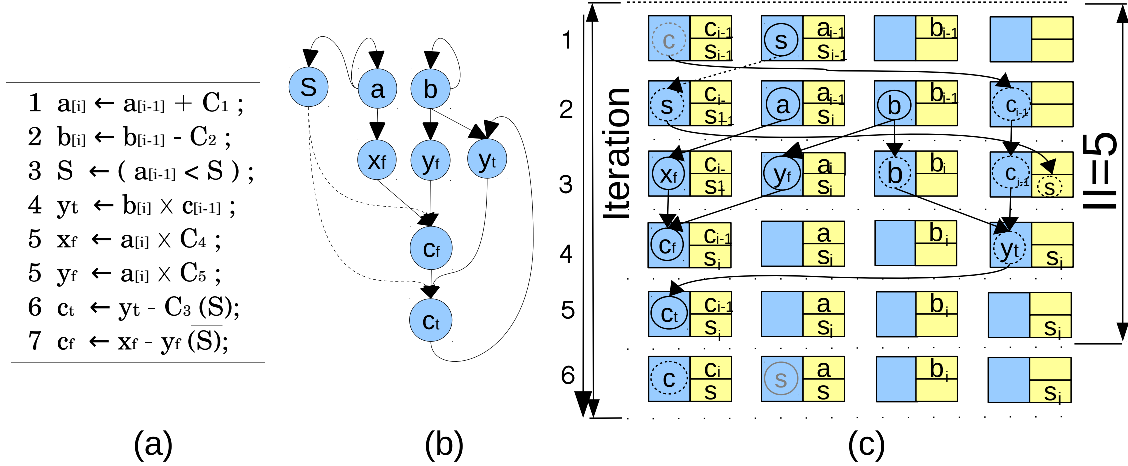


Figure 2.4: (a)Shows the Transformation for Full Predication Scheme, (b) Shows Corresponding DFG for the Full Predication Transformation, (c) Shows a Valid Mapping Obtained for the DFG on a 2×2 CGRA. The II Obtained is 5.

is that several select operations have to be introduced and instruction set has to be extended to support select operation.

2.2 Full Predication

In full predication scheme Han et al. [2013b], the operations that update the same variable in both the paths of an if-then-else structure has to be mapped onto the same PE albeit at different cycles as shown in figure 2.4(c) where operations ct and cf are mapped to the same PE (PE 1) at cycle 4 and 5. The right value is available in the register of the PE (PE 1) after instructions from both paths have been executed at cycle 6. At run time, false path instruction's result is suppressed, meaning the false path instruction does not update the register value.

The architectural support for full predication shown in figure 2.5 is very similar to the architecture for a partial predication technique. However, since the false path operations have to suppressed at run time, there is an additional hardware to enable this feature. The write enable signal to the register file is disabled when executing the instruction of the branch not taken at run time. This way only the instruction of

the branch path taken stores its result in the data register file.

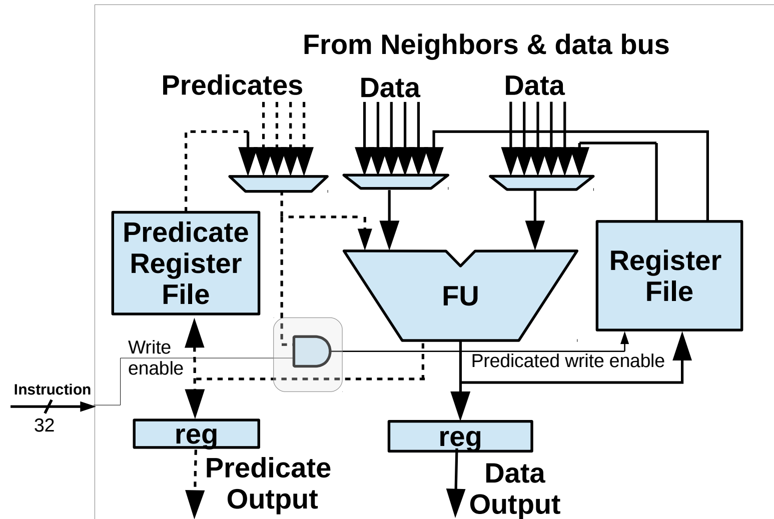


Figure 2.5: PE Architectural Template for Full Predication Scheme.

Full predication eliminates the need for select instructions, but there is an overhead because of the tight constraints on where instruction updating the same variable can be mapped. This leads to poor resource utilization.

2.3 Dual Issue

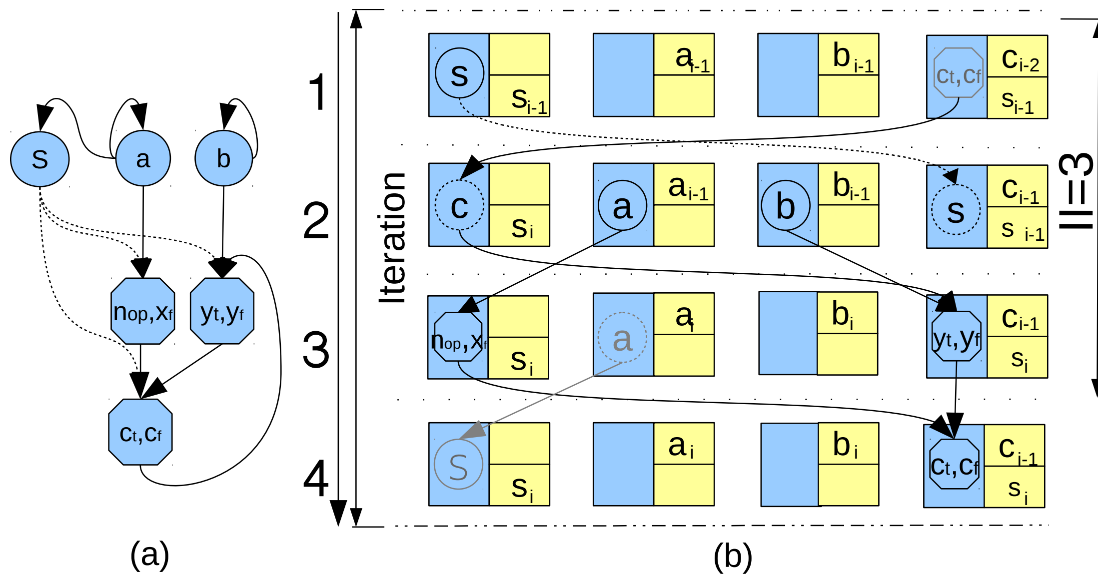


Figure 2.6: (a)Shows the DFG after Packed Node Formation (b) Shows a Valid Mapping Obtained for the DFG on a 2×2 CGRA. The II Obtained is 3.

This scheme aims to improve performance by issuing 2 instructions per PE simultaneously. One from the if-path and the other from the else-path. Nodes that have 2 instructions associated with them are called the packed nodes as shown by a octagon in figure 2.6(a). In case of unbalanced number of operations in the if and $else$ path nop$s$ are used to form the packed node (node nop and x_f in the example). At run-time, the PE selects one of those instructions based on a predicate value. In this case there is no need of a separate select instruction. This way, the problem of unnecessary execution of both the control flow paths in partial and full predication scheme is eliminated.

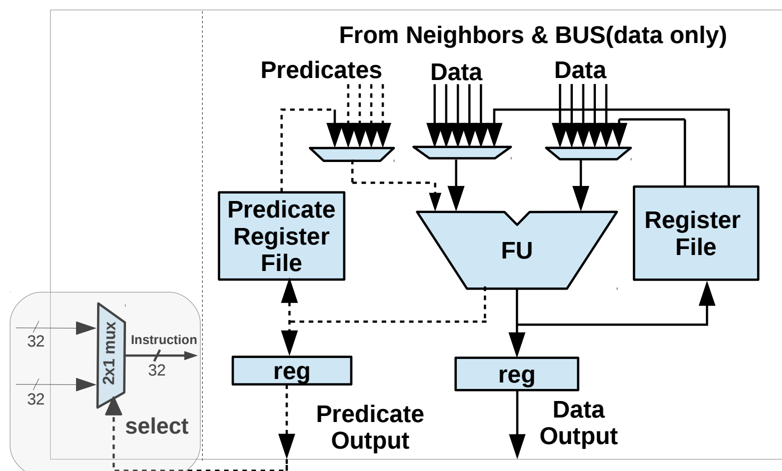


Figure 2.7: PE Architectural Template for Dual Issue Scheme.

The underlying PE structure for dual issue technique shown in figure 2.7 is the similar to the partial predication scheme. Only change that is required here is an additional 2x1 mux which selects either the true path instruction or the false path instruction. A predicate value serves as a select signal to choose 1 instruction from 2 available instructions to that PE at run time.

Dual issue scheme Han et al. [2010]Han et al. [2013a] mitigates some problems associated with partial and full predication schemes, however, this scheme has certain limitations. Firstly, if there are more than 1 instructions in the if block and the $else$

block, all those packed instructions belonging to the same branch should be mapped to the PEs which is hold the predicate value in its internal register. Since this predicate value is used to select the instruction for the packed node, this causes a restriction to map those packed nodes to the PEs where this predicate value is available. This problem is similar to the tight mapping restriction seen in full predication scheme. Secondly, this scheme has the following hardware penalties: First, the instruction fetch bandwidth has to be doubled and the input port size of the PE must be increased to receive two instructions. Then a 2x1 mux has to be added to the input of each PE to select an instruction. These hardware penalties become much more pronounced when the size of the PE array is large. This causes an overhead in the instruction fetch bandwidth of the overall CGRA. Hence the scalability of this scheme is limited.

Chapter 3

LIMITATIONS OF EXISTING TECHNIQUES

The fundamental limitations of the existing solutions to handle loops with control flow are twofold. Firstly, instructions from both the paths of the branch are fetched and issued unconditionally to the CGRA. This leads to more instructions being fetched than necessary, even after the branch outcome is known. For instance, partial and full predication schemes, even after the branch outcome is known at cycle 1, execute three unnecessary operations (x_f, y_f and c_f) if the condition evaluates true for the loop kernel presented in 2.2(b), incurring wastage of PE resources and an overhead in dynamic power by executing unnecessary operations. This poor utilization of PE resources in predication schemes results in inability to achieve low II. Even though the dual issue scheme avoids executing operations from the path not taken, it fetches unnecessary instructions (operations x_f, y_f and c_f) at run time incurring a power overhead in an instruction fetch operation. Secondly, there is a need to communicate the predicate value to operations in the if-path and the else-path. This communication is usually done either by storing the predicate value in the internal register of a PE or through the predicate network via routing. The need for this communication results in restrictions on where the conditional operations can be mapped, for instance, the select operations (c) in partial predication scheme can be mapped only to PE resources at which the corresponding predicate value is available, in full predication scheme, operations c_t, c_f should be mapped onto the same PE (PE1) where the predicate value is available. For dual issue scheme, the predicate value must be communicated to the packed nodes $\langle nop, x_f \rangle$,$\langle y_t, y_f \rangle$ and $\langle c_t, c_f \rangle$. These restrictions in mapping conditional operations leads to poor resource utilization. The impact

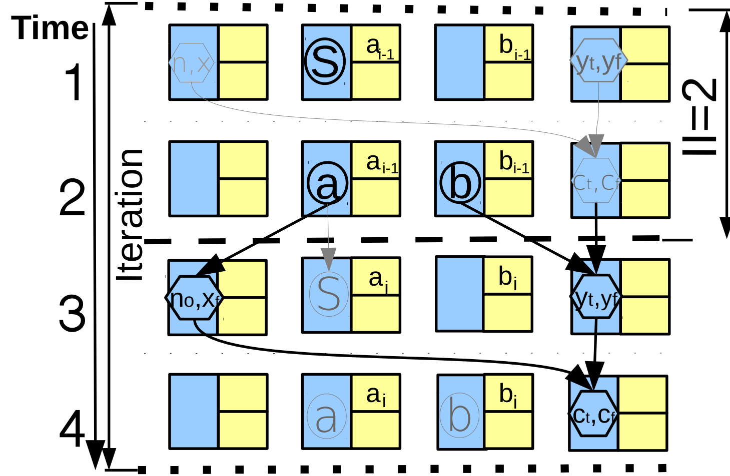
of these effects on performance and energy efficiency is worse especially when the number of instructions in the conditional path is more. In existing solutions, CGRA does not take advantage of the branch outcome, which is available at run time, and is oblivious of the path taken. Either all operations from both paths have to be executed in case of predication schemes, creating more nodes and edges in the DFG, or predicate value has to be communicated to all the packed operations (from both paths) in case of dual issue scheme causing a predicate communication overhead. In either case performance is limited because of the inability to achieve a low II. Moreover, energy efficiency is affected either by executing unnecessary operations (operations in the path not taken) in case of predications schemes or by fetching twice the number of instructions than required (dual issue scheme increases the dynamic power per instruction read operation). In this thesis I attempt to overcome these limitations to improve performance and energy efficiency of accelerating control flow loops.

Chapter 4

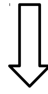
PROPOSED APPROACH: PSB

Considering that only one path is taken at run time for the if-then-else construct, the solution I propose, communicates the predicate (result of the branch instruction) to the instruction fetch Unit (IFU) of the CGRA, to selectively issue instructions only from the path taken by the branch at runtime. This is the essence of Path Selection based Branch (PSB) technique. This is similar to if-then-else execution in general purpose processors but while simultaneously taking advantage of parallelism available in the CGRA for performance improvement.

Figure 4.1 demonstrates how a loop body shown in Fig 2.2(b) can be transformed to work on a 2x2 CGRA as per the proposed approach. Figure 4.1(a) shows the scheduled loop body. It shows a schedule for 4 cycles, in which each cycle has operations for all the 4 PEs. Figure 4.1(b) shows the arrangement of instructions for the CGRA at a high level. In the first cycle, the branch operation $\langle blt\ a[i-1], S \mid 2\rangle$ is executed on PE 2, while the rest of the PEs are idle. The $\langle blt\ a, b \mid k\rangle$ operation compares branches if $a < b$. K represents the no.of cycles required to execute the branch path (max of the cycles required to execute the if-path or else-path). In this case the else-path is composed of instructions for all PEs at addresses 3 and 4, and it takes 2 cycles to execute the else-path operations. The if-path also takes 2 cycles, and is composed of instructions at addresses 5 and 6. Even though the condition in the branch operation executes in cycle 1, the operations in the if or else-path does not begin execution until cycle 3. Cycle 2 is the delay slot of the CGRA. In this cycle, operations independent of the current branch outcome can be executed. This delay slot cycle is used to communicate the branch outcome to the IFU. In this case,



(a)



	PE 1	PE 2	PE 3	PE 4
1	<idle>	<**blt** $a_{[i-1]}$, S ∣ 2>	<idle>	<idle>
2	<idle>	<$a_{[i]} \leftarrow a_{[i-1]} + C_1$>	<$b_{[i]} \leftarrow b_{[i-1]} - C_2$>	<idle>
3 F:	<$x_f \leftarrow a_{[i]} + C_4$>	<idle>	<idle>	<$y_f \leftarrow b_{[i]} \times C_5$>
4	<idle>	<idle>	<idle>	<$c_{[i],f} \leftarrow x_f - y_f$>
5 T:	<**nop**>	<idle>	<idle>	<$y_t \leftarrow b_{[i]} \times c_{[i-1]}$>
6	<idle>	<idle>	<idle>	<$c_{[i],t} \leftarrow y_t - C_3$>

(b)

Figure 4.1: A Valid Instruction Arrangement for PSB

operations $\langle a[i] = a[i-1]+C1\rangle$ and $\langle b[i] = b[i-1]-C2\rangle$ are executed on PEs 2 and 3. After the delay slot the Instruction Fetch Unit (IFU) will start issuing instructions from the path taken by the branch. If the else-path is taken, then instructions 3 and 4 will be issued. After executing else-path instructions, the IFU will skip the next s instructions, and start issuing instructions after that. If the branch is taken, then the IFU will skip s instructions and start issuing true path instructions.

For branch outcome based issuing of instructions, additional hardware support is required as shown in figure 4.2. The architecture of partial predication scheme

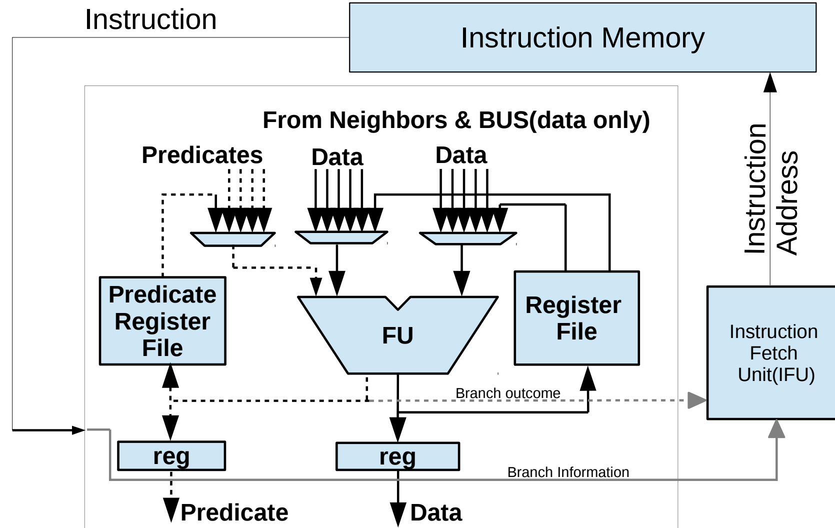


Figure 4.2: Architectural Support for the Proposed Approach. The Branch Parameters and Outcome is Communicated to the Instruction Fetch Unit (IFU) to Issue Instructions Only from the Path Taken at Run Time.

is extended to communicate the branch outcome to CGRA′s IFU along with the information of number of cycles to execute the branch. IFU is modified to issue instructions from the path taken based on branch information (outcome + no.of cycles for conditional path).

4.1 What Must the Compiler Do?

To enable such a branch based issuing of instructions, the compiler must map operations from the loop kernel, (including if-path, else-path and select or phi operations) onto the PEs of the time-extended CGRA. The PEs required to map the if-then-else portion of the loop kernel is the union of the PEs on which the operations from the if-path are mapped and the PEs on which the operations from the else-path are mapped. In case where operations from the if-path and the operations from the else-path are mapped to different PEs, the PEs on which the operations from the if-path are mapped will be inactive when the else-path is executed, and similarly when the if-path is executed the PEs on which the else-path operations are mapped

are not used. Figure 4.3(a) shows a mapping of operations where if-path operations and else-path operations are mapped onto different PEs. Corresponding instruction arrangement is shown in figure 4.3(b). In such a scheme, where if and else-path operations are mapped to different PE resources, the PEs allocated to execute the operations in the conditional path is the sum of the PEs required for the if-path operations and the PEs required for the else-path operations. But at run time only one of the paths is taken, and the PEs on which the other path operations are mapped will not be used, resulting in poor resource utilization and hence poor performance.

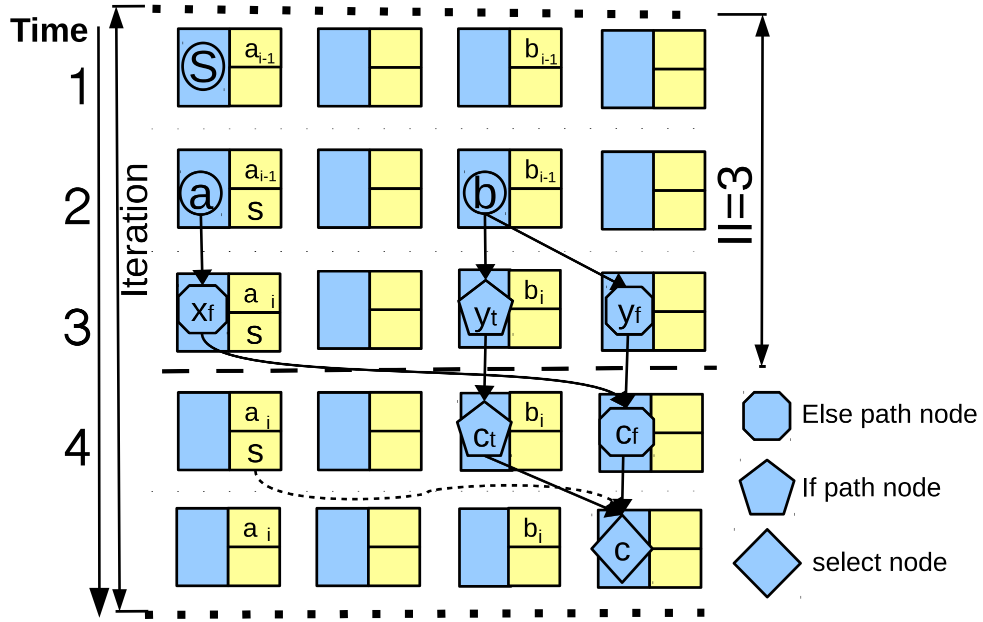
In order to improve the resource utilization and II, PSB maps the operations from the if-path and the operations from the else-path to the same PEs, so that the number of PEs used to map the if-then-else is equal to the maximum of the number of PEs required to map either path's operations. Hence, irrespective of the path taken by branch, the PEs that are allocated operations from the if-then-else path, executes a useful operation from the path taken. This facilitates better utilization of PE resources. By the virtue of improved resource utilization, more PEs are available to map operations from adjacent iterations enabling the use of a modulo scheduling scheme to further improve the performance.

Figure 4.1 shows the loop after it has paired operations per PE and software pipelined via a modulo scheduling scheme. Figure 4.4 shows the corresponding instruction arrangement. Even though the schedule length is 4, the II of this mapping is 2 – which is the best achieved for the loop kernel till now.

Since PSB issues instructions only from the path taken, it overcomes the inefficiencies associated with earlier techniques. PSB utilises the the branch outcome to improve performance and energy efficiency of control flow execution by eliminating issuing and execution of unnecessary operations. In predication based approaches instruction issuing is oblivious of the branch outcome, issuing and executing instructions

	PE 1	PE 2	PE 3	PE 4
1	<blt $a_{[i-1]}$,S ❘ 2>	<idle>	<idle>	<idle>
2	<$a_{[i]} \leftarrow a_{[i-1]} + C_1$>	<idle>	<$b_{[i]} \leftarrow b_{[i-1]} - C_2$>	<idle>
3	F:<$x_f \leftarrow a_{[i]} + C_4$>	*<idle>*	*<idle>*	<$y_f \leftarrow b_{[i]} \times C_5$>
4	*<idle>*	*<idle>*	*<idle>*	<$c_{[i],f} \leftarrow x_f - y_f$>
5	T: *<idle>>*	*<idle>*	<$y_t \leftarrow b_{[i]} \times c_{[i-1]}$>	*<idle>*
6	*<idle>*	*<idle>*	<$c_{[i],t} \leftarrow y_t - C_3$>	*<idle>*
7	*<idle>*	*<idle>*	*<idle>*	*<select(ct, cf, S)>*

(a)



(b)

Figure 4.3: Selective Instruction Issuing Without Pairing of If-Path and Else-Path Operations. a) Shows Instruction Arrangement b) Shows Mapping of the Kernel with Poor Resource Utilization of PEs.

from the path not taken, resulting in poor resource utilization and energy efficiency. Compared to Dual issue scheme, in addition to eliminating the need to fetch two instructions per PE, PSB also alleviate the overhead of communicating the predicate value to all the nodes that execute instructions from the conditional block.

	PE 1	PE 2	PE 3	PE 4
1	F: <$x_f \leftarrow a_{[i]} + C_4$>	***<blt $a_{[i-1]}$, S | 2>***	<idle>	<$y_f \leftarrow b_{[i]} \times C_5$>
2	<idle>	***<$a_{[i]} \leftarrow a_{[i-1]} + C_1$>***	***<$b_{[i]} \leftarrow b_{[i-1]} - C_2$>***	<$c_{[i],f} \leftarrow x_f - y_f$>
3	T: ***<nop>***	<blt $a_{[i-1]}$, S | 2>	<idle>	***<$y_t \leftarrow b_{[i]} \times c_{[i-1]}$>***
4	<idle>	<$a_{[i]} \leftarrow a_{[i-1]} + C_1$>	<$b_{[i]} \leftarrow b_{[i-1]} - C_2$>	***<$c_{[i],t} \leftarrow y_t - C_3$>***

Figure 4.4: Arrangement of Instructions for the Loop Kernel After Modulo Scheduling.

4.2 Problem Formulation

To optimize the resource usage and improve performance, PSB needs to pair operations from the if-path and the else-path to be mapped to the same PE. Hence, I define the problem formulation as obtaining a valid pairing of operations from the if-path and the else-path. The pairing must be done in such a way the the correct functionality of the loop kernel is maintained.

Problem is formulated as finding a transformation $\emptyset_T(D) = P$ from the input DFG: D = (N, E) to an output DFG: $P(M, R)$ with fused nodes, with the objective of $|M| <= |N|$ (N and M represent the set of nodes in D and P) while maintaining functional equivalence between D and P.

Inputs: DFG: $D = (N, E)$ is a data flow graph that represents the loop kernel, where the set of vertices N are the operations in the loop kernel, and for any two vertices, $u, v \in N, e = (u, v) \in E$ iff the operation corresponding to v is data dependent or predicate dependent on the operation u. For a loop with control flow $N = \{N_{if} \cup N_{else} \cup N_{other}\}$ where $\{N_{if}\}$ is the set of nodes representing the operations in the if-path and likewise N_{else} for operations in the else-path. N_{other} is the set of nodes representing operations not in the if-path or the else-path and includes select operations. If a variable is updated in more than one path, a select operation (or phi operation) is required to select the right output based on the branch outcome.

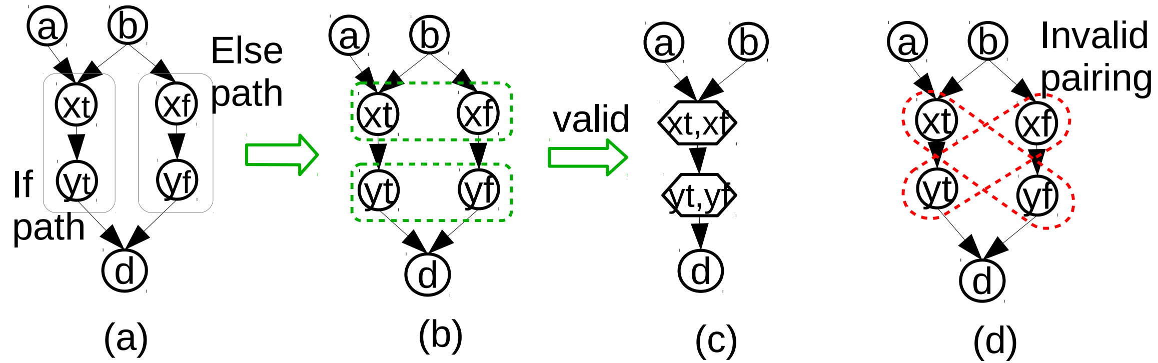


Figure 4.5: (a)(b)(c) Shows a Valid Pairing of Operations from the If and Else-Path. (d) Shows an Invalid Pairing since such a Pairing Fails to Meet the Criteria for Validity and a Feasible Schedule for such a Pairing Does Not Exist.

Output: DFG: $P = (M, R)$: Where M is the set of nodes in the transformed DFG representing the operations in the loop kernel with $M = \{M_{fused} \cup M_{other}\}$. The nodes M_{fused} represent the fused nodes. Each fused node $m \in M_{fused}$ is a tuple $m = \langle m_{if}, m_{else} \rangle$, where $m_{if} \in N_{if} \cup \{nop\}$ and $m_{else} \in N_{else} \cup \{nop\}$. For nodes $x, y \in M_{fused}, r = (x, y) \in R$ iff there is an edge $e_{if} = (x_{if}, y_{if}) \in E$ or an edge $e_{else} = (x_{else}, y_{else}) \in E$. For nodes $x_{other} \in M_{other}, y \in M_{fused}, r = (x_{other}, y) \in R$ iff there is an edge $e_{if} = (x_{other}, y_{if}) \in E$ or an edge $e_{else} = (x_{other}, y_{else}) \in E$ where $x_{other} \in N_{other}$. For nodes $x \in M_{fused}, y_{other} \in M_{other}, r = (x, y_{other}) \in R$ iff there is an edge $e_{if} = (x_{if}, y_{other}) \in E$ or an edge $e_{else} = (x_{else}, y_{other}) \in E$ where $y_{other} \in N_{other}$.

Valid Output: The output DFG P obtained after transformation is valid iff: For two vertices x,y with $x = (x_{if}, x_{else}), y = (y_{if}, y_{else}) \in M_{fused}$ and $r = (x, y) \in R$ then if there is a path from x_{if} *to* y_{if} then there is no path (intra-iteration) from y_{else} *to* x_{else} and if there is a path from x_{else} *to* y_{else} there is no path (intra-iteration) from y_{if} *to* x_{if} originally in the input DFG. However, recurrence paths satisfying inter iteration dependencies are valid. Figure 4.5 shows an example each for a valid paring (4.5(b),(c)) and an invalid pairing (4.5(d)).

Optimization: Objective is to minimize the number of PEs used to map the loop kernel such that the resulting mapping results in good resource utilization :

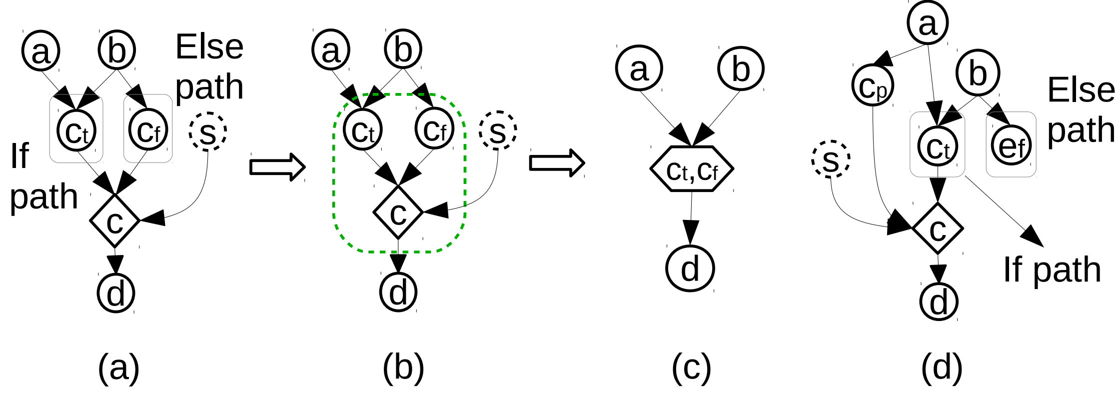


Figure 4.6: (a)(b)(c) Shows Elimination of Eligible PHI/Select Operation with Inputs from If-Path and Else-Path, (d) Shows an Example of a PHI that Cannot be Eliminated to Form a Fused Node since One of its Input Does Not Belong to the Set of If or Else-Path Operations.

minimising $|M|$ can be achieved by minimizing $|M_{fused}|$ and $|M_{other}|$. The proposed approach can minimise M_{fused} by minimizing the number of nops used to make a pair. $|M_{other}|$ can be minimised by eliminating the eligible select or phi operations that belong to N_{other}.

Phi Operation Elimination If a variable which is updated in both the if-path and the else-path serves as an input to an operation after the conditional block, a select operation is used to select the right value from either path based on the branch taken at runtime. Each select instruction has three inputs: an input from if-path, an input from else-path and a predicate boolean input to choose among former two. If the if-path operation and the else-path operation updating the same variable is paired to form a fused node, there is no need for a select operation since at run time only of the paths is executed. Hence, the output of the fused node has the right value after branch execution and can serve as an input to a node which requires this updated value. Figure 4.6(a)(b)(c) shows an scenario in which a select/phi operation can be eliminated. Figure 4.6(d) shows a scenario in which one of the inputs to the phi operation is not from either $\{N_{if}\}$ or $\{N_{else}\}$ of the current iteration, such a phi

node cannot be eliminated.

4.3 My Heuristic

Algorithm 1: PSB (Input $DFG(D)$, Output $DFG(P)$)

1 $n_{if} \leftarrow getLastNode(\{N_{if}\})$;

2 $n_{else} \leftarrow getLastNode(\{N_{else}\})$;

3 **while** ($n_{if} \neq NULL$ *or* $n_{else} \neq NULL$) **do**

4   **if** $n_{if} \in N_{if}$ *and* $n_{else} \in N_{else}$ **then**

5   fuse(n_{if}, n_{else});

6   **else if** $n_{if} \in N_{if}$ *and* $n_{else} == NULL$ **then**

7   fuse(n_{if}, nop);

8   **else if** $n_{if} == NULL$ *and* $n_{else} \in N_{else}$ **then**

9   fuse(nop, n_{else});

10     $n_{if} \leftarrow getLastRemainingNode(\{N_{if}\})$;

11     $n_{else} \leftarrow getLastRemainingNode(\{N_{else}\})$;

12 **for** n_i *such that i=0 to* $|N|$ **do**

13   **if** n_i *is an eligible select operation* $\in N_{other}$, $\ni input_1(n_i),\ input_2(n_i) = m_{fused} \in M_{fused}$ **then**

14         $Eliminate_{phi}(n_i)$;

15 Remove_Redundant_Arcs(E);

16 Prune_Predicate_Arcs(E);

The process of creating a DFG from CFG of a loop is presented in Johnson and Pingali [1993]. The operations from the if-path and else-path form the set of operation N_{if} and N_{else} respectively. The algorithm for forming the DFG with fused node is

shown in Algorithm 1. The algorithm starts with pairing of operations from if-path and else-path. Pairing starts from the terminating operation in both the paths as shown in lines 1,2 in alg. 1. Then the pairing proceeds iteratively through the predecessors of the fused nodes as long as there are unbalanced operations in the if-path and the else-path. Please note that the operations in the if and else-path have a partial order associated with them which is according to the order in which the operations appear in the if block and the else block of the CFG. If the operations in the if and else-path are unbalanced, unbalanced operations are paired with a nop, lines 7,9 in alg. 1. After all operations in the if and else path are paired, eligible select operations which have both the inputs from the same fused node are eliminated via a phi elimination pass, line 14 in alg. 1. Then the redundant edges are between the same nodes are eliminated and predicate arcs are pruned and final output DFG (P) is obtained. The DFG is given as an input to any mapping algorithm to find a valid mapping. However, the mapping algorithm must accommodate the delay slot in its mapping such that the fused nodes are scheduled with 1 cycle delay after the branch operation.

Proof: Next, I present the proof of correctness of the proposed algorithm. For nodes $x_t, y_t \in N_{if}$ and $x_f, y_f \in N_{else}$, such that the partial order of operations in the DFG is $x_t < y_t$ in the if block and $x_f < y_f$ in else block, meaning y_t cannot be scheduled earlier than x_t and y_f cannot be scheduled earlier than x_f. An incorrect pairing is $\langle x_t, y_f \rangle$ and $\langle y_t, x_f \rangle$ as shown in fig. 4.5(d). Since the algorithm starts pairing from the terminating nodes of if-path and else-path, nodes $\langle y_t, y_f \rangle$ in this example and proceeds upward iteratively through the partial order of nodes in the if-path and else-path forming another valid pair $\langle x_t, x_f \rangle$, there is no possibility of breaking the partial order in the process of pairing the operations from both the paths. Hence the algorithm always produces valid a pairing of operations.

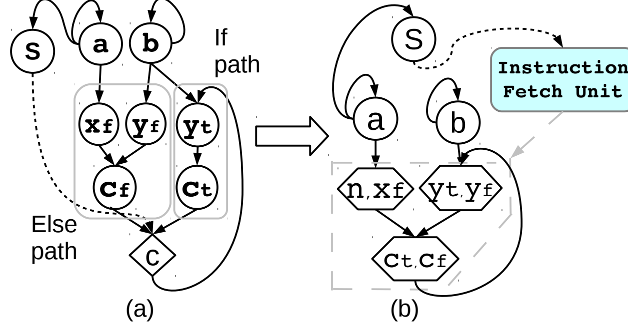


Figure 4.7: Shows Construction of DFG with Fused Nodes from an Input DFG.

Time Complexity: Since the pairing happens as long as there are unpaired operations in the if-path and else-path, the time complexity for pairing the operations in O $max\{|N_{if}|, |N_{else}|\}$ whereas phi eligibility is checked for each node $n \in N_{other}$, hence time complexity for checking phi node eligibility is O($|N_{other}|$). Hence, the overall time complexity of the algorithm is determined to be finite : O($max\{|N_{if}|, |N_{else}|\} + |N_{other}|$)

Figure 4.7 demonstrates how the kernel in figure 2.2 is mapped using PSB. In the proposed approach PSB first pair operations from either path to form a fused node. The pairing starts from the terminating nodes c_t and c_f in the if-path and the else-path respectively. Next, the predecessor operations of the fused node from the if-path and the else-path are paired to form a fused node, $\langle y_t, y_f \rangle$ represents such a pairing, where y_t is an operation from if-path and y_f is an operation from the else-path. Since the number of operations in the if-path is less the the operations in else-path, the unpaired else-path operation x_f is paired with a nop to form a fused node $\langle nop, x_f \rangle$. In this example, the phi operation c has both of its inputs from the same fused node $\langle c_t, c_f \rangle$ and hence it is eligible for elimination. Hence the corresponding phi node c, operation c_t and c_f are transformed into single $\langle c_t, c_f \rangle$ node. The loop kernel after pairing of operations is shown in figure 4.7(b). A DFG for the transformed code is shown in figure 4.7(c). Then dependency edges between all nodes $m \in M$ are

updated. Redundant edges are removed creating a DFG composed of fused nodes as shown in 4.7(c). Figure 4.7(d) shows a valid mapping of the DFG with modulo scheduling scheme. The achieved II =2 which is the lowest compared to all other techniques.

Support for Nested Conditionals: PSB approach provides maximum performance benefit when the percentage of conditional operations in the loop kernel is large. Hence for nested conditionals, the formation of fused nodes is done for the outermost conditional block. All the instructions from the true path and false path of the outermost branch is packed to form the fused node. Since the number of nodes for the inner nests of loops are typically small, the nodes for the inner nests of the conditional blocks are created from predication based transformation. Therefore, true and false path operations of the fused nodes are inherently composed of their respective path′s inner conditional blocks. For this reason, it is necessary to retain predicate dependencies for the inner conditional blocks. A partial predication scheme is preferred over full predication because full predication imposes tight restriction on where the operations inside the conditional block can be mapped.

Chapter 5

EXPERIMENTAL RESULTS

5.1 Experimental Setup

I have modelled CGRA as an accelerator in GemV simulation environment Binkert et al. [2011]. I have integrated PSB compiler technique as a separate pass in the LLVM compiler framework Lattner and Adve [2004]. Computational loops with control flow are extracted from SPEC2006 Henning [2006], biobench benchmarks Albayraktaroglu et al. [2005]. The CFGs of the loops are obtained after -O3 optimization. I explore and compare performance and power consumption of the techniques proposed in related work with PSB solution. PSB maps the loops on a 4×4 regular mesh interconnected CGRA with sufficient instruction memory to hold all instructions within a loop body.

5.2 PSB Achieves Lower II Compared to Existing Techniques to Accelerate Control Flow

First, I compare the performance of the proposed PSB technique with existing techniques to accelerate loops with control flow. I obtain the reduced DFG after PSB transformation and map it using REGIMap mapping algorithm Hamzeh et al. [2013] to obtain II. Figure 5.1 plots the achieved II for the loops mapped by different techniques. PSB solution to accelerate loops with control flow achieves the best performance(lowest II). The full predication technique presented in Han et al. [2013b] which is a special version of a full predication technique, achieves the worst II since all the instructions in a conditional path are mapped onto the same PE resulting in tight restriction in mapping and an increase in schedule length. Partial predication Hamzeh

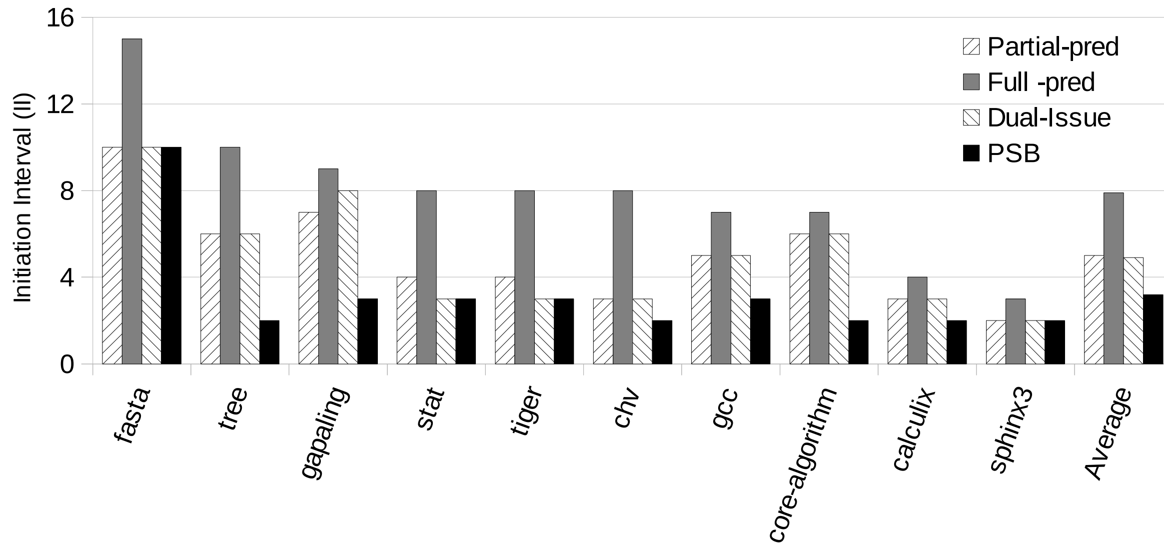


Figure 5.1: Performance of Compiled Loops Using i) Partial Predication Mahlke et al. [1992], ii) Full Predication Han et al. [2013b], iii) Dual-Issue Han et al. [2010], iv) PSB on a 4x4 CGRA

et al. [2014] achieves a better mapping compared to full predication technique since it does not have the restriction in mapping and just adds select nodes to the DFG. Dual issue scheme achieves relatively better II since it is able to pack the nodes whenever packing cost is low. Details about compiler implementation of dual issue scheme is presented in Han et al. [2010]. PSB solution achieves best II due to the following reasons:

1 Since PSB packs instructions from the if path and the $esle$ path to form fused nodes, the node size of the DFG is significantly reduced. Moreover, the edges for the fused nodes are a union of edges for the primary and the secondary instruction of the fused node. This way any redundant edges to a fused node is removed. Further more, all predicate edges but the ones from the dominating compare node to its immediate fused nodes are removed. This further reduces the edge size of the DFG. By virtue of these properties, PSB is able to reduce the size of the input DFG to the mapping algorithm. Figure 5.3 shows the

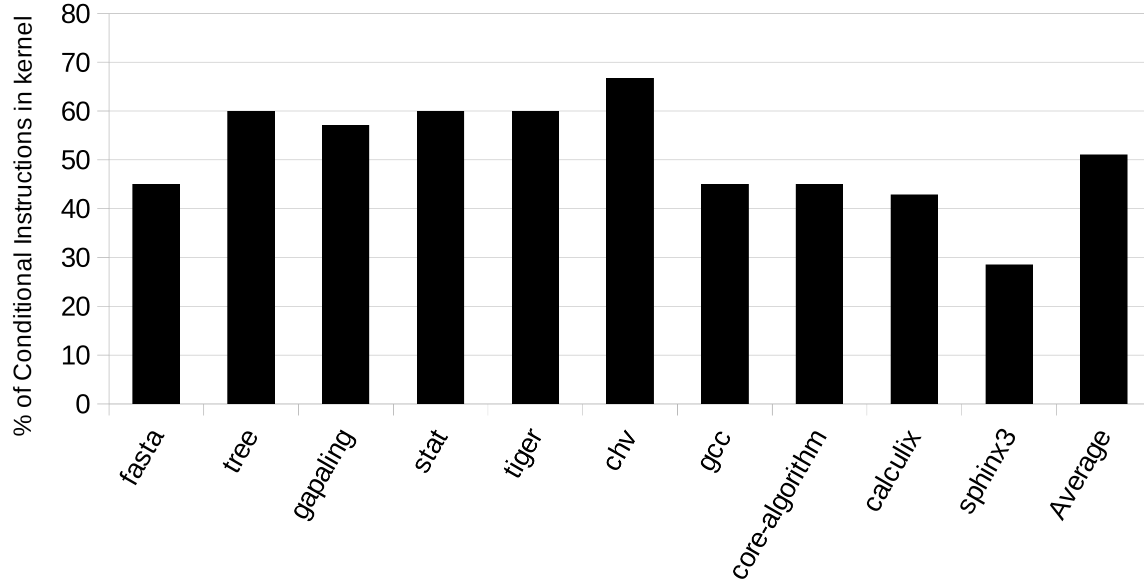


Figure 5.2: % of Conditional Instructions in a Loop Kernel Prior to DFG Formation.

percentage reduction of the input DFG for PSB approach compared to other techniques. From figure it is evident that for loops with good percentage of reduction in DFG size, PSB is able to achieve a good reduction in II compared to other techniques. For the loop kernel of $core - algorithm$ where the number of instructions in the if path and $else$ path is heavily unbalanced(more than 60% of packed nodes are unbalanced), the the scope for good reduction in node size of the DFG is low since PSB packs the unbalanced instructions with nop instruction. In spite of low percentage reduction in node size, PSB is still able to achieve a low II by eliminating the need to communicate the predicate value to all instructions inside the conditional block.

2 When the percentage of conditional instructions in the loop kernel is high, there is good scope for reducing the size of DFG. Hence the percentage reduction in DFG size by PSB is proportional to the percentage of conditional instructions in the loop kernel. Figure 5.2 plots the percentage of conditional instructions in a loop kernal prior to DFG formation. The average percentage of conditional

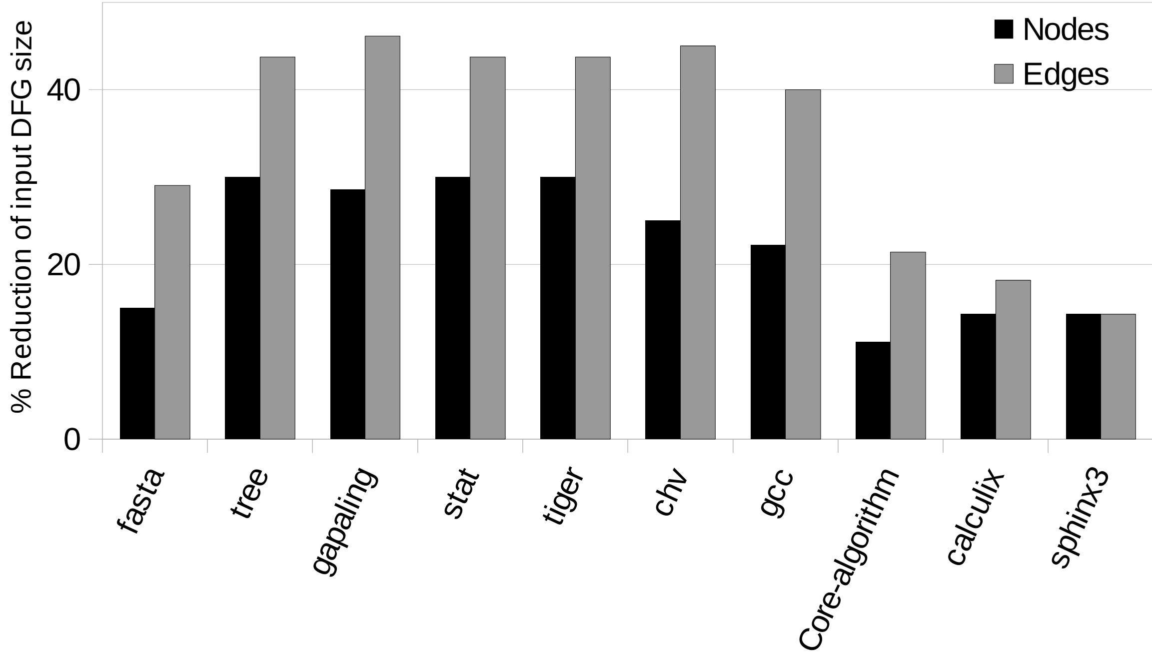


Figure 5.3: % Reduction of Input DFG Size in Terms of Nodes and Edges for PSB Compared to other Approaches. On an Average 40% or More Reduction in Edge Size and 20% or More Reduction in Node Size Translates to Achieving Lower II Compared to other Techniques.

instructions is 51% for the extracted loop kernels. As it can be seen from the figure, when the percentage 45% or more on an average, there is a good reduction in DFG size which in turn leads to achieving a lower II as shown in 5.1. It is observed that the kernel in *sphinx*3 has fewer conditional instructions which leads to poor reduction in DFG size which in turn leads to poor reduction in achieved II compared to all the existing techniques. Hence I deduce that PSB has the best performance improvement over existing techniques when accelerating loops kernels which have 45% or more conditional instructions.

3 Compared to a full predication scheme presented in Han et al. [2013b] there is no restriction on where the conditional operations must be mapped. Hence PSB is able to efficiently explore the solution space by utilizing all the available PE resources.

4 Compared to dual issue scheme there is no overhead of communicating or routing the predicate value to each of the dependent packed nodes.

5.3 PSB Architecture has Comparable Area and Frequency with Existing Solutions

CGRA	Partial Predication	Full Predication	Dual Issue	PSB
Area (sq.um)	375708	384539	411248	384154
Frequency (MHz)	463	477	454	458

Figure 5.4: Hardware Overhead of Supporting Existing Acceleration Techniques used for Executing Loops with Control Flow on 4×4 CGRA with Torus Interconnection Network

Next I compare the area, frequency and power associated with PSB architecture with existing architectures. I implemented an RTL model of a $4 times 4$ CGRA with mesh interconnect network including the Instruction fetch unit for all CGRA architectures. The RTL models were synthesized using 65nm technology library using RTL compiler tool. The models were verified for functionality after synthesis. To obtain the accurate impact of predicate communication in a PSB architecture on the overall frequency and area of CGRA, place and route was performed using Cadence Encounter tool. Final numbers for all designs after place and route, optimized for maximum frequency, without any timing violations, are reported in table 5.4. From the table it is seen that that PSB architecture does not incur any significant overhead in frequency and area and is comparable with existing solutions.

5.4 PSB Achieves Higher Energy Efficiency Compared to Existing Techniques

In this section I evaluate the energy efficiency in executing loop kernels for each benchmark. The power expenditure for each PE for an activity factor of 0.2 is ob-

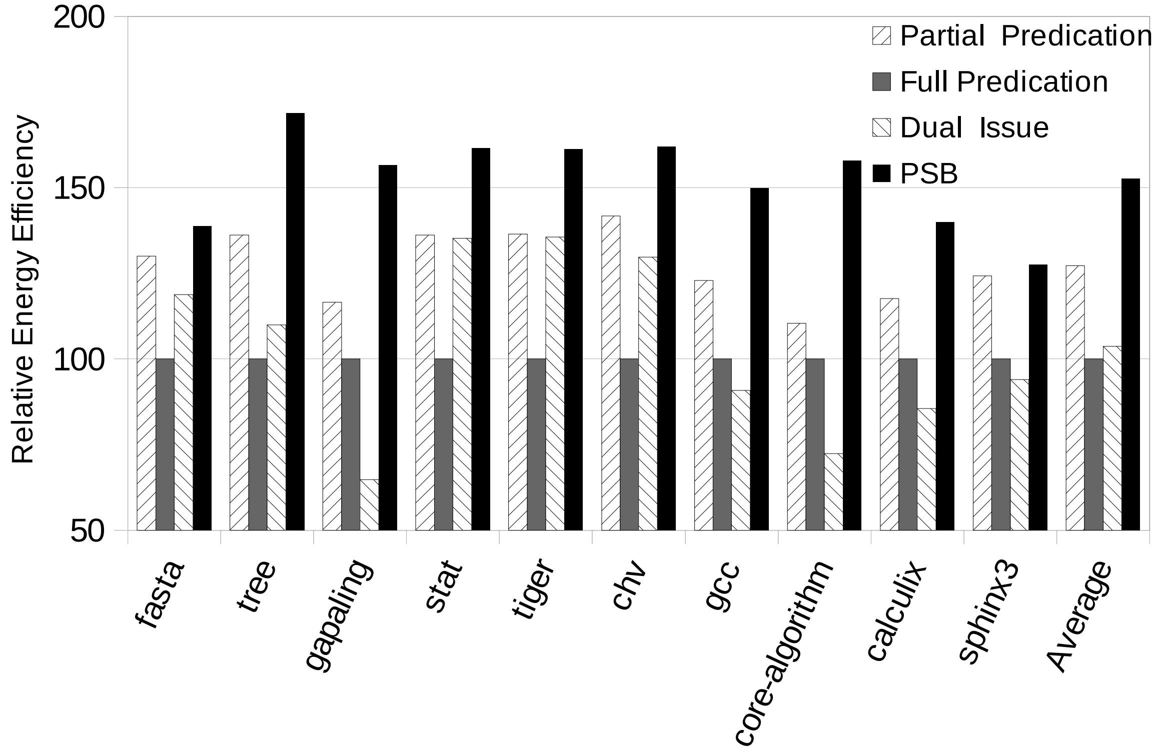


Figure 5.5: Estimation of Relative Energy Efficiency Normalised with Respect to Full Predication Technique for Executing the Kernel of Each Benchmark.

tained from the synthesized netlist and estimates of dynamic power for each type of operation (ALU, routing and IDLE) is scaled to fit the power distribution model presented in Kim et al. [2012]. The power spent by the basic IFL is only 0.4% of the total CGRA power and only 1.9% of the total CGRA power for a PSB version of the IFL. The configuration cache assumed in the model is a 2kb cache, implemented in 65 nm technology, with 16 read ports. The number of bits read per port is 64 bits for a dual issue scheme and 32 bits for PSB, partial and full predication scheme. This cache is modeled in cacti 5.3 tool CACTI [2008] to obtain the total dynamic power per read operation. The total power spent in executing kernel instructions of each benchmark is modeled as the function of the power spent per PE per cycle depending upon the PE operation(ALU,routing or IDLE) and the instruction fetch power from the configuration memory. Results are presented in figure 5.5. Experimental results

show that energy efficiency of 52.1% on an average can be obtained compared to dual issue scheme. This power saving is obtained by virtue of reduced II(34.6% improvement) and also reduced power consumption(34.9% power saving while fetching a 32 bit instruction compared to a 64 bit instruction) during an instruction fetch operation. An average energy efficiency of 53.9% and 35.5% was obtained compared to full predication and partial predication schemes.

5.5 Instruction Memory Overhead in PSB is Tolerable

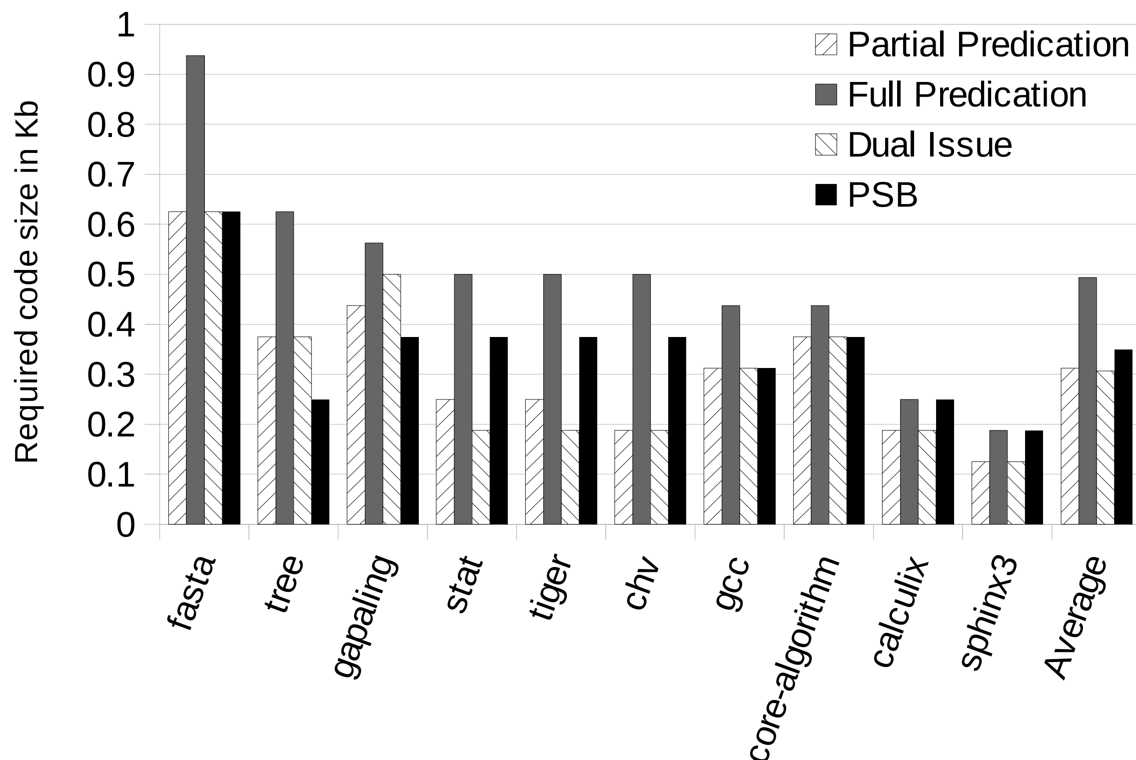


Figure 5.6: Required IMEM Size in Kb for the Benchmarks Used.

I have also estimated the required instruction memory size (figure 5.6) to hold the kernel instructions for each benchmark for all approaches. I assumed an ISA with 32 bits for normal instructions and 64 bits for dual issue scheme. My instruction memory model assumes to have a 32 bit instruction for each PE for each cycle of

the kernel. For instance, a benchmark with II of 2 would require 2 x 16 x 32 bits of instruction memory to hold all instructions. Since Dual issue scheme requires to fetch two instructions per PE in a cycle this causes an overhead in instruction memory size. Even though actual instructions for non-packed nodes or normal nodes needs to be only 32 bits, architecture for dual issue scheme needs to fetch 64 bit instructions. In case of normal node half of the instructions bits are unused and in case of packed nodes full 64 bit is used to store the instruction for the true path and the false path. To overcome this wastage of instruction memory associated with dual-issue scheme, Han et al. [2010] came up with an optimized instruction memory arrangement with interleaving of normal node instructions in a dual instruction, details are presented in Han et al. [2010]. To make a fair comparison, I performed experiments with this optimized instruction memory arrangement for dual-issue scheme. PSB approach selects the right instruction from the instruction block (true block or the false block) for cycles that have a fused node. Similar to the instruction arrangement for the motivating example as shown in figure4.1. Since there is replication of non-fused node instruction for those cycles which have fused nodes, one for the true block and other other for the false block, an overhead of 14.3% on an average compared to dual issue scheme and 13% on an average for the partial predication scheme is incurred in the required instruction memory size for PSB. This trade off is justified by improvement in performance and energy efficiency as shown in experimental results. However, compared to a full predication scheme there is saving of 29% on an average in the required instruction memory size. This is attributed to the poor II obtained via full predication scheme.

Chapter 6

SUMMARY

In this thesis I proposed a novel solution to accelerate control flow loops by utilising the branch outcome. My solution eliminates fetching and execution of unnecessary operations and also the overhead due to predicate communication thus overcoming the inefficiencies associated with existing techniques. Experiments on several benchmarks demonstrate that my solution achieves the best acceleration at minimum hardware overhead.

REFERENCES

K. Albayraktaroglu, A. Jaleel, X. Wu, M. Franklin, B. Jacob, C.-W. Tseng, and D. Yeung. Biobench: A benchmark suite of bioinformatics applications. In *Performance Analysis of Systems and Software, 2005. ISPASS 2005. IEEE International Symposium on*, pages 2–9, March 2005. doi: 10.1109/ISPASS.2005.1430554.

Vicki H. Allan, Reese B. Jones, Randall M. Lee, and Stephen J. Allan. Software pipelining. *ACM Comput. Surv.*, 27(3):367–432, September 1995. ISSN 0360-0300. doi: 10.1145/212094.212131. URL http://doi.acm.org/10.1145/212094.212131.

B. Betkaoui, D.B. Thomas, and W. Luk. Comparing performance and energy efficiency of fpgas and gpus for high productivity computing. In *Field-Programmable Technology (FPT), 2010 International Conference on*, pages 94–101, Dec 2010. doi: 10.1109/FPT.2010.5681761.

Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011. ISSN 0163-5964. doi: 10.1145/2024716.2024718. URL http://doi.acm.org/10.1145/2024716.2024718.

Frank Bouwens, Mladen Berekovic, Bjorn De Sutter, and Georgi Gaydadjiev. Architecture enhancements for the adres coarse-grained reconfigurable array. In *Proc. HiPEAC*, pages 66–81, 2008.

HP CACTI. Hp laboratories palo alto, cacti 5.3, 2008.

Allan Carroll, Stephen Friedman, Brian Van Essen, Aaron Wood, Benjamin Ylvisaker, Carl Ebeling, and Scott Hauck. Designing a coarsegrained reconfigurable architecture for power efficiency, department of energy na-22 university information technical interchange review meeting. Technical report, 2007.

Kyungwook Chang and Kiyoung Choi. Mapping control intensive kernels onto coarse-grained reconfigurable array architecture. In *SoC Design Conference, 2008. ISOCC '08. International*, volume 01, pages I–362–I–365, Nov 2008. doi: 10.1109/SOCDC.2008.4815647.

Shuai Che, Jie Li, J.W. Sheaffer, K. Skadron, and J. Lach. Accelerating compute-intensive applications with gpus and fpgas. In *Application Specific Processors, 2008. SASP 2008. Symposium on*, pages 101–107, June 2008. doi: 10.1109/SASP.2008.4570793.

Liang Chen and T. Mitra. Graph minor approach for application mapping on cgras. In *Field-Programmable Technology (FPT), 2012 International Conference on*, pages 285–292, Dec 2012. doi: 10.1109/FPT.2012.6412149.

E.S. Chung, P.A. Milder, J.C. Hoe, and Ken Mai. Single-chip heterogeneous computing: Does the future include custom logic, fpgas, and gpgpus? In *Microarchitecture (MICRO), 2010 43rd Annual IEEE/ACM International Symposium on*, pages 225–236, Dec 2010. doi: 10.1109/MICRO.2010.36.

Bjorn De Sutter, Praveen Raghavan, and Andy Lambrechts. *Handbook of Signal Processing Systems*, chapter Coarse-Grained Reconfigurable Array Architectures, pages 553–592. Springer, 2 edition, 2013. ISBN: 978-1-4614-6858-5.

M.D. Galanis, G. Dimitroulakos, and C.E. Goutis. Accelerating applications by mapping critical kernels on coarse-grain reconfigurable hardware in hybrid systems. In *Field-Programmable Custom Computing Machines, 2005. FCCM 2005. 13th Annual IEEE Symposium on*, pages 301–302, April 2005. doi: 10.1109/FCCM.2005.15.

M. Hamzeh, A. Shrivastava, and S. Vrudhula. Epimap: Using epimorphism to map applications on cgras. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pages 1280–1287, June 2012.

Mahdi Hamzeh, Aviral Shrivastava, and Sarma Vrudhula. Regimap: Register-aware application mapping on coarse-grained reconfigurable architectures (cgras). In *Design Automation Conference (DAC), 2013 50th ACM / EDAC / IEEE*, pages 1–10, May 2013.

Mahdi Hamzeh, Aviral Shrivastava, and Sarma Vrudhula. Branch-aware loop mapping on cgras. In *Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference*, DAC '14, pages 107:1–107:6, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2730-5. doi: 10.1145/2593069.2593100. URL `http://doi.acm.org/10.1145/2593069.2593100`.

Kyuseung Han, Jong Kyung Paek, and Kiyoung Choi. Acceleration of control flow on cgra using advanced predicated execution. In *Field-Programmable Technology (FPT), 2010 International Conference on*, pages 429–432, Dec 2010. doi: 10.1109/FPT.2010.5681452.

Kyuseung Han, Junwhan Ahn, and Kiyoung Choi. Power-efficient predication techniques for acceleration of control flow execution on cgra. *ACM Trans. Archit. Code Optim.*, 10(2):8:1–8:25, May 2013a. ISSN 1544-3566. doi: 10.1145/2459316.2459319. URL `http://doi.acm.org/10.1145/2459316.2459319`.

Kyuseung Han, Kiyoung Choi, and Jongeun Lee. Compiling control-intensive loops for cgras with state-based full predication. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 1579–1582, March 2013b. doi: 10.7873/DATE.2013.321.

R. Hartenstein. A decade of reconfigurable computing: a visionary retrospective. In *Design, Automation and Test in Europe, 2001. Conference and Exhibition 2001. Proceedings*, pages 642–649, 2001. doi: 10.1109/DATE.2001.915091.

A. Hatanaka and N. Bagherzadeh. A modulo scheduling algorithm for a coarse-grain reconfigurable array template. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–8, March 2007. doi: 10.1109/IPDPS.2007.370371.

John L. Henning. Spec cpu2006 benchmark descriptions. *SIGARCH Comput. Archit. News*, 34(4):1–17, September 2006. ISSN 0163-5964. doi: 10.1145/1186736.1186737. URL http://doi.acm.org/10.1145/1186736.1186737.

Richard Johnson and Keshav Pingali. Dependence-based program analysis. In *Proceedings of the ACM SIGPLAN 1993 Conference on Programming Language Design and Implementation*, PLDI '93, pages 78–89, New York, NY, USA, 1993. ACM. ISBN 0-89791-598-4. doi: 10.1145/155090.155098. URL http://doi.acm.org/10.1145/155090.155098.

Yongjoo Kim, Jongeun Lee, Toan X. Mai, and Yunheung Paek. Improving performance of nested loops on reconfigurable array processors. *ACM Trans. Archit. Code Optim.*, 8(4):32:1–32:23, January 2012. ISSN 1544-3566. doi: 10.1145/2086696.2086711. URL http://doi.acm.org/10.1145/2086696.2086711.

Chris Lattner and Vikram Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization*, CGO '04, pages 75–, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2102-9. URL http://dl.acm.org/citation.cfm?id=977395.977673.

S.A. Mahlke, D.C. Lin, W.Y. Chen, R.E. Hank, and R.A. Bringmann. Effective compiler support for predicated execution using the hyperblock. In *Microarchitecture, 1992. MICRO 25., Proceedings of the 25th Annual International Symposium on*, pages 45–54, Dec 1992. doi: 10.1109/MICRO.1992.696999.

S.A. Mahlke, R.E. Hank, J.E. McCormick, D.I. August, and W.-M.W. Hwu. A comparison of full and partial predicated execution support for ilp processors. In *Computer Architecture, 1995. Proceedings., 22nd Annual International Symposium on*, pages 138–149, June 1995.

Hyunchul Park, Kevin Fan, Scott A. Mahlke, Taewook Oh, Heeseok Kim, and Hong-seok Kim. Edge-centric modulo scheduling for coarse-grained reconfigurable architectures. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pages 166–176, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-282-5. doi: 10.1145/1454115.1454140. URL http://doi.acm.org/10.1145/1454115.1454140.

Kara K. W. Poon, Steven J. E. Wilton, and Andy Yan. A detailed power model for field-programmable gate arrays. *ACM Trans. Des. Autom. Electron. Syst.*, 10 (2):279–302, April 2005. ISSN 1084-4309. doi: 10.1145/1059876.1059881. URL http://doi.acm.org/10.1145/1059876.1059881.

B. Ramakrishna Rau. Iterative modulo scheduling: An algorithm for software pipelining loops. In *Proceedings of the 27th Annual International Symposium on Microarchitecture*, MICRO 27, pages 63–74, New York, NY, USA, 1994. ACM. ISBN 0-89791-707-3. doi: 10.1145/192724.192731. URL `http://doi.acm.org/10.1145/192724.192731`.