A METHODOLOGY FOR EXPLORING COMMUNICATION ARCHITECTURES OF CLUSTERED VLIW PROCESSORS

ANUP GANGWAR

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY DELHI

©Indian Institute of Technology Delhi - 2005

All rights reserved.

A METHODOLOGY FOR EXPLORING COMMUNICATION ARCHITECTURES OF CLUSTERED VLIW PROCESSORS

by

ANUP GANGWAR

Department of Computer Science and Engineering

Submitted

in fulfillment of the requirements of the degree of Doctor of Philosophy

to the

Indian Institute of Technology Delhi

July 2005

Certificate

This is to certify that the thesis titled **A Methodology for Exploring Communication Architectures of Clustered VLIW Processors** being submitted by **Anup Gangwar** for the award of **Doctor of Philosophy** in **Computer Science & Engg.** is a record of bona fide work carried out by him under our guidance and supervision at the **Dept. of Computer Science & Engg., Indian Institute of Technology Delhi**. The work presented in this thesis has not been submitted elsewhere, either in part or full, for the award of any other degree or diploma.

M. Balakrishnan	**Anshul Kumar**
Professor	Professor
Dept. of Computer Science & Engg.	Dept. of Computer Science & Engg.
Indian Institute of Technology Delhi	Indian Institute of Technology Delhi

Acknowledgments

I am greatly indebted to my supervisors Prof. Anshul Kumar and Prof. M. Balakrishnan for their invaluable technical guidance and moral support. I would like to thank Basant Dwivedi for his feedbacks during our informal discussions and my other colleagues specially Manoj Jain, Subhajit Sanyal, Parag Chaudhuri and Rohit Khandekar for their cooperation and support. I would also like to thank the staff of Philips, FPGA and Vision laboratories, IIT Delhi for their help. I am very thankful to Naval Research Board, Govt. of India for being enabler of my work.

My parents showed immense patience and provided me great moral support during the course of my work. My other family members specially my elder sister also helped me a lot in getting me this far.

Abstract

VLIW processors have started gaining acceptance in the embedded systems domain. However, monolithic register file VLIW processors with a large number of functional units are not viable. This is because of the need for a large number of ports to support FU requirements, which makes them expensive and extremely slow. A simple solution is to break up this register file into a number of small register files with a subset of FUs connected to it. These architectures are termed as clustered VLIW processors.

This thesis focuses on customizing inter-cluster inter-connection networks (ICN) in high issue-rate clustered VLIW processors. While a wide variety of inter-cluster ICNs are reported in literature what is missing is a quantitative evaluation of this design-space. Researchers have used specific tools and methodologies for architecting such VLIW processors, wherein some of the other ICNs are qualitatively eliminated.

We build a basis for exploring high issue-rate processor by showing that on an average the media applications [could have an ILP of 20 or even higher]. Towards this end we classify the previous reported results on ILP measurement, coining a novel measurement technique, Achievable-H ILP, which is useful for predicting future architecture requirements. We present a methodology along with the supporting tool chain for exploring the design-space of inter-cluster ICNs. We also classify the previously used architectures and demonstrate that a vast part of this design-space is currently [unexplored]. We conclusively establish that most of the bus-based RF-to-RF style ICNs are heavily performance constrained. Finally to prove the superiority of point-to-point type ICNs we develop a parameterized clustered VLIW generator. Using the generated architectures as input to industry standard synthesis, place and

route tools we present results on the implementation characterstics of the various ICNs.

Contents

List of Tables v

List of Figures vii

1 Introduction 1

1.1 Background 1

1.2 Motivation 2

1.3 Brief Review of Previous Work 5

1.4 Overview of Our Work 7

1.5 Organization of Thesis 8

2 ILP Studies – A Review 9

2.1 Introduction 9

2.2 Classification of ILP Measurement Techniques 10

2.2.1 Available ILP 12

2.2.2 Achievable-H ILP 13

2.2.3 Achievable-S ILP 13

2.2.4 Achieved ILP 14

2.3 Previous Work 14

2.4 Evaluation Methodology 16
2.4.1 Trace Generation 18
2.4.2 DFG Generation 18
2.4.3 Instruction Scheduling 19
2.5 Experimental Results and Observations 19
2.5.1 Achievable-H ILP in Benchmarks 19
2.6 Exploiting the Achievable ILP 21
2.7 Summary 22

3 Inter-cluster Communication – A Review **24**
3.1 Previous Work 25
3.1.1 Previously Reported Architectures 25
3.1.2 Previous Classification of ICNs 28
3.2 Our Interconnection Design Space 31
3.3 Our Classification of Inter-cluster ICNs 34
3.4 Summary and Conclusions 35

4 Design Space Exploration Framework **37**
4.1 Existing Code Generation Techniques for Clustered VLIWs 38
4.2 Design Space Exploration Methodology 40
4.2.1 DFG Generation 42
4.2.2 Chain Detection 42
4.2.3 Singleton Merger 44
4.2.4 Clustering 45
4.2.5 Binding 50
4.2.6 Final Scheduling 53

4.3 Summary 55

5 Experimental Results for Design Space Exploration 56

5.1 Experimental Results 57

5.2 Summary 64

6 Effect on Clock Period of Inter-cluster Communication Mechanisms 65

6.1 Architecture of Modeled Processors 66

6.2 Evaluation Methodology 74

6.3 Results of Clock Period Evaluation 78

6.4 Cumulative Experimental Results and Observations 81

6.5 Summary 86

7 Conclusions 87

7.1 Summary 87

7.2 Main Contributions and Highlights of the Results 89

7.3 Future Work 90

Bibliography 91

List of Tables

2.1 Classification of ILP Measurement Techniques 11

2.2 Classification of Previous ILP Studies 16

3.1 Overall Design-Space for Direct Communication Architectures 35

4.1 Code Scheduling for Clustered VLIWs 40

5.1 ILP for (8-ALU, 4-MEM) 4-Clust Architectures 57

5.2 ILP for (12-ALU, 6-MEM) 6-Clust Architectures 58

5.3 ILP for (16-ALU, 8-MEM) 8-Clust Architectures 58

5.4 ILP for (20-ALU, 10-MEM) 10-Clust Architectures 59

5.5 ILP for (24-ALU, 12-MEM) 12-Clust Architectures 59

5.6 ILP for (28-ALU, 14-MEM) 14-Clust Architectures 60

5.7 ILP for (32-ALU, 16-MEM) 16-Clust Architectures 60

6.1 Instruction Set Architecture of Modeled Processors 67

6.2 Clock Period (ns) for UMC 0.13μ ASIC Tech. 79

6.3 Clock Period (ns) for UMC 0.18μ ASIC Tech. 79

6.4 Interconnect Area (as % of Chip Area) for UMC 0.13μ ASIC Tech. . 79

6.5 Interconnect Area (as % of Chip Area) for UMC 0.18μ ASIC Tech. . 80

List of Figures

1.1 Customization Opportunities in VLIW ASIPs 4

2.1 Source code for example.c 12

2.2 Evaluation Framework for ILP in Media Applications 17

2.3 *Achievable-H* ILP in MediaBench-II Applications 20

2.4 *Achievable-H* ILP in MediaBench Applications 20

2.5 *Achievable-H* ILP in DSPStone Kernels 21

3.1 Stanford Imagine Architecture 26

3.2 MIT RAW Architectures 27

3.3 Architectures from [Terechko et al., 2003] 29

3.4 Architectures From [Terechko et al., 2003] 30

3.5 Broadcast Architecture From [Terechko et al., 2003] 30

3.6 RF-to-RF ($\delta = 1$) 31

3.7 Architectures with ($\delta = n_clusters$) 32

3.8 Architectures with ($\delta = n_clusters/2$) 33

4.1 DSE Framework 41

4.2 Longest Chain Detection 43

4.3 Reasons for Singleton Generation 44

4.4 Detected Connected Components-I 49

4.5 Detected Connected Components-II 49

4.6 Heuristics for Binding 51

4.7 Connectivity and Criticality Between Clusters 54

5.1 Average Variation in ILP With $n_{clusters}$ 62

5.2 Average Variation of ILP as fraction of Pure VLIW With $n_clusters$. 63

6.1 Pipelined RF-to-RF 66

6.2 Processor Pipeline 68

6.3 Instruction Encoding 69

6.4 Detailed Architecture of Clusters for WA.1 70

6.5 Pin Organization for Various Clusters 75

6.6 Floorplans for RF-to-RF Architectures 77

6.7 Variation of Clock Period for Xilinx XC2V8000-FF1152-5 Device . . . 81

6.8 Bus Based Interconnects: Performance for Multi-cycle Configurations 83

6.9 Average Variation in Performance for Pipelined Bus Configurations . 84

6.10 $ILP_{effective}$ for UMC 0.13μ ASIC Technology 85

Chapter 1

Introduction

1.1 Background

Embedded computing platforms form a very important class of computing systems. The number of embedded computing systems deployed is already many orders of magnitude higher than general purpose computing systems and is on the rise. Current embedded computing systems rival the performance of personal computers available five years ago and the performance gap is decreasing. A very important class of these systems is the one dealing with processing of streaming media data. Streaming media applications typically are highly data and computation intensive. Also, since most of these devices are mobile there is a major constraint on power consumed by the entire system. Designing embedded systems for such applications poses a number of unique challenges.

Current and past trend in the industry has been towards the use of either vanilla RISC processors or those augmented with special purpose functional units (FUs). However, increasing performance requirements from these systems have forced the

designers to look for alternative solutions. Single-chip multiprocessors offer solutions which not only have higher performance but are also cheaper cost-wise. Choosing multiple-issue processors as compute engines in these multiprocessors, leads to efficient utilization of both coarse grain (thread level) as well as fine grained (instruction level) parallelism. Media applications typically exhibit enormous amounts of parallelism at both coarse and fine levels.

The Embedded Systems Group at IIT Delhi has been working towards an integrated methodology for synthesis of single-chip application specific multiprocessor systems under the research project titled Srijan [Balakrishnan et al., 2004]. Srijan attempts to develop embedded systems built using a variety of processing elements. The processing elements in Srijan can be either plain RISC processors, Application Specific Instruction Processors (ASIPs) or ASICs. Such a wide choice of processing elements offers tremendous amounts of architectural flexibility. However, to properly utilize this flexibility, efficient design-space exploration mechanisms need to be devised. This thesis focuses on customizations of individual processors with high performance requirements.

1.2 Motivation

VLIW processors are multiple-issue processors with instructions statically scheduled by a compiler and a parallel instruction word formed. This is in contrast to superscalar processors wherein the instructions are dynamically issued by a issue logic. The issue logic in superscalar processors dynamically schedules instructions by evaluating data dependency and resource contention. It tends to be very large and slow. Due to its high logic complexity it also consumes significant power and area. Whereas

superscalar processors are more popular in the general purpose computing domain, VLIW processors have gained wide acceptance in embedded systems domain. Some of the successful VLIW processors are: TriMedia, TiC6x, Equator's MAP-CA and MAP-1000 [Glaskowsky, 1998], HP-ST Microelectronic's Lx [Faraboschi et al., 2000]. This is primarily due to the fact that in embedded systems, application is known apriori and thus fine grain concurrency can be extracted by using both compilation as well as program transformation techniques. Due to the simplified hardware, VLIW processors are more amenable to customization and also consume less power and area. These have emerged as the processors of choice for implementing embedded systems.

Several customization opportunities exist in VLIW processors [Aditya et al., 1999] [Jacome and de Veciana, 2000] [Rau and Schlansker, 2001] [Schlansker and Rau, 2000]. The key customization domains which are also highlighted in Figure 1.1 are:

1. No. and types of functional units
2. Interconnection network both between FUs and between register files
3. Register file structure
4. Instruction encoding for NOP compression

Functional units lend themselves naturally to specializations. The core set of FUs shown in Figure 1.1, support the usual fine grain operations like add, multiply, compare etc. augmented by application specific extensions (shown as AFUs). These extensions are centered around some medium or coarse grain FUs defining new instructions for implementing some critical functionality of the specific application. Actually the fine grain core may not be absolutely rigid, it may be generic in

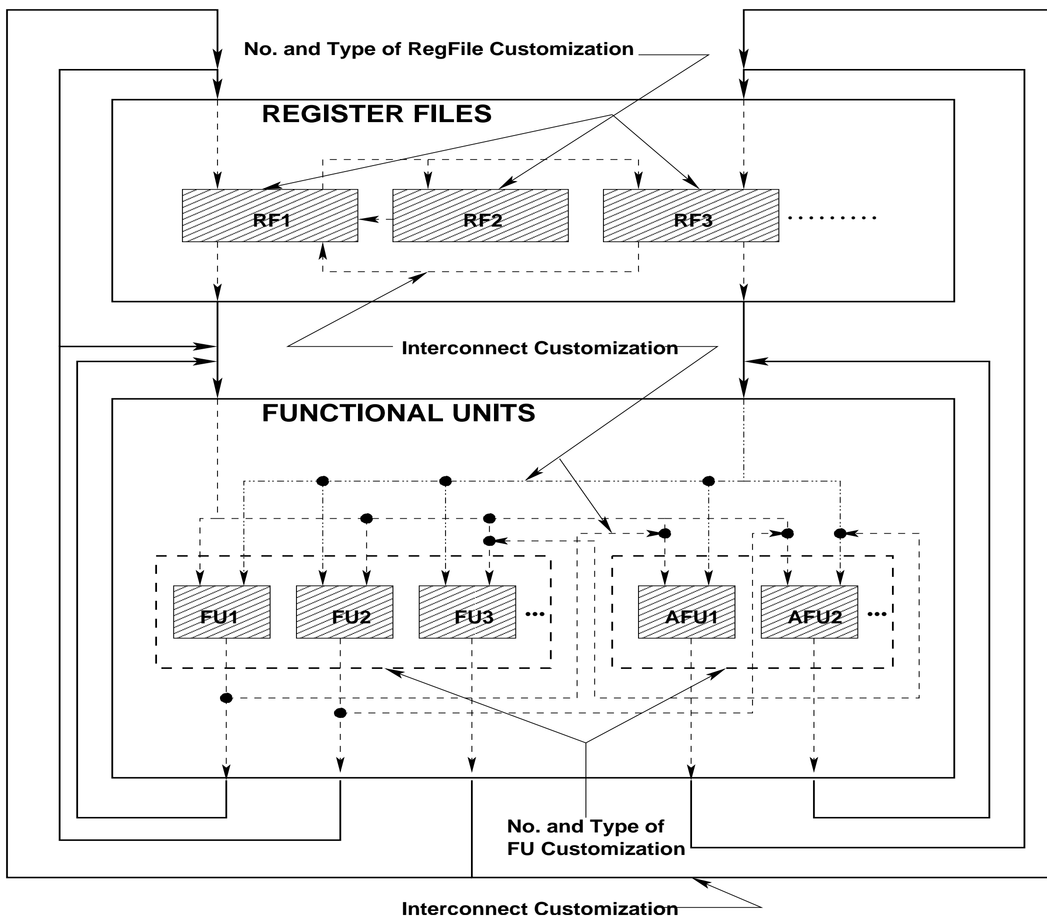


Figure 1.1: Customization Opportunities in VLIW ASIPs

some limited sense. The presence of core makes things easier by providing a default implementation for any part of the application.

However, in scaling the number of functional units to increase operation level concurrency, there are some bottlenecks. The most serious among these is the monolithic register file for supporting a large number of functional units. This is not viable because of the need for a large number of ports to support FU operand and result requirements. This makes the register file very expensive and slow. As per one reported work [Rixner et al., 2000], for N functional units connected to a RF the

Area grows as N^3, Delay as $N^{\frac{3}{2}}$ and Power as N^3. A simple solution is to break up this register file into a number of small register files with a subset of FUs connected to it. These architectures are termed as clustered VLIW processors. However this gain does come at a cost. In particular it may lead to data copying from one set of registers to another and thus increase execution latency.

Clustering functional units together implies deciding the interconnection strategy between different clusters as well as between clusters and memory. Application analysis for observing the data access patterns allows for evaluating the various cost and performance tradeoffs. Some vendors are offering ASIP cores with clustered architectures [Tencilica, 2003] [Siroyan, 2002]. However, the interconnection amongst clusters is fixed beforehand offering no possibility of customization. Figure 1.1, shows a few places in a VLIW architecture wherein interconnect specialization can be applied.

Code size minimization is an important problem for VLIW processors. If either the application does not contain enough parallelism or the compiler is not able to extract enough parallelism from it, then the VLIW instruction will contain a large number of NO-Ops (NOP). However any encoding strategy, to reduce the number of NOPs, affects the performance and memory bandwidth along with code size.

1.3 Brief Review of Previous Work

A wide variety of work is available for customizing VLIW processors based on application requirements e.g. [Rau and Schlansker, 2001] [Middha et al., 2002] [Jacome et al., 2000]. Predominantly, researchers have tried to customize the FUs based on application requirements. [Rau and Schlansker, 2001] describe an automated VLIW processors

customization system, Program In Chip Out (PICO). PICO system presumes an underlying Instruction Set Architecture (ISA) and is able to evaluate a range of different FU configurations. In turn it generates a set of pareto-optimal design points, where the metrics are processor performance, silicon area etc. The user can thus choose from one of these customized processor designs or constrain the system to provide other design alternatives. [Jacome et al., 2000] describe a framework, which can be used to quickly evaluate a range of processor configurations (number and type of functional units) for a given application. The underlying clustered VLIW processor is assumed to have a fixed inter-cluster interconnect. Our developed framework [Middha et al., 2002], too showed impressive performance gains for selected applications. We extended a state of the art compiler infrastructure, Trimaran [Trimaran Consortium, 1998], and ran the developed framework on few applications of interest. The FUs were detected manually and the input application specification (C-program) was also augmented manually towards this end. FU, customization is only one aspect of VLIW specialization.

Multiple-issue processors suffer from the cost limitations of a centralized RF [Rixner et al., 2000]. Clustering FUs along with RFs is an attractive solution to this problem. Clustering is nothing new, as a larger number of commercial as well as research architectures have had clustered architectures e.g. [Siroyan, 2002], TiC6x, Sun's MAJC, Equator's MAP-CA, TransMogrifier [Lewis et al., 1997]. There is wide variety in the inter-cluster ICN utilized in these architectures. While in some cases this clustering is visible to the compiler e.g. TiC6x, in others this clustering is compiler transparent e.g. Sun's MAJC. From the researcher's point of view, a fully connected architecture is simpler for compiler scheduling and thus with the exception of [Terechko et al., 2003], predominantly, they have used bus-based interconnects

[Faraboschi et al., 1998] [Jacome et al., 2000] [Sanchez and Gonzalez, 2000] [Balasubramonian, 2004] [Sanchez et al., 2002].

1.4 Overview of Our Work

We start by a classification and then a quantification of the achievable ILP in media applications. This is necessary to justify building high-issue rate processors with clustered architectures. It is tempting to quantify exploitable ILP looking purely at the algorithmic behavior of the application under consideration. However, any design-space exploration starts with an executable specification of the application. Thus it is important to consider the implementation limitations. We thus approach quantification by first picking up representative implementations of most commonly used media applications and kernels. Our benchmarks of choice are the widely used MediaBench Application Suite [Lee et al., 1997], the proposed MediaBench-II Application Suite[Fritts and Mangione-Smith, 2002] and DSPStone Kernels [Zivojinovic et al., 1994].

The objective of this thesis is to quantify the impact of inter-cluster ICNs on processors performance. Towards this end, we first provide a classification of the inter-cluster ICN design-space, qualitatively eliminating architectures which are infeasible. For classification we utilize the I/O behavior of FUs and RFs. Compilation for clustered architectures is a very complex problem. Researchers have used tools which are targeted to one particular interconnection network for exploring a subset of the design-space. As these tools are not retargetable, they are not able to evaluate alternative ICN. We thus build a framework which is retargetable to this huge design-space. To overcome compiler limitations we work directly with the data-flow

of an application.

Since, number of cycles and cycle time itself are both necessary to provide a comprehensive performance estimate, we also evaluate the impact of a particular ICN on clock-period and chip-area. Towards this end, we built synthesizable parameterized VHDL models of processors, supporting the various interconnect architectures. Next, we followed a hierarchical place and route approach for synthesis, place and route and obtained the final clock-period and interconnect area.

1.5 Organization of Thesis

Rest of this thesis is organized as follows: Chapter 2 discusses the evaluation of ILP in media applications which is exploitable by present and future compilers. Chapter 3 presents a review of the previously reported architectures and inter-cluster interconnection networks. It also presents a classification of the this design-space. Chapter 4 gives the design-space exploration methodology for exploring inter-cluster ICNs in clustered VLIW processors. It discusses the necessity to develop a new design-space exploration (DSE) framework for exploration. Chapter 5 presents the results obtained using our DSE framework. Chapter 6 discusses the evaluation of implementation characteristics of clustered architectures. It combines the results obtained in Chapter 5 to conclusively establish that the bus-based communication mechanisms have very poor performance as compared to the point-to-point type communication mechanisms. Finally Chapter 7 summarizes our contributions and also gives directions for future research.

Chapter 2

ILP Studies – A Review

2.1 Introduction

Amount of available instruction level parallelism (ILP) is an important metric which governs the design of multiple-issue processors either of VLIW or SuperScalar type. Although a large number of studies have been carried out, there is wide disparity in reported ILP numbers as they try to measure ILP in different ways. Further, some of these studies measure the ILP totally ignoring architecture behavior which leads to unrealistic results.

In this chapter we present a classification of the different ILP studies and discuss their utility for different purposes. Next, we carry out a study of the amount of hardware constrained achievable ILP in media applications with VLIW architecture as target. The objective of this study is to establish the range of processor configurations, in terms of adequate concurrent instructions, that need to be explored, with reasonable assumptions about future memory hierarchies. Our target processor is a VLIW processor with no software speculation. Effectively we answer the following

important question about VLIW processors: *What is the amount of achievable ILP in media applications which is exploitable by the current compilers and those likely to be available in the near future.*

Most of the current literature (for media processing) reports ILP which is measured using a compiler and simulator. The drawback of such measurements for architectural decisions is that the architecture is not future-proof. The obtained results are limited by how good the compilation system and memory architectures are using the available technology. Compilation for VLIW architectures is a rapidly changing field, with many new techniques being reported each year. Basing architectural decisions on inferior compilation techniques makes the final application performance architecture constrained and not really compilation constrained as is the case now for VLIW architectures. Another consideration is the relevance of the application set. As some of the reported studies are too old, the considered application set is no longer pertinent.

2.2 Classification of ILP Measurement Techniques

Looking at the various reported results on ILP studies [Stefanovic and Martonosi, 2001] [Nicolau and Fisher, 1981] [Lee et al., 2000] [Lam and Wilson, 1992] [Ebcioglu et al., 1999] [Fritts and Wolf, 2000] [Jouppi and Wall, 1989] [Huang and Lilja, 1998] [Chang et al., 1991] [Talla et al., 2000] [Patt et al., 1997] [Liao and Wolfe, 1997] [Smith et al., 1989], a wide variation in the perspective comes to light. While some researchers analyze a given application in isolation to find out how much ILP is theoretically *available* in it, others make an actual measurement to find the ILP actually *achieved* by running the application on a given hardware or its simulation model. The objective in

the former case is to determine the maximum amount of parallelism present in an application that can be extracted by an ideal tool-chain, ignoring all architectural constraints. On the other hand, the latter approach involves the use of a specific compiler to generate code for a specific architecture. Available ILP is an inherent property of the algorithms used in the application, whereas, achieved ILP is an indication of the performance figure obtained taking into account limitations of the software (tool-chain) and constraints imposed by the hardware (architecture).

In between these two extremes, several intermediate scenarios can be conceptualized by taking into account only some of the constraints and ignoring the others. Two interesting cases among these are the following:

a) ILP that can be achieved taking into account only hardware constraints, assuming software to be ideal

b) ILP that can be achieved with unrestricted hardware but taking into account software limitations.

The terms *achievable-H* (hardware constrained achievable ILP) and *achievable-S* (software constrained achievable ILP), are explored further.

Term	**H/W Constraints Considered**	**S/W Limitations Considered**	**Purpose**
Available	No	No	Application algorithm analysis
Achievable-H	Yes	No	Explore architectures
Achievable-S	No	Yes	Study compiler effectiveness
Achieved	Yes	Yes	Study actual performance

Table 2.1: Classification of ILP Measurement Techniques

Table 2.1 summarizes these terms, showing the purpose for which different types of ILP estimation/measurements may be done. In all these cases (including the first) there has to be an underlying definition of primitive operation set. Each of these categories is discussed in detail next with a running example shown in Figure 2.1.

```
#include <stdio.h>
int A[10];
main(){
  A[0] =  0;       /* 1 */
  if( A[0] == 0 ) /* 2 */
    A[1] = A[0];  /* 3 */
}
```

Figure 2.1: Source code for example.c

2.2.1 Available ILP

The category, *Available* is useful for designing new algorithms for an application, as it looks purely at application behavior under an ideal environment. In such cases the only limit on detected ILP is the inherent limits of algorithm along with the quality of the analysis tools. Since the analysis tools tend to be imperfect, this bound can only be predicted and not measured accurately. The predicted values themselves depend on the effectiveness of analysis. One way of measuring this ILP is the *As Soon As Possible* (ASAP) schedule of the obtained trace. Assume a compiler with no optimizations and a pure RISC style ISA for the base processor, with the store instruction needing addresses in registers. The source code of Figure 2.1 translates to assembly as follows: Statement 1 consists of two operations (address computation and store), Statement 2 of two operations and Statement three four operation. A simple analysis environment, would sequentialize the operations of steps 1, 2 and 3

in that order. However, a more sophisticated environment would detect that check in statement 2 is unnecessary and bring out the concurrency between operations of statement 1 and statement 3. In the first case the detected ILP would be 1 (8 operations and 8 schedule steps), however, in the latter, the detected ILP would be 2 (8 operations and 4 schedule steps), ignoring memory latency and functional unit (FU) constraints.

2.2.2 Achievable-H ILP

The category, *Achievable-H* (hardware constrained achievable ILP), is most useful for predicting future architecture performance as well as requirements. Here, the architecture can be taken to varying degree of perfection (ideal caches, perfect branch prediction etc.) which reflects the increasing sophistication of future architectures. However, again as analysis environment tends to be imperfect, this can only be predicted and not measured. For our example, if one assumes perfect branch prediction, one cycle memory access and one integer ALU, the detected ILP would be 1.6 (8 operations and 5 schedule steps). However, assuming a memory latency of 3 cycles, this would be 0.88 (8 operation and 9 schedule steps).

2.2.3 Achievable-S ILP

The third category, *Achievable-S* (software constrained achievable ILP), is most useful for understanding application implementation and quality of the compiler used. This can be carried out (using large number of resources, perfect caches etc.) in frameworks such as IMPACT [Chang et al., 1991] and Trimaran [Trimaran Consortium, 1998]. For the current example, assuming that the compilation environment is not able to

disambiguate global memory accesses properly, it would sequentialize all the operations. Assuming a memory access latency of 1, the detected ILP would be 1 (8 operations and 8 schedule steps), however, if memory access latency is 3 cycles this would fall to 0.53 (8 operations and 15 schedule steps).

2.2.4 Achieved ILP

The category, *Achieved*, gives the current architecture performance, using a compiler and simulator or compiler and real hardware if available. This can be measured accurately as the full tool chain is available. It is useful for obtaining application performance using current technology. However, the results obtained using such techniques are not suitable for predicting future architecture performance or trade-offs, since they represent limitations of the current architectures as well as compilers. For the running example, if we say the processor has only one ALU and a cache miss incurs a penalty of 35 cycles and the processor has write-through caches, the detected ILP would be 0.11 (8 operations and 76 processor cycles) assuming that the branch is predicted correctly. If the branch is mispredicted it would fall to 0.10 (8 operations and 79 cycles) for a 3 cycle misprediction penalty.

2.3 Previous Work

One of the first reported work for quantifying ILP is [Nicolau and Fisher, 1981]. The employed technique assumed presence of an *Oracle* to make perfect predictions, typically branches, register renaming and cache hits. To test limits of ILP, the authors initially assume an infinite-resource machine. Later this infinite-resource assumption is removed to obtain more realistic ILP values. The subsequent experiments

by [Ebcioglu et al., 1999] and [Liao and Wolfe, 1997], refined this technique as well as presented methods by which certain *Oracle* assumptions could be upheld for real architectures. A compiler-simulator based approach was used for measurement in [Jouppi and Wall, 1989] [Huang and Lilja, 1998] [Chang et al., 1991] [Smith et al., 1989] [Patt et al., 1997], with resource assumptions to mimic the then relevant architectures. On similar lines, manual code transformations were applied in [Talla et al., 2000] and performance evaluated on real hardware. Control-flow presents the biggest hurdle to achievable ILP. In [Lee et al., 2000], [Lam and Wilson, 1992] and [Smith et al., 1989], the authors try to bypass such limitations and then measure ILP. The setup is a trace driven simulator with varying degree of control-flow resolution. Additionally, in [Lee et al., 2000], the authors make extensive use of *value prediction* [Nakra et al., 1999]. A totally unconstrained system is assumed in [Stefanovic and Martonosi, 2001] to obtain upper bounds on ILP, however, as stated previously this is not exploitable by any real compiler and architecture.

The underlying application set plays an important role in the detected ILP. Most of the considered applications in [Nicolau and Fisher, 1981] are not relevant now. General applications (SPEC benchmark suite etc.) have been used in [Ebcioglu et al., 1999] [Liao and Wolfe, 1997] [Lam and Wilson, 1992] [Smith et al., 1989] [Lee et al., 2000] [Stefanovic and Martonosi, 2001]. Media applications (MediaBench, DSP-Stone etc.) have been considered in [Fritts and Wolf, 2000, Liao and Wolfe, 1997]. [Liao and Wolfe, 1997] only presents results for some video applications and miss out other applications such as, JPEG, ADPCM etc.

Table 2.2, summarizes this classification of the previous work. Here, the studies which report multiple results, have been shown under all applicable categories. The column ILP, shows the range of values which have been reported in these studies. It

	General Purpose Applications	ILP	Media Applications	ILP
Available	[Stefanovic and Martonosi, 2001] [Nicolau and Fisher, 1981] [Lam and Wilson, 1992] [Ebcioglu et al., 1999]	500+		
Achievable-H	[Nicolau and Fisher, 1981] [Lee et al., 2000] [Lam and Wilson, 1992] [Ebcioglu et al., 1999]	20-40	***Our Measurements***	
Achievable-S			[Fritts and Wolf, 2000] [Patt et al., 1997]	3-7
Achieved	[Jouppi and Wall, 1989] [Huang and Lilja, 1998] [Chang et al., 1991] [Smith et al., 1989]	1.5-2.5	[Talla et al., 2000] [Patt et al., 1997]	1.5-2.5

Table 2.2: Classification of Previous ILP Studies

can be clearly seen from this table that no previous work is available which tries to measure hardware constrained achievable ILP (*Achievable-H)* in media applications.

2.4 Evaluation Methodology

Our goal is to measure *Achievable-H* ILP for media applications. Towards this end, the analysis environment is built using a state of the art compiler-simulator, Trimaran [Trimaran Consortium, 1998], with all the supported ILP enhancing optimizations in-place. The hardware constraints are brought in using a DFG generation and instruction scheduling phase. Working at the generated trace level has the advantage that all the run time details are available. For example control-flow information, exact addresses for perfect memory disambiguation etc. To simplify scheduling, we assume that each instruction takes at most one cycle. Also, the architecture has perfect caches and the processor register file has sufficient number of ports to support the high degree of concurrency. The ports assumptions is made so as to ensure that

there is no concurrency loss due to insufficient number of ports. Moreover, our base Instruction Set Architecture (ISA) is HPL-PD [Kathail et al., 2000]. This methodology, is an adaptation of the technique reported in [Lee et al., 2000] and also the constrained resource Oracle case [Nicolau and Fisher, 1981, Ebcioglu et al., 1999, Liao and Wolfe, 1997]. Our methodology also takes care of control-flow resolution as discussed in [Lam and Wilson, 1992]. However, unlike them, we do not present results for several machine models.

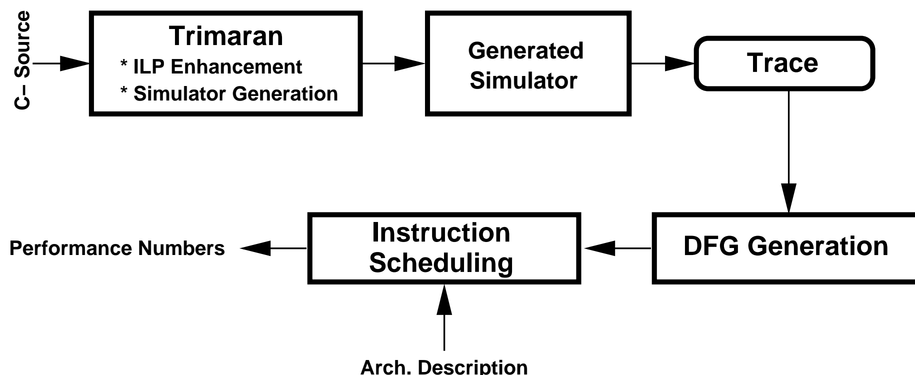


Figure 2.2: Evaluation Framework for ILP in Media Applications

Figure 2.2, shows the evaluation framework. The C-source program is passed through Trimaran compilation system [Trimaran Consortium, 1998]. This generates an execution driven simulator which produces a trace of the program execution. This trace is consumed by the Data Flow Graph (DFG) generation phase which builds a DFG out of this trace. This generated DFG is scheduled and bound to the FUs by the instruction scheduling phase. The instruction scheduling phase totally discards any previous instruction scheduling and binding information and reschedules and re-binds the instructions to FUs.

2.4.1 Trace Generation

The Trimaran system [Trimaran Consortium, 1998] has been augmented so that the generated simulator produces a trace of the program execution. To obtain unconstrained ILP initially, the machine model used for Trimaran is an extremely flexible one. It has 256 GPR and 1024 Predicate registers. The number of predicates is large because each predicate register is only one bit wide. An ILP compiler with predication typically needs these in a far larger number than GPRs. Also, the machine model has 32 Float, Integer, Memory and Branch Units each to remove any ILP losses due to insufficient resources. Trimaran performs a number of ILP enhancing transformations, out of which the loop unrolling transformation is the most important one. This reduces false register dependences introduced due to register reuse. These dependencies are introduced as we do not have any value prediction in the system [Nakra et al., 1999].

2.4.2 DFG Generation

A DFG is extracted for each of the functions from the trace generated using the Trimaran system. The DFG generation phase picks up each instruction in sequence and searches backwards from that instruction. In this process it finds out the first occurrence of the source registers/address of this instruction in the destination field of a previous instruction. If the source register is found, a data-flow edge is introduced. However, if the addresses are found to match, then the communication has happened through memory and a false edge is introduced. This false edge doesn't denote data dependency, rather, it denotes a constraint for the scheduler that this instruction should not be scheduled before the instruction which stores this value in memory

has been scheduled.

2.4.3 Instruction Scheduling

The last phase is the instruction scheduling phase. This phase uses the widely used resource constrained list scheduling algorithm (for data-flow graphs) with distance-to-sink as the priority function. The input architecture, specifying the number of FUs of various types forms the resource constraint. Since, the type of architecture which we are targeting is a monolithic VLIW architecture, no other information is needed at this level of abstraction.

2.5 Experimental Results and Observations

The main criteria governing the choice of benchmarks was relevance. The primary benchmark sources were MediaBench I [Lee et al., 1997] and proposed Mediabench II [Fritts and Mangione-Smith, 2002]. A second smaller set of benchmarks was chosen from various sources, primarily, DSPStone [Zivojinovic et al., 1994] and Trimaran [Trimaran Consortium, 1998]. The smaller set of benchmarks are complete programs i.e. compilable and executable, whereas for the remaining benchmarks, a set of most time consuming functions have been chosen. We have tried to capture around 85% execution time for each of the larger benchmarks, though in case of MPEG Decoder this has not been possible due to unavailability of a small *critical set*.

2.5.1 Achievable-H ILP in Benchmarks

The results are presented in Figures 2.3, 2.4 and 2.5. Looking at these figures, it is clear that the achievable parallelism in media applications is very high. It is also

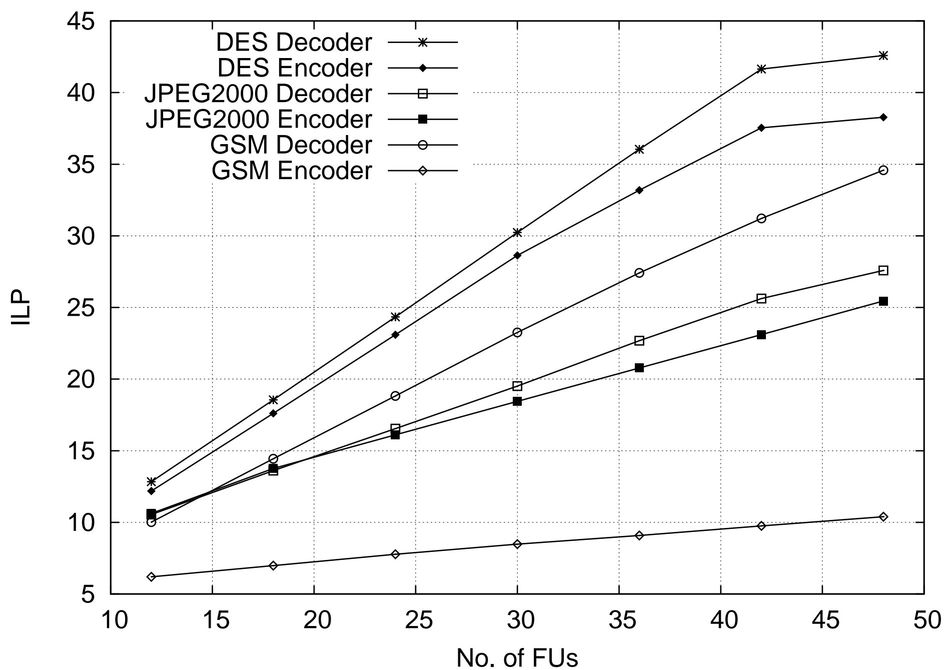


Figure 2.3: *Achievable-H* ILP in MediaBench-II Applications

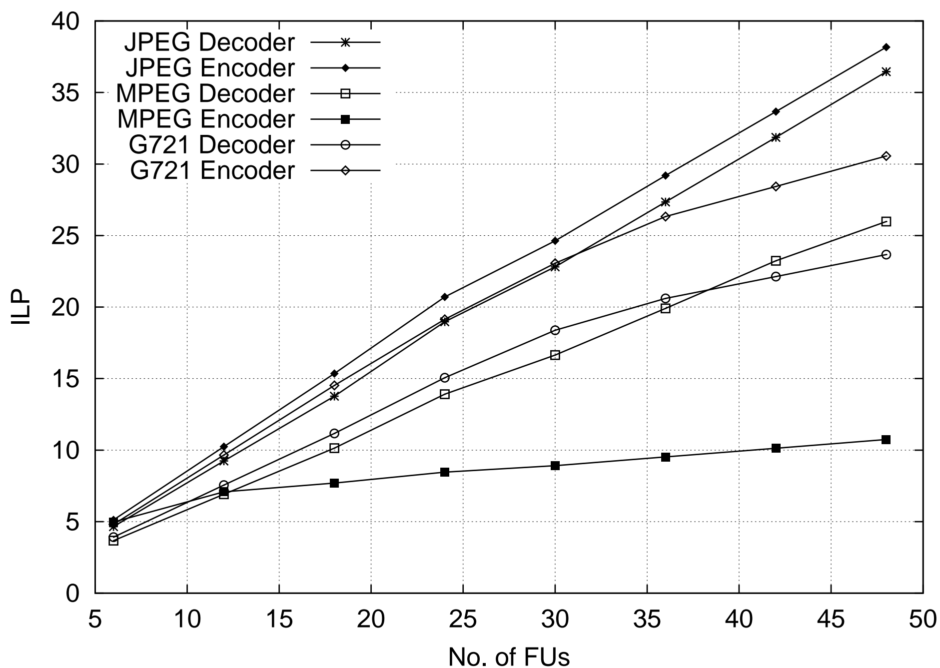


Figure 2.4: *Achievable-H* ILP in MediaBench Applications

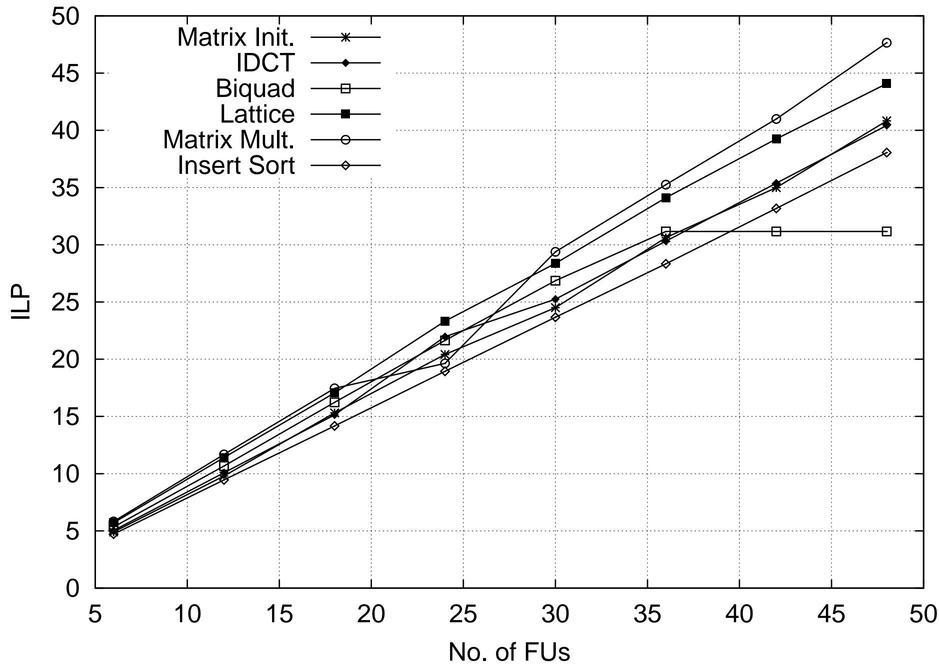


Figure 2.5: *Achievable-H* ILP in DSPStone Kernels

clear that the amount of achievable ILP increases with the increase in number of functional units. In few cases, most notably MPEG2 Encoder and GSM Encoder, the ILP is quite low as compared to other applications. This is due to the fact that functions which consumes more than 85% of the execution time have very little ILP. Hence, higher ILP for these applications can only be achieved if the underlying algorithm is changed. The ILP in case of *Biquad*, levels off at around 36 FUs. The implication is that the speedup in Biquad is sustainable only for upto 36 FUs, and increasing the FUs beyond that doesn't lead to any further gains.

2.6 Exploiting the Achievable ILP

Most of the available ILP is compiler exploitable as we have carried out only function level DFG generation and scheduling for measurement. Moreover, the typical media

kernels are data-flow graphs, which when represented in the form of loops present branch conditions. However, in most of the media applications the loop bounds are exactly known. Thus, unrolling loops to a known depth (also fully unrolling) doesn't pose any problem. Also, the data-parallelism which is typically found in such applications is often in the form of an external loop, which can be exploited either manually or by using some high level code transformations. The biggest challenge though is to efficiently schedule instructions beyond control-block boundaries (if-else). Since, we have used instruction traces, these instructions do appear as free and hence contribute to the ILP. Aggressive Hyperblock [Mahlke et al., 1992] formation is not a solution, as a large Hyberblock doesn't really translate to faster code due to dynamic nullification of unused instructions. This indeed is an open problem, given the massive amounts of ILP which could be exploited.

2.7 Summary

In this chapter we have presented a classification of the different ILP measurement techniques as well as discussed their utility. While detecting ILP in media applications, we have taken a different approach. We have bypassed typical compiler limitations in dealing with pointer aliasing by working on the instruction trace. Further we have made use of existing ILP enhancing compiler optimizations by obtaining the trace from a state of the art VLIW compiler infrastructure, namely, Trimaran.

Results clearly show that for a statically scheduled VLIW processor the average ILP in media applications is 20 or higher. This builds a strong motivation for exploring very high issue rate VLIW processors. To support such high issue rate with a large number of FUs clustered VLIW is a suitable architectural option. Though it

does bring in additional complexity in instruction scheduling.

Chapter 3

Inter-cluster Communication – A Review

Chapter 2 conclusively established that there is large amount of ILP present in media applications which is potentially exploitable by VLIW processors. A large amount of ILP naturally justifies large number of FUs for multiple-issue processors, which in turn due to the prohibitively expensive register file [Rixner et al., 2000] justify clustering. Clustering is nothing new, as a large number of architectures, both commercial as well as research, have had clustered architectures. These include Siroyan[Siroyan, 2002], TiC6x [Texas Instruments, 2000], Sun's MAJC, Equator's MAP-CA and TransMogrifier [Lewis et al., 1997]. A large variety of inter-cluster interconnection mechanisms is seen in each of these processors. However, the tools which have been developed for these architectures are specific to the particular architecture and are not retargetable. Also, what is missing is a concrete classification of interconnection design-space as well as a formal study of the impact of different interconnections on performance.

3.1 Previous Work

3.1.1 Previously Reported Architectures

Inter-cluster ICN in which only the RF of each cluster is connected to the RF of other cluster using buses have been most commonly reported in literature [Zalamea et al., 2001] [Ozer et al., 1998] [Sanchez et al., 2002] [Faraboschi et al., 2000] [Cruz et al., 2000] [Codina et al., 2001] [Smits, 2001] [Fisher et al., 1996] [Song, 1998]. While some researchers have used only buses for inter-cluster connectivity [Zalamea et al., 2001] [Ozer et al., 1998] [Smits, 2001] [Fisher et al., 1996], others have made some minor modifications to this mechanism. In one such approach the RF-to-RF path exists, however, the L1 data-cache is also clustered and brought inside each of the clusters (Enric at. el. [Sanchez et al., 2002]). The L2 cache maintains consistency using some cache coherency protocol. The Lx, technology platform from HP (Paolo at. el. [Faraboschi et al., 2000]), uses many such buses for transfers. Since, Lx is a family of architectures the number of such buses is not fixed but may be increased or decreased as per application requirements. In fact Fisher proposes in [Fisher et al., 1996] that such a customization is inevitable. Another variation, is that reported by [Cruz et al., 2000]. In their architecture, the RF is organized at different levels, the lowest level (those communicating with FUs) RFs have larger number of ports but not many registers, while those at the higher level have fewer number of ports but more registers. Values are moved from lower to the higher level and are cached at the higher levels. [Codina et al., 2001] report a RF-to-RF architecture, which uses separate read and write FUs for the buses. Additionally the RF has one port on the L1 cache. Finally the future Intel Itanium processors will be clustered and will employ the RF-to-RF mechanisms [Song, 1998].

The other commonly found inter-cluster ICN is the one in which FUs from cluster can write directly to RF of other cluster [Lewis et al., 1997] [Aditya et al., 1998] [Texas Instruments, 2000]. In a variation of this the FUs may write to the bypass network of the destination cluster. Such a scheme has obvious advantages as compared to the bus-based mechanism. Since buses are global there is severe contention for acquiring them. Long running wires lead to a decrease in the clock-period of the target implementation. Moreover the buses incur a data transfer penalty of one cycle, whenever a value needs to be moved from one cluster to another. However, buses are easier for compiler scheduling as they present a fully connected architecture. Architectures with direct write paths from one cluster to another circumvent these problems. However, they do lose out, when either values need to be moved to multiple clusters (multiple consumers) or the values need to be transfered to distant clusters.

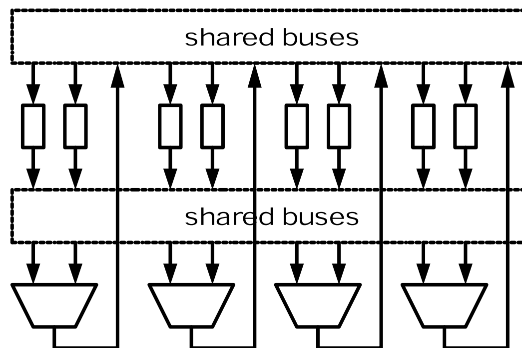


Figure 3.1: Stanford Imagine Architecture

Figure 3.1, shows the inter-cluster ICN used in the Stanford Imagine media processor (Mattson at. el. [Mattson et al., 2001]). This is an interesting addition to the already discussed architectures. In the Imagine architecture, all the FUs are connected directly to all the RFs using shared buses. This provides maximum flexibility

for compiler scheduling as well as inter-cluster communication. Since, all the FUs can communicate with all the RFs directly there is very little need for incurring the overhead of copying values from one RF to another. Though this may become a necessity in cases where the register pressure on one RF becomes prohibitively high. There are some disadvantages of using such an architecture. Firstly long running wires will ofcourse reduce the achieved target frequency of the architecture. Secondly, the compiler complexity is high because now the compiler must also allocate the multiple independent communication buses amongst FUs.

At the far end of the spectrum are the inter-cluster ICNs employing cross-bars amongst clusters [Bhargava and John, 2003] [Fritts et al., 1999]. Additionally [Bhargava and John, 2003] predominantly use the data-memory for communication amongst clusters, however there is provision for an inter-cluster data bypass. Similarly [Fritts et al., 1999], use cross-bar for communication amongst clusters, however they constrain the clusters to share I/Os. The justification for sharing I/Os is not very clear.

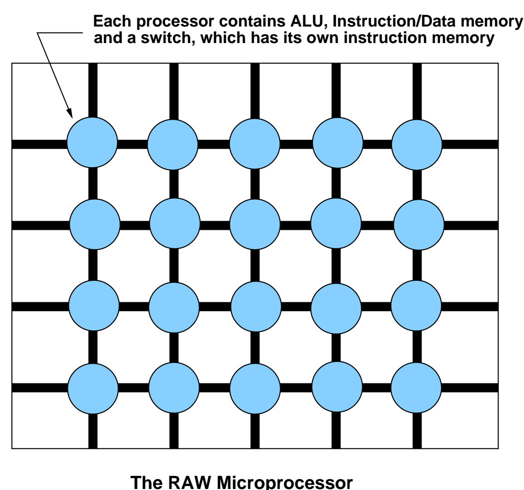


Figure 3.2: MIT RAW Architectures

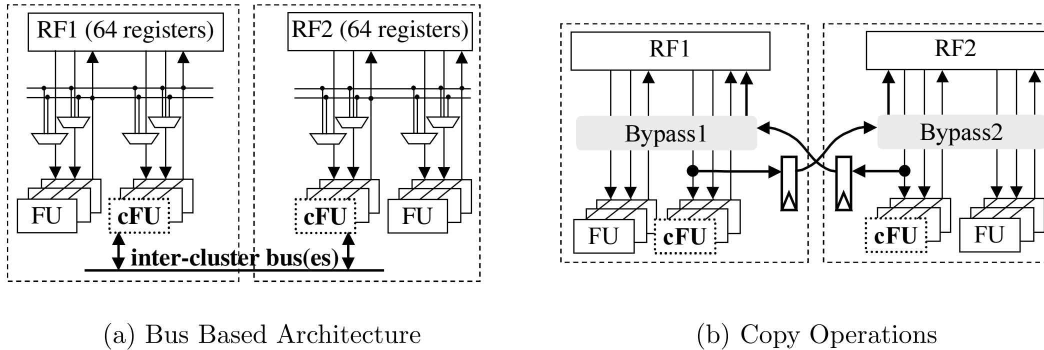
Figure 3.2, shows the MIT RAW architecture [Lee et al., 1998]. The RAW architecture is organized in the form of a rectangular array of tiles. Each tile contains few processing elements along with a communication processor (switch). Each tile can communicate with the adjacent four (max) tiles. Data can be further moved amongst tiles by using the communication processor. RAW is not a VLIW architecture, as it has multiple flows of control, however, it is interesting to note the flexibility provided in this architecture. Also, RAW is similar to VLIW in the sense that it is a statically scheduled architecture. On somewhat similar lines the Berkeley IRAM architecture [Kozyrakis et al., 1997] gets rid of traditional caches and builds a chip out of DRAMs, wherein each DRAM has vector processing elements attached to them.

As is quite evident, the design-space for these ICNs is quite wide and researchers have used a wide variety of ICNs without a quantitative justification. Future media processors will have a very wide issue-width and also they will have a clustered architecture [Conte et al., 1997] [Doug Burger and James R. Goodman, 2004], to efficiently explore this design-space it is necessary to first present a proper classification.

3.1.2 Previous Classification of ICNs

[Terechko et al., 2003], were the first to explore the performance tradeoffs in different inter-cluster ICNs. Figures 3.3, 3.4 and 3.5, show the five different inter-cluster ICNs which they have explored. We use their nomenclature for further discussion on each of these.

Figures 3.3(a) shows an architecture which uses inter-cluster buses for communication. This architecture uses dedicated communication FUs for transferring data. If



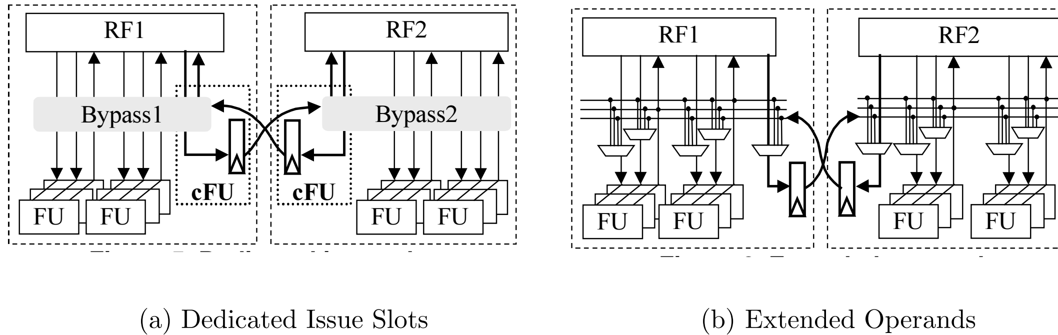
(a) Bus Based Architecture

(b) Copy Operations

Figure 3.3: Architectures from [Terechko et al., 2003]

a value from one cluster needs to be moved to another, an explicit copy instruction is issue. This additional instruction incurs one compiler cycle penalty during the production and consumption of each such value. It needs to be noted that the buses are shared across all clusters, which may lead to severe contention on them. Figure 3.3(b), shows an architecture which again has dedicated FUs for communication. The instruction may be issued in regular VLIW slots and copies the value directly to the RF of the destination cluster. This value may be fed additionally to the bypass network of the destination cluster to avoid an additional one cycle penalty. However, in this case the communication happens using dedicated *point-to-point* links between the clusters, which to some extent avoids resource contention. It needs to be noted that the communication FUs have a direct path to all the other clusters.

Figure 3.4(a), shows an architecture which is similar to the one shown in Figure 3.3(b), the difference is that in this case the copy instruction has to be issued in a dedicated issue slot. Since, the communication FUs are connected directly to the RF it leads to an increase in the number of ports on the RF. The number of communication FUs can be increased or decreased depending on the application requirements.



(a) Dedicated Issue Slots (b) Extended Operands

Figure 3.4: Architectures From [Terechko et al., 2003]

Figure 3.4(b), shows the extended operand architecture. In this architecture, the operands are extended with the *cluster id* of the source cluster. It may lead to a decrease in the register pressure as the produced value may directly be consumed, whereas in the case of architectures with copy operations, this value would first need to be stored and then copied over. A downside is that each of the operands would end up requiring more number of bits as compared to the previous architectures. On similar note, the results can be directly sent to the destination cluster by extending them with the destination clusters *cluster id*.

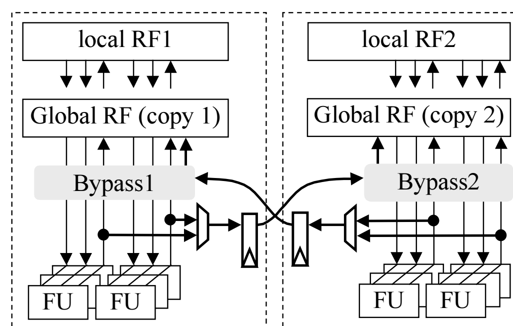


Figure 3.5: Broadcast Architecture From [Terechko et al., 2003]

Figure 3.5, shows the architecture which can broadcast results to other clusters. In this architectures, the address space for registers are shared. It is the hardware's

job to keep the two RFs synchronized. This effectively makes it compiler transparent in the sense that the compiler need not be aware of clustering.

[Terechko et al., 2003] presented a first attempt at classification. However, as is quite evident their work has some limitations. Firstly, they have not classified the domain, rather they have given five representative architectures which have been previously used. Also, there is mix of compiler and architectural issues in their representative architectures. For example the segregation of Copy Operation and Dedicated Issue Slot architectures does not vary the way in which these two architectures are interconnected, still they have reported it as a separate category. In the next section we use the I/O behavior of the FUs to present a coarse-grained classification of the entire design-space.

3.2 Our Interconnection Design Space

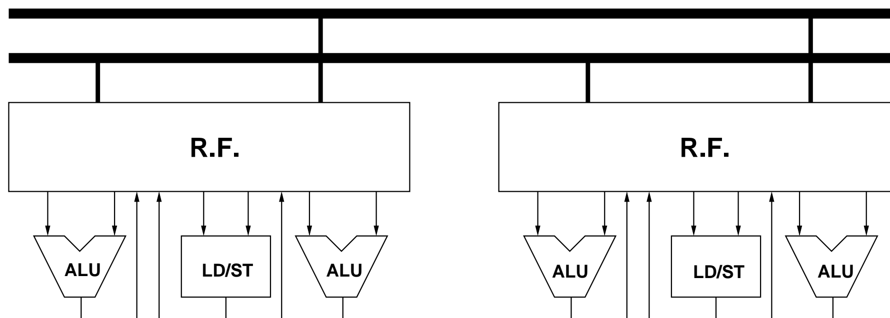
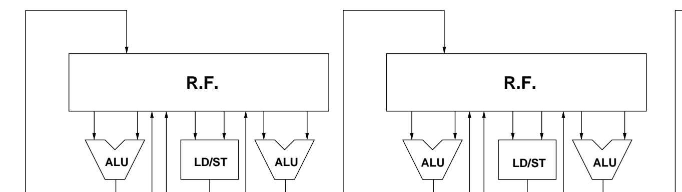


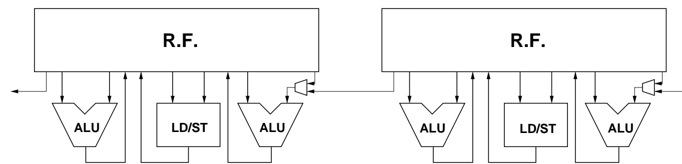
Figure 3.6: RF-to-RF ($\delta = 1$)

Clustered VLIW processors can be classified on the basis of their inter-cluster communication structures. At the top level we can divide them into two sub-categories: a) Those supporting inter-cluster RF-to-RF copy operations and b) Those supporting direct inter-cluster communication between FUs and RFs. There

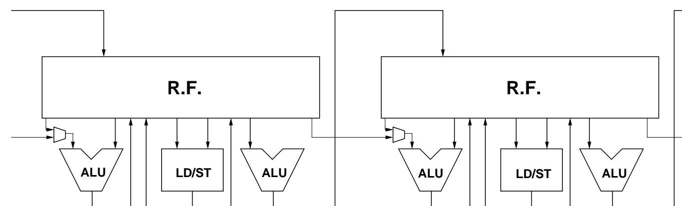
is very little variety in the architectures supporting RF-to-RF copy operations. At most these can be classified on the basis of the interconnect mechanism which they use for supporting these transfers. The examples of such architectures are: Lx [Faraboschi et al., 2000], NOVA [Jacome and de Veciana, 2000], [Sanchez and Gonzalez, 2000], IA-64 [Song, 1998]. An example of RF-to-RF architecture is shown in Figure 3.6.



(a) Write Across-1



(b) Read Across-1

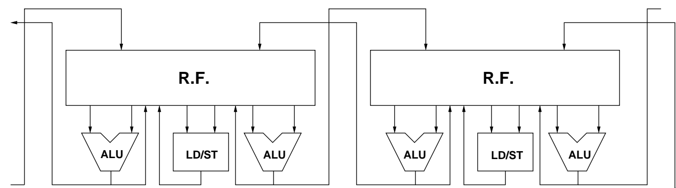


(c) Write/Read Across-1

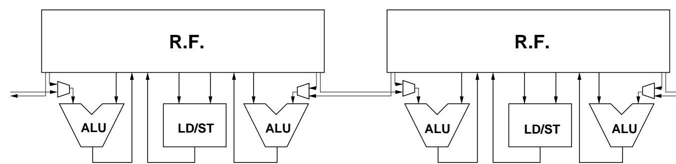
Figure 3.7: Architectures with ($\delta = n_clusters$)

We use the RF→FU (read) and FU→RF (write) communication mechanisms to classify direct inter-cluster communication architectures. The reads and writes can be from either the same cluster or across clusters. The communication can be

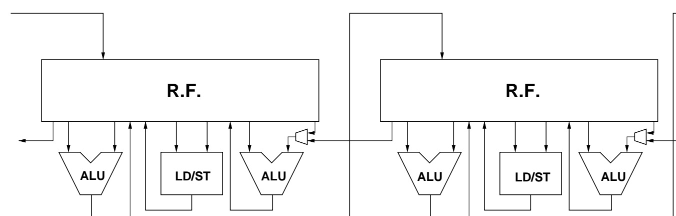
either using a point-to-point network, a bus or a buffered point-to-point connection. In a buffered point-to-point connection, an additional buffer is introduced in the interconnect, to contain an increase in clock-period. An underlying assumption is that FUs always have one path to their RF (both read and write) which they may or may not use. A few examples of these architectures are shown in Figures 3.7 and 3.8. The architecture shown in Figure 3.8(a) has been used by Siroyan [Siroyan, 2002], Transmogrifier [Lewis et al., 1997] etc.



(a) Write Across-2



(b) Read Across-2



(c) Write/Read Across-2

Figure 3.8: Architectures with ($\delta = n_clusters/2$)

3.3 Our Classification of Inter-cluster ICNs

Our complete design space of clustered architectures is shown in Table 3.1. The Columns 1 and 2 marked as *Reads* and *Writes* denote whether the reads and writes are across (A) clusters or within the same (S) cluster. Columns 3 and 4 marked as *RF→FU* and *FU→RF*, specify the interconnect type from register file to FU and from FU to register file respectively. Here, *PP* denotes *Point-to-Point* and *PPB* denotes *Point-to-Point Buffered*. This table also shows in Column 5, the commercial or research architectures which have been explored in this complete design space. For example the TiC6x is an architecture, which reads across clusters and writes to the same cluster; it uses buses for reading from RFs and point-to-point connections for writing back results to RFs.

We would like to contrast here our classification with what has been presented in [Terechko et al., 2003] and also discussed in Section 3.1.2. They have only considered five different communication mechanisms without a generic classification. The *bus-based* and *communication FU* based interconnects which they have considered are part of the RF-to-RF type communication domain in our classification. The *extended results* type architecture is a *write across* architecture in our classification and *extended operands* is basically a *read across* type of architecture as per our classification. However, here again, they have considered only one type of interconnect, point-to-point, whereas others such as point-to-point buffered or buses are also possible. These have been shown in Table 3.1.

It can be clearly seen from Table 3.1 that a large range of architectures have not been explored. For each of these architectures an important metric is the maximum hop distance between any two clusters (δ). Hop distance, is the shortest number

Reads	Writes	RF→FU	FU→RF	Available Architectures
S	S	PP	PP	TriMedia, FR-V, MAP-CA, MAJC
A	S	PP	PP	
A	S	Bus	PP	Ti C6x
A	S	PPB	PP	
S	A	PP	PP	Transmogrifier, Siroyan, A|RT
S	A	PP	Bus	
S	A	PP	PPB	
A	A	PP	PP	
A	A	PP	Bus	
A	A	PP	PPB	
A	A	Bus	PP	
A	A	Bus	Bus	Stanford Imagine
A	A	Bus	PPB	
A	A	PPB	PP	
A	A	PPB	Bus	
A	A	PPB	PPB	

Table 3.1: Overall Design-Space for Direct Communication Architectures

of hops using direct interconnects between any two clusters. For example in case of architecture in Figure 3.7(a) $\delta = 8$ and for architecture in Figure 3.8(b), $\delta = 4$ assuming an 8-cluster configuration. δ is not an independent parameter, it can be calculated from the architecture type and number of clusters ($n_clusters$).

3.4 Summary and Conclusions

This chapter presented a review of the existing inter-cluster interconnects in clustered VLIW architectures. Table 3.1, clearly established that a larger part of the inter-cluster interconnect design-space is currently unexplored. To be able to quantify the

merits and demerits of individual interconnects, suitable design-space exploration must be performed. Next chapter discusses the existing design-space exploration methods and also presents a new design-space exploration methodology.

Chapter 4

Design Space Exploration Framework

Design-space exploration for VLIW processors is typically carried out using a compiler-simulator pair [Rau and Schlansker, 2001] [Jacome et al., 2000]. A retargetable compiler such as [Trimaran Consortium, 1998] is used to compile the code for the target architecture. This is in turn simulated on a simulator for the target architecture. For VLIW processors, simulation of generated code is straightforward while compilation is complicated. Typical exploration techniques, such as those presented in [Rau and Schlansker, 2001] and [Jacome et al., 2000] assume a baseline ISA which eases the task of compilation. Design-space exploration is then restricted to the number and types of FUs, processor data-width etc., but not the inter-cluster interconnection network. The next section presents a review of the existing code generation techniques pinpointing the specific inter-cluster interconnect architectures to which they can be applied.

4.1 Existing Code Generation Techniques for Clustered VLIWs

Code generation for clustered VLIW processors is a complicated process. The typical process of **a)** Register allocation and **b)** Code scheduling has to be augmented with cluster assignment of operations. [Jacome et al., 2000], report a technique which works on DFGs for RF-to-RF architectures. The DFGs are divided into a collection of *vertical* and *horizontal* aggregates. A derived value *load* for each cluster is used to decide whether the next operation is scheduled onto this cluster or not. The aggregates are scheduled as a whole and are not further subdivided. Since, the algorithm is fast it can be used for both design-space exploration and code generation. [Sanchez and Gonzalez, 2000], report another technique which works on DFGs for RF-to-RF architectures. They employ a greedy algorithm which tries to minimize communication amongst clusters. The algorithm performs simultaneous cluster assignment of operations along with scheduling. Register allocation in their approach is trivial with generated values going to the cluster in which operation generating this value has been scheduled.

[Desoli, 1998], proposed another approach for code generation. He terms his algorithm as Partial Component Clustering (PCC). The algorithm works for DFGs and RF-to-RF architectures. Problem reduction is achieved by identifying small portions of Directed Acyclic Graph (Sub-DAG) which are in turn scheduled and bound to a cluster as a whole. The algorithm works well for applications when the Sub-DAGs are balanced in terms of number of operations and critical path lengths. [Kailas et al., 2001], propose a framework for code generation, CARS. Therein cluster assignment, register allocation and instruction scheduling are done in a single

step to avoid back-tracking (rescheduling an already scheduled instruction) and thus leading to better overall schedules. The algorithm works on any region i.e. Basic-Blocks, HyperBlocks, SuperBlocks, Treegions but only for RF-to-RF type architectures. Register allocation is performed on the fly to incorporate effects of generated spill code.

[Banerjia et al., 1997] propose a global scheduling algorithm for RF-to-RF type architectures. This works on a tree of basic-blocks termed as Treegions [Banerjia et al., 1997]. The basic-blocks included in the Treegion are sorted as per the execution frequency. Then list scheduling is done from the root of the Treegion, assuming speculative execution similar to Superblock scheduling [Hwu et al., 1993]. This process is repeated till all the basic-blocks have been scheduled. [Terechko et al., 2003] present another scheduling algorithm based on Treegion and which can be applied to fully-connected architectures, i.e. architectures wherein each cluster has a direct write/read path to any other cluster.

[Leupers, 2000], proposed a simulated annealing based approach. The algorithm works for DFGs and for Write Across type architectures (TiC6201). Operation partitioning is based on simple simulated annealing with cost as schedule length of the DFG accounting for the copy operations needed to move data between clusters. The reported execution speed of algorithm is good, with a 100 node DFG being partitioned in 10 CPU seconds on a SUN Ultra-1. However, this could become prohibitively expensive for large DFGs. The MIT RAW architecture employs multiple threads of control unlike a VLIW processor and hence its scheduling technique is not discussed here.

Table 4.1 compares the various code generation algorithms. What has been shown as *Our Approach* in this table is our design-space exploration methodology as dis-

Name	**Region**	**Architecture(s)**	**Run-time**
[Jacome et al., 2000]	DFG	RF-to-RF	May be high
[Sanchez and Gonzalez, 2000]	DFG	RF-to-RF	Low
PCC	DFG	RF-to-RF	May be high
CARS	All	RF-to-RF	Low
TTS	Treegion	RF-to-RF	Low
[Terechko et al., 2003]	Treegion	RF-to-RF and other fully-connected architectures	Low
[Leupers, 2000]	DFG	Write Across	May be high
Our Approach	DFG	All	High

Table 4.1: Code Scheduling for Clustered VLIWs

cussed in Chapter 4. As is quite clear from this table, the existing techniques are not applicable across architectures. Most of the techniques (except [Leupers, 2000]) are geared towards the simple to schedule, RF-to-RF type architecture. To explore this design-space, it is thus necessary to develop a new exploration framework.

4.2 Design Space Exploration Methodology

A first study of various inter-cluster communication mechanisms has been presented in [Terechko et al., 2003]. However, they have considered only five different communication mechanisms and also the amount of ILP in their benchmarks is quite low (maximum is around 4). While our work focuses more on the inter-cluster interconnects, their work focused more on the instruction issue mechanisms. As they only have an ILP of four, they have not explored beyond four clusters. It was our hypothesis and which has been validated by the results, that restricting to four clusters would not bring out the effects of different interconnection mechanisms. Another limitation of [Terechko et al., 2003] is that they have used compiler scheduling for result generation. Thus amount of detected parallelism is directly proportional to

the size of the block being formed by the VLIW compiler for scheduling. While they do work on a specialized block called Treegion [Banerjia et al., 1997], which is larger than the Hyper Blocks [Mahlke et al., 1992] or Super Blocks [Hwu et al., 1993] typically formed by a VLIW compiler, the extracted parallelism is still quite small. Based on these observations, we work directly with the data-flow of an application, as described in [Lee et al., 2000]. An instruction trace is obtained from a VLIW compiler, Trimaran [Trimaran Consortium, 1998], and then the data-flow graph (DFG) is generated from this. This DFG is scheduled and bound to the various FUs to obtain the final performance numbers. Using such a methodology allows us to bypass compiler limitations.

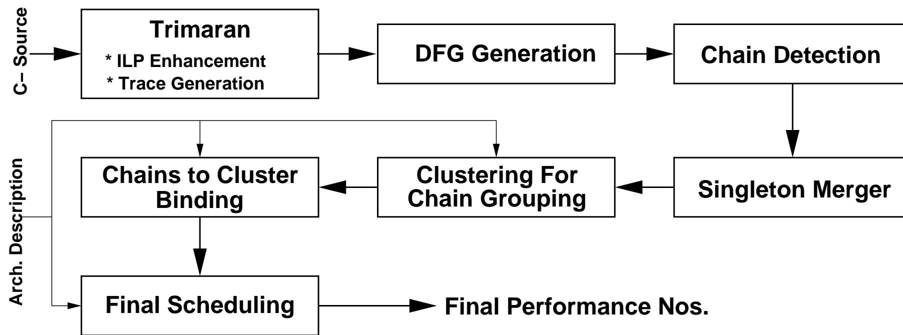


Figure 4.1: DSE Framework

Figure 4.1 shows the overall design space exploration methodology. The Trimaran system is used to obtain an instruction trace of the whole application from which a DFG is extracted for each of the functions. Trimaran also performs a number of ILP enhancing transformations. This trace is fed to the DFG generating phase, which generates a DFG out of this instruction trace. The chain detection phase finds out long sequences of operations in the generated DFG. The clustering phase, which comes next, forms groups of chains iteratively, till the number of groups is reduced to the number of clusters in the architecture. The bind-

ing phase, binds these groups of chains to the clusters. It is well known that optimal results are obtained, when all the subproblems i.e. operation to cluster and FU assignment, register allocation and scheduling are done simultaneously [Kailas et al., 2001, Ozer et al., 1998]. However, this makes the problem intractable for large graphs. We thus divide the problem as follows: First operation to cluster binding is done followed by operation to FU within a cluster. Since, during clustering, the partial schedules are calculated (explained in detail later), the typical phase coupling problem [Kailas et al., 2001, Ozer et al., 1998], is contained to a large extent, while still keeping the overall problem size manageable. Lastly, a scheduling phase schedules the operations into appropriate steps. More details of each of these phases follows.

4.2.1 DFG Generation

The DFG generation is carried out as described in Chapter 2.

4.2.2 Chain Detection

Our main emphasis is on minimizing communication amongst various clusters. The long sequences of operations denote compatible resource usages, as well as production and consumption of values. This makes them ideal candidates for merger into a single cluster [Jacome et al., 2000]. Since the operations are sequential anyway, no concurrency is lost due to this merger. In the chains detection phase, long sequences of operations are found in this DFG. The idea is to bind these chains to one cluster. Since, the DFG is acyclic, standard algorithms can be applied to find the long sequence of operations (chains). It needs to be noted that the detected chains are not

unique and will vary from one chain detection algorithm to another.

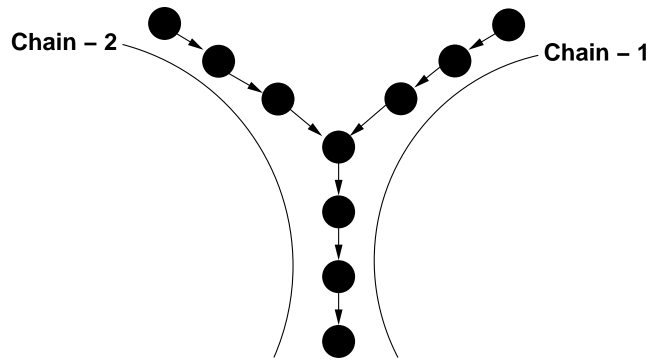


Figure 4.2: Longest Chain Detection

Figure 4.2, shows how different algorithms might end up detecting different longest chains. The two chains shown in this figure Chain-1 and Chain-2, have the same length. Had this not been the case, the algorithm would have picked up the longest of the two. So, whether Chain-1 is detected or Chain-2, the remaining operations will lead to a smaller chain with same length in both the cases. The longest chain i.e. is the critical path represents a lower bound on the schedule length of the DFG. Also, since we are picking up the chains in descending order of their length, there is no concurrency loss. This is due to the fact that the chain which is detected next again has a producer-consumer relationship between operations. Hence these operations would need to be serialized anyways. Thus it is evident that neither chain detection nor the order in which these chains are detected is constraining the ILP in any way and effectively represent a lower bound on the schedule length which can be obtained for the DFG, which in turn represent an upper bound on the ILP.

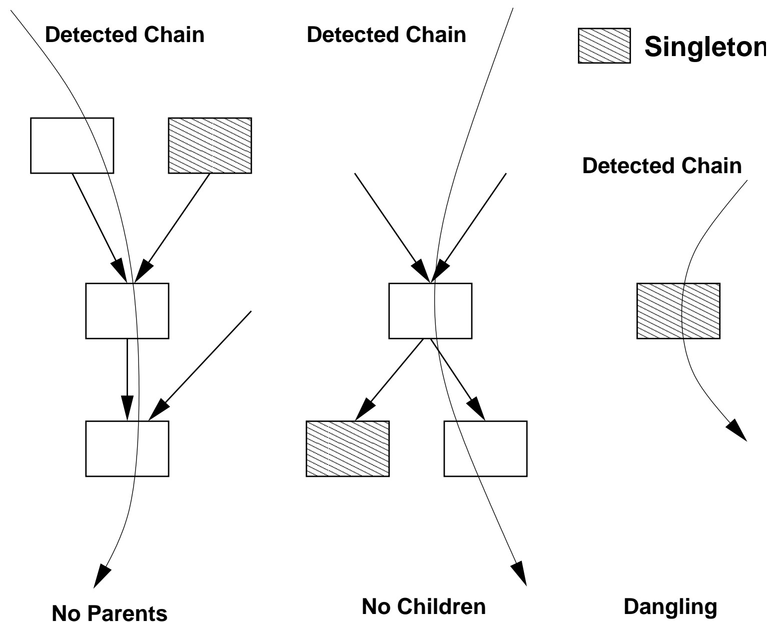


Figure 4.3: Reasons for Singleton Generation

4.2.3 Singleton Merger

The chain detection phase returns a large number of chains, including a significant number which have only one element (singleton). After observing the large number of singletons, we specifically introduced a singleton merging phase. The singletons can be generated due to a number of reasons. The three most prominent of these are shown in Figure 4.3:

a). *No parents:* One of the source nodes is left alone as a consequence of merger of its only child.

b). *No children:* The destination node is left alone as a consequence of merger of both its parents in separate chains.

c). *Dangling:* The node doesn't have any parents or children and as a consequence is dangling. Dangling nodes are basically constant predicates which were gen-

erated by VLIW compiler, but never got used during run-time.

In the singleton merger phase, the singletons which do not have any source or destination are distributed in $n_clusters$ number of chains (groups). The idea is that these would not constrain the scheduler in any way, as they do not have any source or destination dependency. Nodes, which have no destinations (sources) are merged with the shortest chain of their sources (destinations).

4.2.4 Clustering

The next phase is the clustering phase (Algorithm 1). Here the number of chains is reduced to the number of clusters, by grouping selected chains together. The idea here is to reduce the inter-cluster communication between various groups. However, the closeness between these groups is architecture dependent. This can be better understood by examining architectures shown in Figures 3.7(a) and 3.8(c). While in the former, each cluster is "write close" to the adjacent cluster, in the latter, any two adjacent clusters are both "read close" as well as "write close". During the clustering process, we assume the best connectivity between clusters. So, if a value needs to be moved from cluster A to cluster B, and the architecture only supports neighbor connectivity, we assume that cluster B is a neighbor of cluster A. Though after final binding this may not be true. Thus, this schedule length effectively represents a lower bound on the actual schedule length.

The loop (steps 3 to 15) reduces the number of chains to $n_clusters$ by pairwise merger of chains. The chains are examined pairwise for their affinity (steps 4 to 11) and a lower triangular matrix C is formed. Each element c_{ij} of this matrix gives the estimated schedule length due to merger of i and j (step 9). The C matrix is

Algorithm 1 Clustering Algorithm

```
1:  resources ⇐ ⋃_{All Clusters} Resources per cluster
2:  do_pure_vliw_scheduling(graph, resources)
3:  while (no_of_chains(graph) > n_clusters) do
4:     for (i = 1 to no_of_chains(graph)) do
5:        for (j = 0 to i) do
6:           duplicate_graph ⇐ graph_duplicate(graph)
7:           duplicate_chains ⇐ graph_duplicate(chains)
8:           merge_chains(duplicate_graph, duplicate_chains, i, j)
9:           c_{i,j} ⇐ estimate_sched(duplicate_graph, duplicate_chains)
10:       end for
11:    end for
12:    SORT(C); Sorting priority function:
       i) Increase in sched_length
       ii) Larger communication edges, if sched_length is same
       iii) Smaller chains, if sched_length and communication edges are equal
13:    n_merge ⇐ MIN(0.1 * n_chains, n_chains − n_clusters)
14:    merge top n_merge chains
15: end while
```

sorted according to multiple criteria (step 12). This in order of priority considers, estimated performance represented by schedule length, communication edges and chain lengths. At each stage most beneficial n_{merge} pairs are selected and merged (steps 13 to 14). Step 13, shows a tradeoff between speed and quality of results. The parameter, 0.1 is chosen empirically, after experimenting with various values of this parameter. In case this parameter is too high, a large number of chains get merged and the algorithm converges quickly. If this parameter is kept very low, very few chain mergers get updated at each step and the algorithm takes a long time to converge. Whereas in the latter case, better results are obtained as schedule length estimations are done after a small number of chain mergers, in the former, schedule length estimations become quite skewed.

The schedule estimation algorithm (Algorithm 2), works as follows: Each of the

Algorithm 2 Estimating Schedule (all except RF-to-RF)

```
1: Initialize the schedule step of all nodes in this chain to 0
2: prior_list ⇐ Build priority list
3: curr_step ⇐ 1
4: repeat
5:    Initialize all used capacities to zero
6:    ready_list ⇐ Build ready list
7:    for all (Resources) do
8:       curr_node ⇐ list_head(ready_list)
9:       while (curr_node ≠ NULL) do
10:         rqd_cap ⇐ Incoming External Edges
11:         resv_step ⇐ 0
12:         if (rqd_cap + used_capacity > avail_cap_rd) then
13:            for all (Incoming valid external edges) do
14:               Update schedule of dirty nodes
15:               Find first free write slot from this cluster
16:               used_capacities + = 1
17:               wrt_slot[src_cluster] ⇐ sched_step[src_node] + 1
18:               if (used_capacities ≥ max_cap_wr) then
19:                  used_capacities ⇐ 0
20:                  wrt_slot[src_cluster] + = 1
21:               end if
22:               resv_step ⇐ max(all write slots)
23:            end for
24:         end if
25:         curr_node.sched_step ⇐ max[resv_step, curr_step]
26:         Mark all out nodes as dirty
27:         list_remove(ready_list, curr_node)
28:      end while
29:   end for
30:   curr_step + = 1
31: until (prior_list.n_nodes ≠ 0)
32: Return max. schedule length
```

nodes in the particular merged group of chains is scheduled taking into account data dependency. To simplify scheduling, we assume that each operation takes one cycle. The basic scheduling algorithm used is list scheduling with distance from sink as the priority function. This algorithm makes a best case schedule estimate, so it represents a lower bound on the final schedule. Towards this end, it assumes a

best case connectivity between clusters. However, if in any graph a value needs to be transferred to multiple clusters (broadcast), then transfer operations are scheduled on each consecutive cluster (hops). The algorithm tries to take advantage of bidirectional connectivity architectures by propagating data in both the directions if needed. Also to save computation, if at any stage a node has any outgoing edge, then the node connected to that particular edge is marked dirty. However, the schedule of this particular node and all its children is not updated till it is needed. If at a later stage any node has an incoming edge from this node or its children, then the schedule of the dirty node along with the schedule of all its connected nodes are updated. This leads to a significant saving in computation.

Steps 1 and 2, prepare the graph for scheduling. The priority list, contains the nodes in this chain sorted in descending order of distance-to-sink. The main loop (steps 4 to 31) is repeated till all the nodes in the graph have been scheduled. The second loop body, (steps 7 to 29), tries to schedule as many operations in this cycle as possible. It starts by picking the node at the head of the ready list (step 8) and checks if the external edges feeding this node exceed the read capacity (step 12). The read capacity is simply the number of external values which can be simultaneously read by a cluster. For example for the architecture shown in Figure 3.7(b), the read capacity is one and for architecture shown in Figure 3.8(b) it is two. The node assumes best connectivity between any two clusters. If the required capacity is exceeded, it tries to estimate the effect of transfer delay as well as book transfer slots (steps 14 to 22). When all the nodes in the chain have been scheduled this algorithm returns the value of maximum schedule length in the graph.

We observed the results of this phase on small graphs using a graph visualization tool. The set of heuristics is effectively able to capture all the connected components

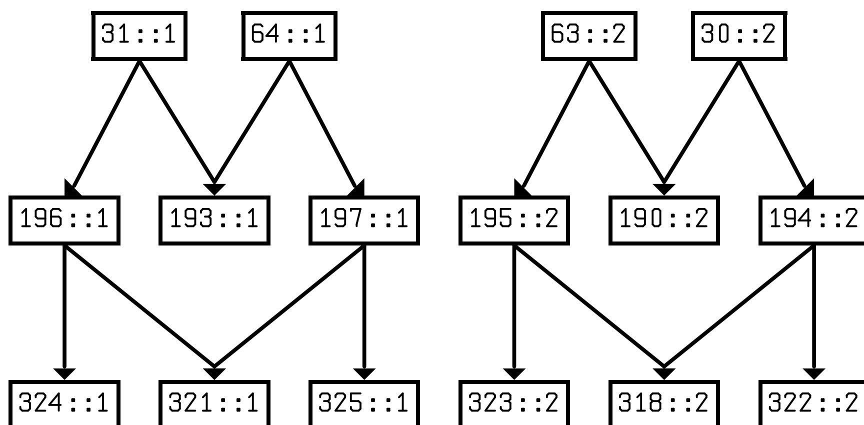


Figure 4.4: Detected Connected Components-I

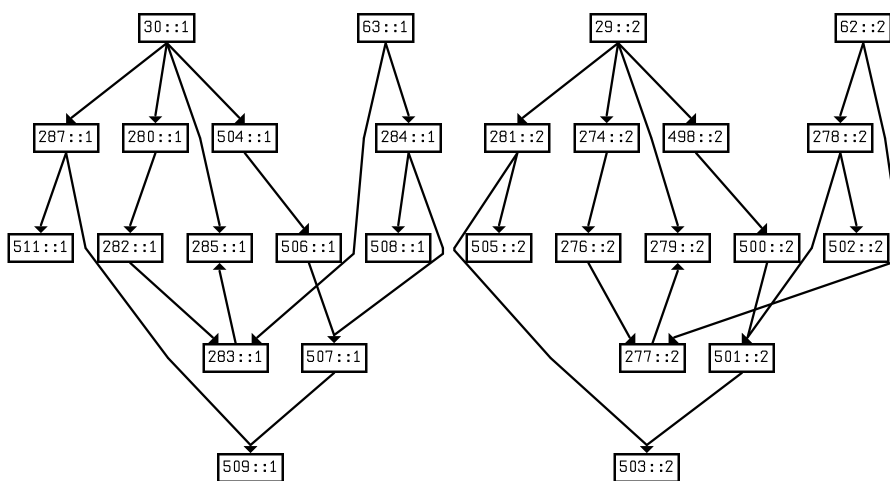


Figure 4.5: Detected Connected Components-II

as described in [Desoli, 1998]. These connected components or parts of graph with heavy connectivity are prime candidates to be scheduled onto one cluster. Using such assignments reduces the inter-cluster bandwidth requirements leading to better schedules. It needs to be noted that the algorithm does not try to merge chains based purely on connectivity. It takes into account the impact on schedule length while performing such mergers. Figures 4.4 and 4.5, show the detected connected components for two examples. The numbers inside the nodes, show *node no.::group no.* The source code in both these cases was doing some computation and the resultant value was being assigned to distinct matrix elements. It needs to be noted that we have not carried out an explicit connected component detection as in [Desoli, 1998].

4.2.5 Binding

The next step is to bind these groups of chains to clusters. Although the value of *n_clusters* is quite small (8 in our case and generally not more than 16), still the number of possible bindings is quite large. This effectively rules out any exhaustive exploration of the design space. The following observation, established through our experimentation, makes this stage extremely important: *while a good result propagation algorithm (to move data across various clusters) can affect the schedule length by around a factor of two, a poor binding at times can lead to schedules which are more than four times larger than the optimal ones.*

The binding heuristics are driven by what impact the communication latency of a particular node will have on the final schedule. In effect we recognize that the data transfer edges from each of the merged group of chains to some other group are not equivalent. Some are more critical than the others in the sense that they would

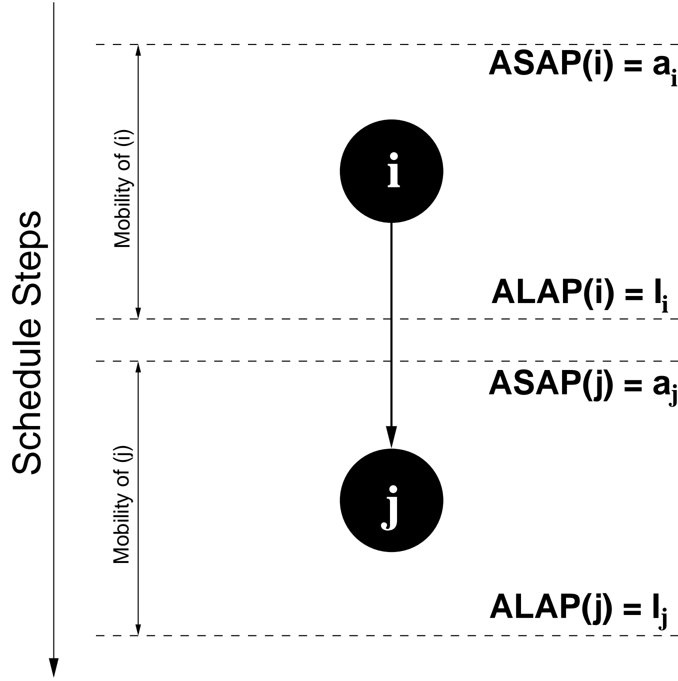


Figure 4.6: Heuristics for Binding

affect the schedule to a larger extent. The heuristics try to capture this, without explicit scheduling (Figure 4.6). We calculate the As Soon As Possible (ASAP) and As Late As Possible (ALAP) schedules for each of the individual nodes. A first order estimate of this impact is given by the mobility of each individual node. Say, we have a communication edge from V_i to V_j, and ASAP and ALAP schedules for these nodes are a_i, a_j and l_i, l_j respectively. Then if $(a_j - l_i) \geq \delta$, where δ is the maximum communication distance between any two clusters, then this edge is not critical at all as there is enough slack to absorb the effect of even the largest communication latency. On the other hand, if $(l_j = a_i + 1)$ the node has zero mobility and is thus most critical. We calculate the weight of each communication edge as follows:

$$W_{i,j} = max\left(0, \delta - \left(\frac{a_j + l_j}{2} - \frac{a_i + l_i}{2}\right)\right)$$

While the weight measure is able to segregate non-critical edges from critical ones, it is not able to distinguish clearly between edges whose nodes have equal mobility. To take this second effect into consideration, we also consider the distance from sink, or remaining path length for each of the source nodes. This path length when multiplied with Wi, j, gives us the final weight for each of the communication edges.

Algorithm 3, shows the binding algorithm. The algorithm works on a weighted connectivity graph, which is generated as discussed above. The initial part of the algorithm (steps 2 to 5) is basically a greedy one. While it seems to work well for architectures which communicate only in one *direction*, the algorithm is not very effective for architectures which can both read from as well as write to the adjacent clusters. Partially motivated by this and partially by [Lapinskii et al., 2001], we thus bring in an additional iterative improvement phase, by performing a local search around this initial binding (steps 6 to 13).

Algorithm 3 Binding Algorithm

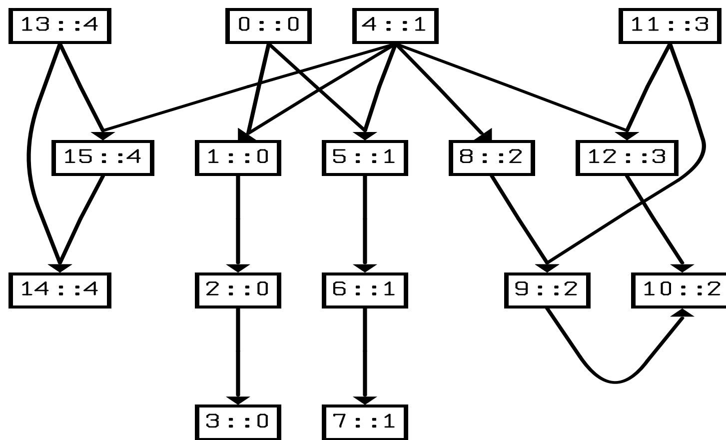
1: $connect_graph \Leftarrow gen_connect_graph(graph, chains)$
2: **while** $(Not\ all\ nodes\ in\ connect\ graph\ are\ bound)$ **do**
3: $\quad source_node \Leftarrow find_highest_weight_edge(connect_graph)$
4: $\quad$ Bind both nodes of this edge to closest clusters
5: **end while**
6: **while** $(Not\ all\ clusters\ have\ been\ considered)$ **do**
7: $\quad prev_sched_len \Leftarrow sched_length(graph)$
8: $\quad$ Swap binding for two adjacent clusters
9: $\quad sched_len \Leftarrow schedule_graph(graph)$
10: $\quad$ **if** $(prev_sched_len < sched_len)$ **then**
11: $\quad\quad$ Swap back bindings for these clusters
12: $\quad$ **end if**
13: **end while**

An example of this appears in Figure 4.7. The input graph is shown in Figure

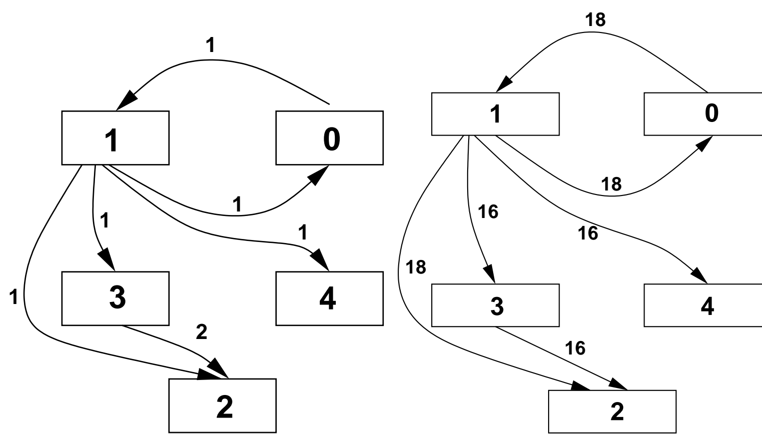
4.7(a) and the corresponding connectivity, with each of the group of chains as a node is shown in Figure 4.7(b). For the connectivity graph, the edge weights represents number of transfers across groups of chains. Looking at this graph, it appears that groups, 2 and 3 which are the most heavily connected need to be assigned to clusters which are close. However, from the input graph in Figure 4.7(a), it is clear that this is not the case. Both the groups accept input from group 1, and till the time that communication doesn't take place, nodes 8 and 12 cannot be scheduled. This makes the connections between groups 1 and 2 more critical than those between groups 2 and 3. The criticality graph shown in Figure 4.7(c), shows that using the set of heuristics above, this fact has actually been brought out.

4.2.6 Final Scheduling

The scheduling phase is architecture specific. We have separate schedulers for RF-to-RF type architectures and direct inter-cluster communication architectures. Although a few scheduling algorithms for clustered architectures have been reported in literature, they are all specifically targeted towards RF-to-RF type of architectures [Desoli, 1998, Lapinskii et al., 2001, Sanchez and Gonzalez, 2000]. Our scheduling algorithm again is a list scheduling algorithm with distance of the node from sink as the priority function. What it additionally contains is steps to transfer data from one cluster to other. The result propagation algorithm, tries to move data to clusters using the hop paths in direct communication architectures. It finds the shortest path to the closest cluster to which data needs to be transferred and moves data to this cluster. If the communication mechanism is bidirectional as in Figure 3.8, it moves data in both the directions. It repeats this process till data has been moved to all



(a) **Input Graph**



(b) **Connectivity** (c) **Criticality**

Figure 4.7: Connectivity and Criticality Between Clusters

the required clusters. During experimentation we realized that efficient result propagation is very important for processors with large number of clusters (more than four).

4.3 Summary

This chapter presented a new framework for design-space exploration of inter-cluster interconnect architectures in clustered VLIW processors. To bypass compiler limitations, the framework works directly with the data-flow of an application. The next chapter presents detailed results obtained using this framework.

Chapter 5

Experimental Results for Design Space Exploration

Chapter 4, presented the design-space exploration algorithms for the design-space of inter-cluster ICNs. The algorithms presented are able to target the entire design-space. In this chapter we present results of exercising the framework on selected architectures. We use the same benchmarks as those in Chapter 2 for this study. The results are presented as a fraction of unclustered VLIW. An unclustered VLIW running at the same frequency as the clustered architecture, represents a theoretical upper limit on the performance which may be achieved, but is unrealizable in practice. This analogy is similar to multi-processors, wherein the ideal speedup obtained using n processors is n, but is impossible to obtain in practice.

5.1 Experimental Results

Our framework is very flexible and supports exploration of a wide design space based on varying interconnect types, number of clusters, cluster composition etc. The architectures for which we present results are shown in Figures 3.6, 3.7 and 3.8. For the RF-to-RF architecture it is assumed that the number of buses is 2 and the bus latency is 1.

			$\delta = 4$			$\delta = 2$		
Bench.	**PV**	**RF**	**WA.1**	**RA.1**	**WR.1**	**WA.2**	**RA.2**	**WR.2**
matrix_init	9.80	6.28	6.12	6.12	6.12	6.12	6.12	6.12
biquad	10.67	2.61	8.56	8.47	8.75	8.75	8.85	9.06
mm	11.68	3.19	6.81	9.28	8.48	8.48	7.80	8.52
insert_sort	9.45	4.09	7.22	7.19	7.80	7.92	7.87	7.94
h2v2_fancy	9.71	2.04	6.02	5.64	6.85	6.84	6.17	6.67
encode_one	10.28	2.19	6.46	6.50	7.50	6.72	6.65	6.69
h2v2_down	11.06	3.43	5.61	5.49	5.94	5.94	5.50	5.90
form_comp	10.75	5.18	5.21	5.43	5.61	5.61	5.52	5.58
decd_mpeg1	10.90	2.16	7.43	7.60	8.61	8.61	8.14	8.67
dist1	7.44	2.62	6.41	7.00	7.02	7.15	6.41	7.15
pred_zero	9.48	4.88	5.26	5.37	5.57	5.57	5.53	5.57
pred_pole	9.56	5.49	5.16	5.16	5.38	5.27	5.16	5.27
tndm_adj	10.19	5.71	6.85	6.34	6.85	6.85	6.73	6.85
update	9.70	2.20	6.31	6.36	7.36	7.36	7.16	7.43
g721enc	10.50	3.62	6.54	6.34	6.72	6.69	6.60	6.63

Table 5.1: ILP for (8-ALU, 4-MEM) 4-Clust Architectures

The obtained ILP numbers for various architectures for different cluster configurations are shown in Tables 5.1, 5.2, 5.3, 5.4, 5.5, 5.6 and 5.7. Here the different benchmarks are the actual functions from the benchmark suites of DSP-Stone and MediaBench as has been discussed in Chapter 2. The register file for each of the ar-

Bench.	PV	RF	$\delta = 6$			$\delta = 3$		
			WA.1	**RA.1**	**WR.1**	**WA.2**	**RA.2**	**WR.2**
matrix_init	15.31	10.65	10.21	10.21	10.21	10.21	10.21	10.21
biquad	16.23	2.00	10.82	10.82	11.02	11.46	11.46	11.46
mm	17.46	2.16	12.68	10.88	12.62	12.42	12.50	12.78
insert_sort	14.17	2.28	7.75	7.60	7.60	7.86	7.81	7.84
h2v2_fancy	14.56	2.07	11.28	8.70	11.94	11.90	10.84	10.21
encode_one	15.43	2.01	11.57	11.01	11.83	12.06	11.64	11.44
h2v2_down	16.59	3.43	6.60	6.36	6.93	6.56	6.36	6.64
form_comp	16.21	6.33	9.20	8.65	9.57	9.37	9.28	9.37
decd_mpeg1	16.29	2.23	11.76	10.17	10.89	12.07	11.08	11.17
dist1	7.44	3.23	6.76	6.31	6.70	7.02	5.81	6.53
pred_zero	14.22	4.39	7.53	7.60	7.98	7.92	8.00	7.84
pred_pole	14.33	3.79	8.90	9.21	9.10	9.21	9.21	8.90
tndm_adj	15.71	3.25	9.67	9.67	9.43	9.92	9.92	9.92
update	14.45	2.05	8.28	8.20	8.20	8.74	8.64	8.64
g721enc	15.86	2.05	11.14	10.57	11.45	11.67	11.05	11.40

Table 5.2: ILP for (12-ALU, 6-MEM) 6-Clust Architectures

Bench.	PV	RF	$\delta = 8$			$\delta = 4$		
			WA.1	**RA.1**	**WR.1**	**WA.2**	**RA.2**	**WR.2**
matrix_init	20.42	11.67	11.14	11.14	11.14	11.14	11.14	11.14
biquad	21.64	2.00	12.98	13.91	15.58	15.58	15.90	16.23
mm	19.64	2.12	9.90	13.06	15.60	15.60	14.69	14.69
insert_sort	18.94	2.21	11.03	12.32	13.48	13.48	13.39	13.62
h2v2_fancy	19.38	2.07	10.00	9.63	13.04	13.08	11.71	12.55
encode_one	19.14	2.18	12.02	13.04	14.44	14.39	14.39	14.70
h2v2_down	19.04	2.05	4.90	6.45	6.35	6.35	6.49	6.45
form_comp	21.72	4.63	7.85	9.91	10.86	10.98	10.98	10.98
decd_mpeg1	21.81	2.18	9.45	10.24	12.40	12.52	12.29	12.64
dist1	7.44	3.38	5.03	5.72	7.02	7.02	5.47	6.64
pred_zero	19.20	4.98	5.37	5.41	5.82	5.77	5.77	5.77
pred_pole	19.85	3.35	9.92	9.92	10.32	10.32	10.32	10.32
tndm_adj	17.95	2.58	11.78	11.42	12.16	12.16	12.16	12.16
update	19.39	3.46	11.20	11.36	13.25	13.03	12.42	12.42
g721enc	21.14	2.07	13.64	12.47	14.17	14.17	13.14	13.51

Table 5.3: ILP for (16-ALU, 8-MEM) 8-Clust Architectures

Bench.	PV	RF	$\delta = 10$			$\delta = 5$		
			WA.1	**RA.1**	**WR.1**	**WA.2**	**RA.2**	**WR.2**
matrix_init	24.50	14.41	13.61	13.61	13.61	13.61	13.61	13.61
biquad	26.86	2.00	12.77	12.37	12.37	13.67	13.43	13.67
mm	29.38	2.20	17.99	13.56	12.68	18.18	16.95	16.63
insert_sort	23.67	2.22	10.84	10.78	10.35	11.32	11.19	11.35
h2v2_fancy	24.06	2.05	11.56	11.36	11.51	11.95	11.79	11.90
encode_one	25.66	2.00	14.18	13.56	12.71	15.80	15.32	15.37
h2v2_down	19.04	2.05	7.17	7.06	7.02	6.83	7.15	7.10
form_comp	26.87	4.70	10.53	10.31	9.04	10.64	10.98	10.98
decd_mpeg1	27.59	2.14	11.86	10.99	9.80	12.52	12.29	12.07
dist1	7.44	2.86	5.47	5.64	4.48	7.02	5.39	6.10
pred_zero	24.00	2.18	11.64	10.24	9.97	12.00	10.82	11.46
pred_pole	23.45	2.02	9.56	12.90	13.58	16.12	16.12	16.12
tndm_adj	17.95	2.48	10.47	9.67	9.92	10.77	10.77	10.47
update	24.09	2.12	14.45	13.03	12.62	15.29	13.95	15.00
g721enc	26.53	2.02	14.03	11.86	11.77	14.59	14.03	13.76

Table 5.4: ILP for (20-ALU, 10-MEM) 10-Clust Architectures

Bench.	PV	RF	$\delta = 12$			$\delta = 6$		
			WA.1	**RA.1**	**WR.1**	**WA.2**	**RA.2**	**WR.2**
matrix_init	30.62	22.27	20.42	20.42	20.42	20.42	20.42	20.42
biquad	31.16	2.00	13.43	13.20	12.98	14.98	15.27	15.27
mm	35.26	2.06	19.59	12.87	13.77	19.16	17.81	18.36
insert_sort	28.34	2.22	10.81	10.52	10.52	11.06	11.25	11.35
h2v2_fancy	28.80	2.06	13.32	12.92	11.82	18.31	17.51	18.35
encode_one	30.98	2.00	21.31	16.92	16.64	22.38	20.66	21.20
h2v2_down	19.04	3.53	6.96	6.87	6.68	6.75	6.87	7.00
form_comp	32.94	2.95	10.53	9.82	8.88	10.42	10.42	10.64
decd_mpeg1	32.98	2.12	15.19	11.46	10.82	16.10	14.86	15.19
dist1	7.44	2.84	5.10	5.55	4.18	7.02	5.47	6.76
pred_zero	28.44	2.03	18.29	14.77	14.77	19.20	17.07	17.86
pred_pole	23.45	2.72	8.60	11.73	12.29	14.33	14.33	14.33
tndm_adj	17.95	2.50	12.16	14.50	13.00	17.14	16.39	16.39
update	28.39	2.58	18.49	17.28	15.00	21.49	19.39	20.92
g721enc	31.72	2.05	20.55	16.97	16.39	21.78	20.26	21.14

Table 5.5: ILP for (24-ALU, 12-MEM) 12-Clust Architectures

Bench.	PV	RF	$\delta = 14$ WA.1	RA.1	WR.1	$\delta = 7$ WA.2	RA.2	WR.2
matrix_init	35.00	27.22	24.50	24.50	24.50	24.50	24.50	24.50
biquad	31.16	2.07	13.67	13.20	12.56	14.43	13.91	14.70
mm	41.00	2.06	20.03	13.77	12.33	18.36	17.28	18.18
insert_sort	33.18	2.22	13.84	13.77	12.90	14.91	14.59	15.01
h2v2_fancy	33.57	2.06	18.30	12.46	12.76	18.82	17.99	17.57
encode_one	35.96	2.02	12.28	12.21	10.77	13.29	13.08	13.21
h2v2_down	19.04	3.53	6.93	6.87	6.64	6.77	6.87	7.00
form_comp	37.81	4.69	10.31	9.20	8.96	10.21	10.21	10.21
decd_mpeg1	38.63	2.21	14.38	11.96	10.73	19.59	16.90	18.78
dist1	7.44	2.50	4.33	5.17	3.80	7.02	5.24	7.15
pred_zero	33.39	2.10	16.34	16.00	14.49	20.76	19.69	20.21
pred_pole	23.45	2.02	7.37	11.22	6.97	17.20	17.20	18.43
tndm_adj	17.95	2.42	10.77	15.71	10.77	17.14	17.14	17.14
update	28.39	2.13	18.93	15.59	14.45	23.38	20.38	21.49
g721enc	37.41	2.00	20.55	17.58	16.39	24.73	23.16	24.32

Table 5.6: ILP for (28-ALU, 14-MEM) 14-Clust Architectures

Bench.	PV	RF	$\delta = 16$ WA.1	RA.1	WR.1	$\delta = 8$ WA.2	RA.2	WR.2
matrix_init	40.83	27.22	24.50	24.50	24.50	24.50	24.50	24.50
biquad	31.16	2.06	12.98	15.90	12.37	24.34	24.34	22.91
mm	47.65	2.18	20.50	11.09	10.75	26.71	21.24	20.99
insert_sort	38.06	2.21	15.91	15.28	14.32	16.52	16.45	16.59
h2v2_fancy	39.04	**X**	**X**	**X**	**X**	**X**	**X**	**X**
encode_one	41.10	2.02	19.00	15.73	15.14	21.20	20.45	19.94
h2v2_down	19.04	4.01	5.91	5.93	5.76	5.87	5.94	6.00
form_comp	44.39	4.91	10.21	9.54	8.51	10.31	10.11	10.21
decd_mpeg1	40.97	2.23	10.90	9.39	9.07	11.36	10.32	11.56
dist1	7.44	2.50	3.54	4.96	3.35	7.02	6.31	7.15
pred_zero	38.40	2.07	12.19	14.22	13.24	19.69	16.34	19.69
pred_pole	23.45	2.02	5.61	10.32	9.21	19.85	19.85	19.85
tndm_adj	17.95	2.34	9.92	16.39	10.77	17.14	17.14	17.14
update	28.39	2.17	15.90	15.00	13.95	23.38	20.92	22.71
g721enc	42.91	2.02	13.90	12.80	11.67	14.59	14.17	14.17

* X implies that computation was abandoned due to excessive time consumption

Table 5.7: ILP for (32-ALU, 16-MEM) 16-Clust Architectures

chitectures has sufficient number of ports to allow simultaneous execution on all the FUs e.g. for the monolithic RF architecture of Table 5.3, the RF has $24*3 = 72$ *Read* and $24*1 = 24$ $Write$ ports (assuming 1 read port for predicate input). This is done so as to ensure that there is no concurrency loss due to insufficient number of ports and the effect of interconnection architecture stands out. Abbreviation PV (column 2) has been used in this table for *Pure VLIW (or single RF)*, *RF* (column 3) for RF-to-RF, *WA* for *Write Across* (column 4 and 7), *RA* for *Read Across* (column 5 and 8) and *WR* for *Write/Read Across* (column 6 and 9). The entries in these tables, show the ILP for each of the benchmark functions e.g. for insert sort for a 4-cluster configuration, the ILP for a write across architecture (Figure 3.7(a)) is 7.22, for read across (Figure 3.7(b)) it is 7.19. From the results we conclude the following:

1. Loss of concurrency vis-a-vis pure VLIW is considerable and application dependent.

2. In few cases the concurrency achieved is almost independent of the interconnect architectures. This denotes that a few grouped chains in one cluster are limiting the performance along with a few critical transfers.

3. For applications with consistently low ILP for all architectures, the results are poor due to a large number of transfers amongst clusters.

4. In some cases the performance in case of $n_clusters = 4$ architecture is better than performance in case of $n_clusters = 8$ architecture (dist1). This is because of the reduced average hop distance amongst clusters and this behavior is common across different architectures. In such cases communication requirements dominate over the additional computational resources provided by the

clusters.

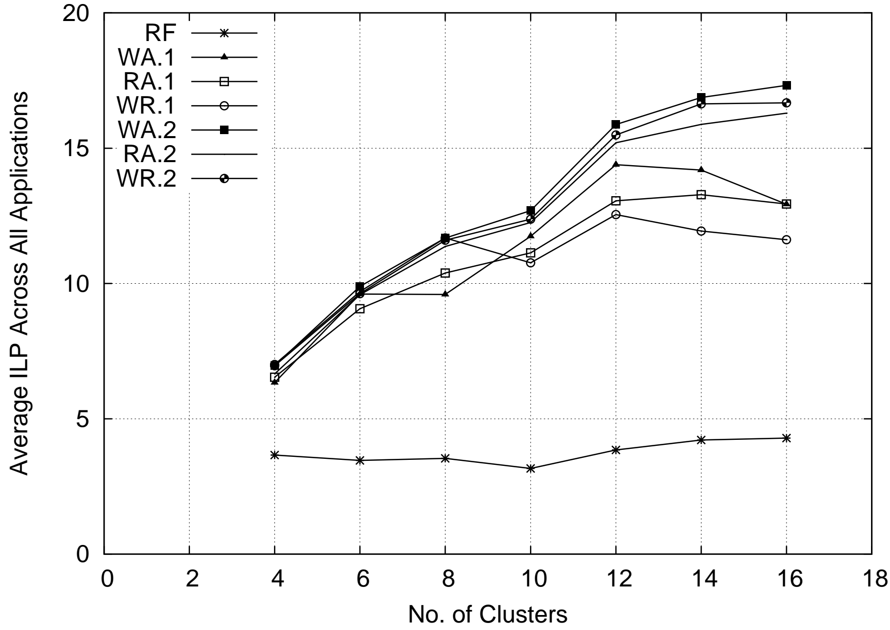


Figure 5.1: Average Variation in ILP With $n_{clusters}$

Figures 5.2 and 5.1 show the average application performance across architectures with a variation of $n_clusters$. Whereas Figure 5.1, shows the absolute ILP averaged across applications, Figure 5.2, shows this average ILP as fraction of Pure VLIW. It needs to be noted that number of cycles and cycle-time itself are two metrics for processor performance. Our further experiments, reported in [Gangwar et al., 2005], have established that the cycle-time is severely limited in case of unclustered VLIWs as well as RF-to-RF mechanisms. Whereas in case of other interconnects, there is as little as 10% variation in cycle-time.

For $n_clusters$ less than 8, the behavior of different interconnection networks is not brought out, however, once $n_clusters$ grows beyond 8, the superiority of *WR.2* and *WA.2* is clear. Both of these are able to deal well with the applications which

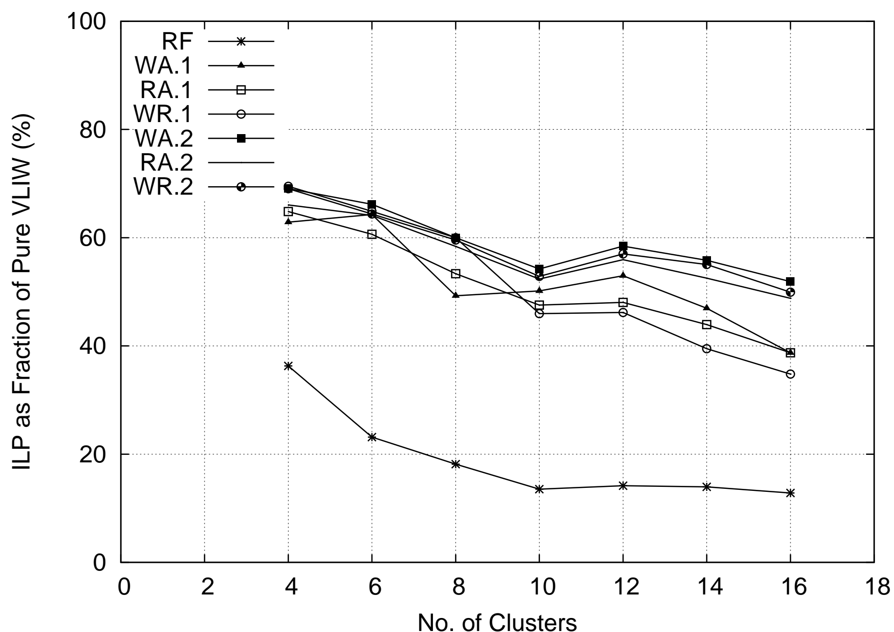


Figure 5.2: Average Variation of ILP as fraction of Pure VLIW With $n_clusters$

require heavy communication. The minimum loss of concurrency in these cases is around 37% and 30% for architectures with $n_clusters = 8$ and $n_clusters = 4$ respectively. It needs to be noted that *WR.2* is the type of interconnect employed by most commercial as well as research architectures as shown in Table 3.1. The performance of *RA.2* in general, is inferior to *WA.2* because if there is more than one consumer of the propagated value in the target cluster, the value first needs to be moved to target cluster using the available read path, which amounts to an additional processor cycle. It is interesting to note that the performance of RF-to-RF type of architecture is quite poor, with an average loss of 81% for $n_clusters = 8$ and 64% for $n_clusters = 4$ and deteriorates further with increase in the number of clusters. This is ignoring the latency of such transfers using global buses (assumed bus latency is 1) vis-a-vis local transfers. In a real scenario the global buses would either be much slower than the rest of the system (multi-cycle) or pipelined to prevent the system

clock period from dropping [Gangwar et al., 2005]. It would be quite interesting to identify application characteristics by which the suitability of an architecture can be established.

5.2 Summary

In this Chapter, we evaluated a range of clustered VLIW architectures. The results conclusively show that application dependent evaluation is critical to make the right choice of an interconnection mechanism. Also, it is quite evident from the results that all the architectures are inter-cluster communication bandwidth constrained. Once the communication delay starts dominating, the different inter-cluster transfer mechanisms fail to achieve performance close to Pure VLIW. This happens even though the supported concurrency has increased due to an increase in the number of available FUs. Also, the RF-to-RF type of architectures considered here, fail to perform to expected levels because even for adjacent clusters they have to gain access to global buses which are shared. The next chapter explores, effect of interconnection architecture on clock period and combines the results obtained here with those from clock period for a comprehensive performance estimate.

Chapter 6

Effect on Clock Period of Inter-cluster Communication Mechanisms

Chapter 5, established that considerable performance gains may be obtained by using a variety of inter-cluster ICNs. While there we established the need for more variety in inter-cluster interconnect mechanisms we did not report implementation characteristics of these mechanisms. Several questions need to be answered before one type of interconnect can be chosen over other: What is the impact on clock period of these interconnect mechanisms? What is the penalty in terms of increase in interconnect area which these mechanisms result in? etc. Also, the bus based mechanisms were evaluated using a few specific design points. In Andrei at. el. [Terechko et al., 2003] a single cycle transfer bus was connected to all the clusters whereas in Chapter 3 two single cycle transfer buses were connected to all the clusters.

In this chapter we extend the previous reported work on performance of bus based

inter-cluster interconnects in clustered VLIW processors as well as show the implementation characteristics of the various other ICNs. We start by showing that the clock-period is heavily constrained beyond two cluster architectures if single cycle transfer buses are used. Towards this end, we model the various clustered architectures in VHDL and perform synthesis, place and route using industry standard tools. We next use these results to arrive at realistic latencies for multi-cycle and pipelined buses such that the clock-period is not affected. Finally we carry out exhaustive performance evaluation of these bus-based architectures using the obtained latencies.

6.1 Architecture of Modeled Processors

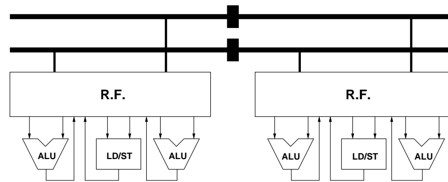


Figure 6.1: Pipelined RF-to-RF

To focus primarily on the inter-cluster interconnect mechanisms, we model a family of very simple processors. However, this set of processors support all the interconnect mechanisms as discussed in Chapter 3. Additionally the pipelined interconnect mechanism which has not been discussed there is shown in Figure 3.8. Interconnect mechanisms presented in Andrei at. el. [Terechko et al., 2003] are a subset of this domain. Authors additionally dealt with the problem of issue slots in Andrei at. el. [Terechko et al., 2003]. We have made specific choices regarding issue slots and these have been explained at appropriate places later on.

Mnemonic	Explanation	Cycles
NOP	No Operation	1
ADD	$R_{dest} \Leftarrow R_{src1} + R_{src2}$	1
SUB	$R_{dest} \Leftarrow R_{src1} - R_{src2}$	1
MUL	$R_{dest} \Leftarrow R_{src1} * R_{src2}$	3
DIV	$R_{dest} \Leftarrow R_{src1}/R_{src2}$	3
MOV	$R_{dest} \Leftarrow R_{src}$	1
LD	$R_{dest} \Leftarrow MEM[R_{src1}]$	3
ST	$MEM[R_{dest}] \Leftarrow R_{src1}$	3
CMP	$R_{dest} \Leftarrow R_{src1} > R_{src2}$	1
BRL	Branch if less to *Address*	3
BRA	Branch always to *Address*	3
MVIH	Load (Immediate) higher order bytes of R_{dest}	1
MVIL	Load (Immediate) lower order bytes of R_{dest}	1
RFMV	Inter-cluster move for RF-to-RF Architectures	1

Table 6.1: Instruction Set Architecture of Modeled Processors

All the processors in our set have a common Instruction Set Architecture (ISA). The ISA resembles that of a simple RISC processor having load/store architecture with standard ALU, data movement and branch operations. This is shown in Table 6.1. In this table the column *Mnemonic*, gives the instruction mnemonic, column *Explanation*, gives instruction semantics and column *Cycles*, shows number of cycles required to execute this instruction. For simplicity, our modeled processors contain only integer operations but addition of floating point operations is a simple extension. Each of the operations with the exception of immediate move (MVIH and MVIL) instructions takes register addresses in source and destination fields. Even the load (store) instructions require the address to be present in a register before the data can be loaded (stored) from (to) memory. This has been done so as to keep the control part as simple as possible, while the data path still mimics a real processor. Immediate move operation has been broken into MVIH and MVIL instructions, which load half the higher order and lower order bytes respectively. This is done to keep

the instruction size same for all instructions. For example if the data-width is set as 32, MVIH loads two most significant bytes and MVIL two least significant bytes into the destination register. RFMV instruction is special and is present only in RF-to-RF type of architecture. It moves values from source cluster (cluster in which this instruction is issued) to any destination cluster.

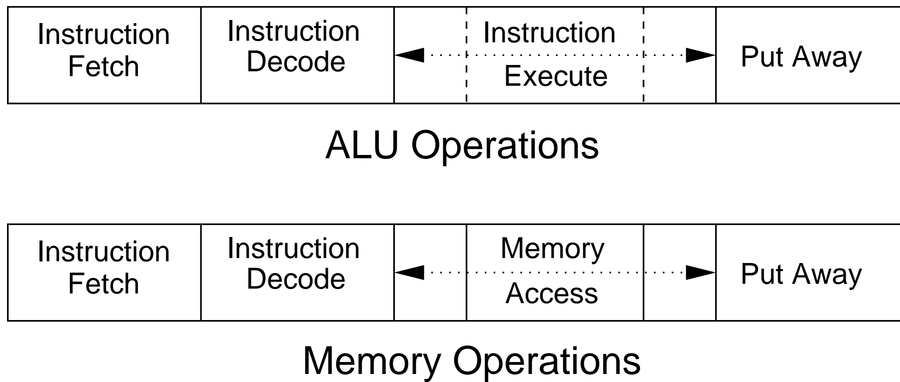


Figure 6.2: Processor Pipeline

The processor has a reduced version of the classical five stage pipeline. The various pipeline stages are shown in Figure 6.2. All the operations are fully pipelined. ALU instructions take variable number of cycles in their execution stage (1-3), while load and store instructions take three cycles each. Load/store unit is again fully pipelined and a new operation can be started on them in each cycle. The branch instructions, BRL and BRA take three cycles each to complete and require a pipeline flush.

Figure 6.3 gives the instruction encoding for various instructions. For *Write Across* type of architectures one additional bit is required in the destination register field as these architectures can write to adjacent clusters. The *Read Across* architectures can read one operand from adjacent cluster and thus an additional bit is required in one of the source register fields. Using similar reasoning, *Read/Write*

across instructions require an additional bit in one of the source operand fields and the destination register field, as they can both read from and write to the adjacent cluster. The *RF-to-RF* architectures contain specialized instructions to move data between clusters (copy operations). This instruction is issued in the regular ALU issue slot and not in any additional issue slot. Destination register in this case consists of the address of destination cluster and the register in that particular cluster. Since, we have considered architectures with upto ten clusters, the destination cluster field contains only four bits. The destination register field requires $log_2(RF\ Size)$ bits to point to a unique location in the destination clusters RF. Encoding for other instructions is straight forward. The instruction encoding has deliberately been kept simple, to ensure a low complexity instruction decoder.

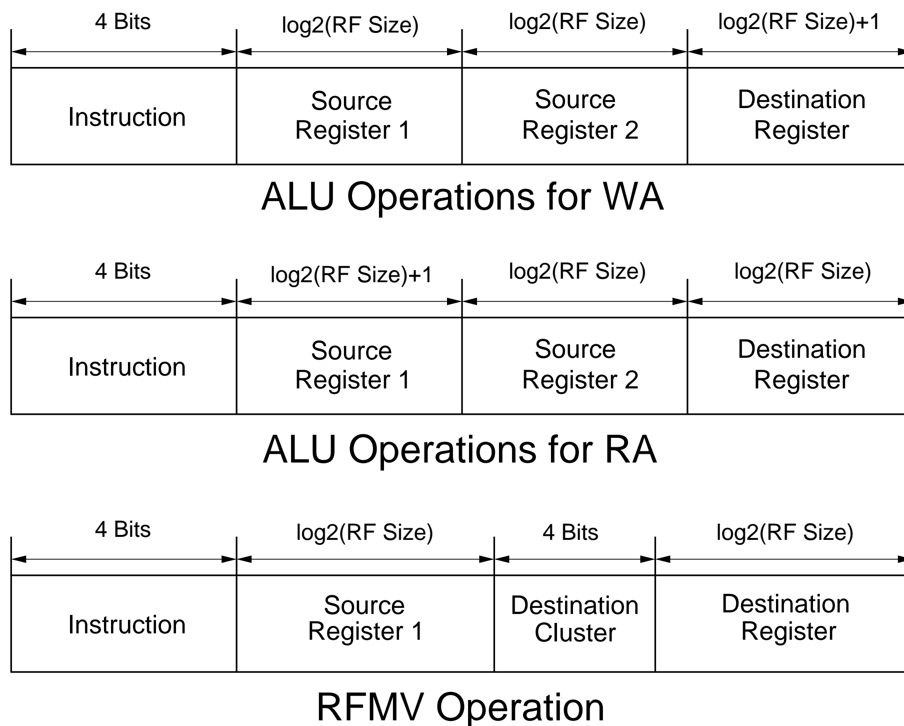


Figure 6.3: Instruction Encoding

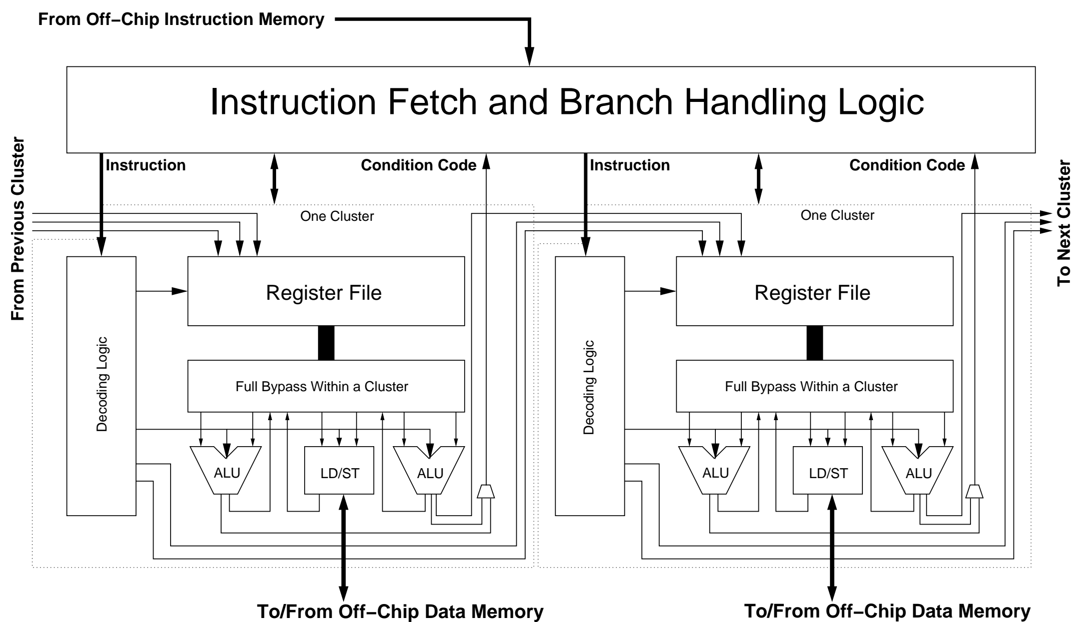


Figure 6.4: Detailed Architecture of Clusters for WA.1

Figure 6.4, shows the detailed architecture of modeled processors for *WA.1* type of inter-cluster communication. There is no Instruction or Data cache in this processor. Also, a centralized cache presents scalability problems and this area is being actively researched [Gibert et al., 2003]. To circumvent such problems, we chose to model a cache-less processor. The off-chip data memory is assumed to be banked, so, each load store unit has a direct path to a particular data memory module. The instruction RAM supplies an entire VLIW word in parallel i.e. single instruction fetch. It is assumed that the RAM has sufficient data-width to handle such parallel data transfer either through banking or other mechanisms. This helps keep the instruction fetch mechanism simple and removes any need for an instruction align mechanism.

Any instruction execution progresses as follows: The centralized instruction fetch unit gets instruction from the off-chip instruction RAM in a single cycle. This is

registered and presence of a branch instruction is tested. The branch instruction can only be issued in the first instruction slot. If a branch instruction is found, the next instruction is fetched from the branch target location depending on the branch instruction as well as the condition code. The condition code itself is obtained from the various clusters and is registered inside the branch handling logic i.e. a centralized place. Since, the branch instructions always contain an absolute location encoded in the instruction itself no other information is required from the clusters to execute this branch instruction.

If a branch instruction is not found, then normal execution must continue. In this case, the long instruction (*MultiOp*) is broken up into cluster level instructions (*UniOps*) and forwarded to each cluster. Further decoding is done in the decode logic of each cluster itself. In case any compare operation is executed, then the condition code is latched in the control register inside the branch handling unit during the put away stage of pipeline. Memory operations requiring access to off-chip data memory have direct paths using the load/store units present in each of the clusters. For inter-cluster communication, read, write, control and data signals are generated individually from each of the clusters. In the *Write Across-1* type architecture, one write port of the RF in any cluster is connected directly to the control and data signals from adjacent clusters. These control signals are generated from the decode unit of adjacent clusters. The ALU output lines are directly connected to the data port. In case any inter-cluster write needs to be executed, the control logic sets the corresponding enable bit high. Each cluster contains a full bypass logic. However, inter-cluster transfer paths feed directly to the RF and not to the bypass logic. This was done because, we believe that feeding to the bypass would result in a critical path through it.

The architectures for other interconnect mechanisms are similar. *Read Across* architectures have input paths to the ALUs after muxing. Generation of control signals for these are handled in the decode logic within each cluster, rather than the centralized unit. *RF-to-RF* architectures, rely on global broadcast of data and register address. This is handled by decode logic within each cluster. Bus arbitration in this case is not required as the instructions have been scheduled by the compiler resolving the bus contention. However, in case of multiple buses, additional muxing logic is required at each RF's data input and output ports. This is assuming that each bus doesn't require any additional port in RF of each cluster. What this effectively means is that only one data element can be moved to any cluster at a time, however, multiple such transfers may be in progress at the same time. These mechanisms have been discussed at length in Andrei at. el. [Terechko et al., 2003]. The muxing logic if present, is again controlled by decode logic within each cluster. Here we have not presented any results for multi-bus architectures.

The processors are modeled using parameterized RTL description (VHDL). The ALU components: adder, multiplier and divider are obtained using the Synopsys DesignWare libraries. This ensures that the critical path is not constrained due to inefficient implementation of these modules. The load/store unit follows standard Static RAM (SRAM) timing, such as that presented in the Leon Processor [Gaisler, 2001]. RF for any processor is custom designed [Tseng and Asanovi, 2003] [Cruz et al., 2000]. We have a RF model without the detailed internal logic as the RF design would normally be using some efficient storage cells. This RF doesn't contain the complicated muxing logic which a real RF has, however, it does ensure that the proper port mappings are in place so that the static timing analysis (STA) reports proper paths. The entire VHDL has been written at Register Transfer Level

(RTL) to obtain an efficient implementation. The parameterized VHDL model allows variation in the following features:

1. Interconnect architecture
2. Number of clusters
3. Register file size
4. Register file (and ALU) width
5. Data RAM size
6. Instruction RAM size

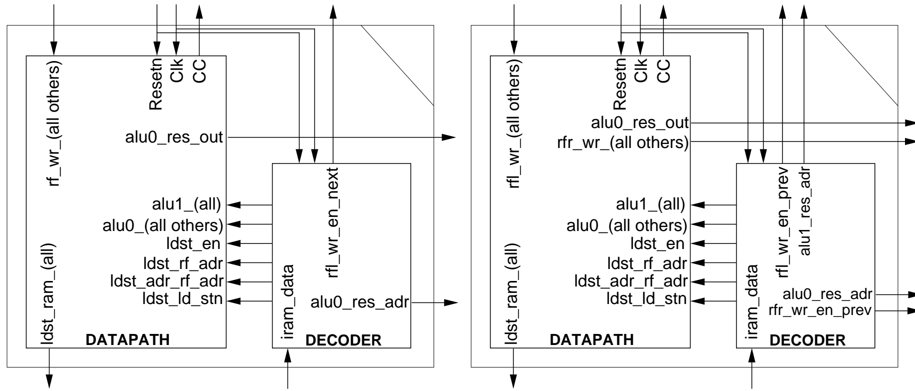
We use the industry standard, Synopsys frontend and Cadence backend flow. Synopsys Design Compiler, synthesizes the RTL to a gate-level netlist. To obtain the best results from Design Compiler, the system is given a very tight timing constraint. This gate-level netlist is then placed and routed using Cadence SoC Encounter. Functional simulation is done using test programs at various stages of implementation to ensure correctness of design. Initially we experimented with a flat place and route, however, the results were not satisfactory. In a flat place and route, the tools moved decode and other logic to far away locations, which naturally led to long critical paths through this logic. Thus to obtain analyzable results we perform a hierarchical place and route. Towards this end, the clusters, as shown in Figure 6.4, are individually synthesized and a flat place and route done on them. After this, timing and hierarchical abstracted models are created which are used further in the flow. The top level flow, doesn't redistribute the logic contained in these models, as it treats them as black-boxes. The only limitation is that the tools currently

do not support rearrangement of pins on black-boxes during a top level place and route run, once they have been created earlier. To circumvent this problem, we do a manual pin assignment based on final expected chip floorplan. The clk and reset pins are handled separately, for which buffer insertion is done so as to maintain the best possible timing.

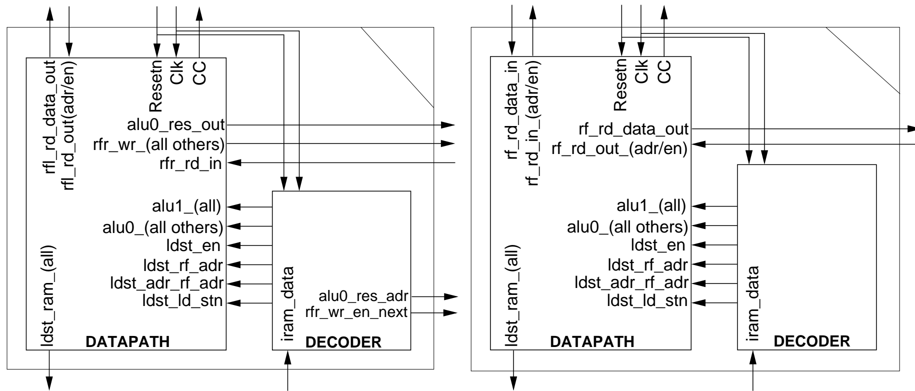
6.2 Evaluation Methodology

Figure 6.5, shows the pin organization for each of the cluster types. The clusters are placed and routed with our supplied pin positions. The clusters as shown, with the slit at the top right corner are in upright position. During top-level place and route, the tool re-positions as well as rotates the clusters to obtain better routability and timing. We maintain an aspect ratio of one for each of the clusters, as this gives good final floorplans. However, the final chip (entire processor) is not necessarily square for all cluster configurations.

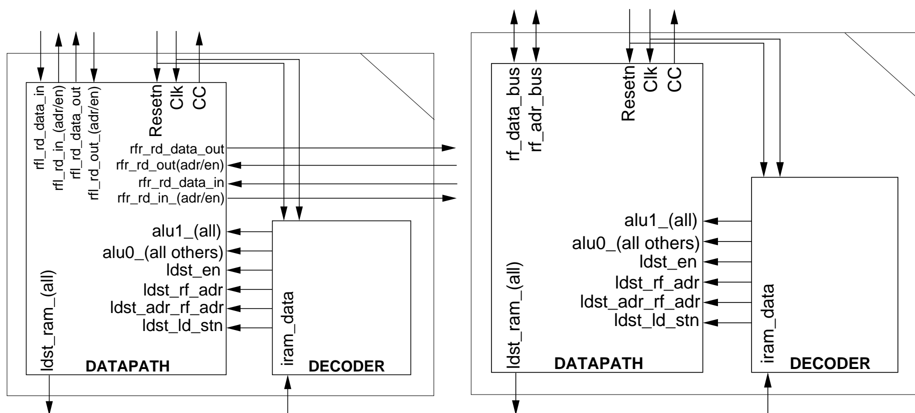
To understand why the pin positions have been chosen in this manner, it needs to be noted that some of the wires go to the centralized branch handling and instruction fetch unit, while other wires are needed for either connecting to the adjacent cluster or for load/store unit's communication with off-chip data RAM. The type of interconnects which we are evaluating can be efficiently routed if the clusters themselves are organized in a ring fashion. This is so because apart from *RF-to-RF* type of architecture, all other architectures only rely on neighbor connectivity. The pins are so organized that a simple rotation of the individual cluster can provide good connectivity. The pin positions on clusters aid such a final placement during floorplanning. Since, the final chip need not be square, we have fixed various aspect ratios



(a) WA.1 (b) WA.2



(c) WR.2 (d) RA.1



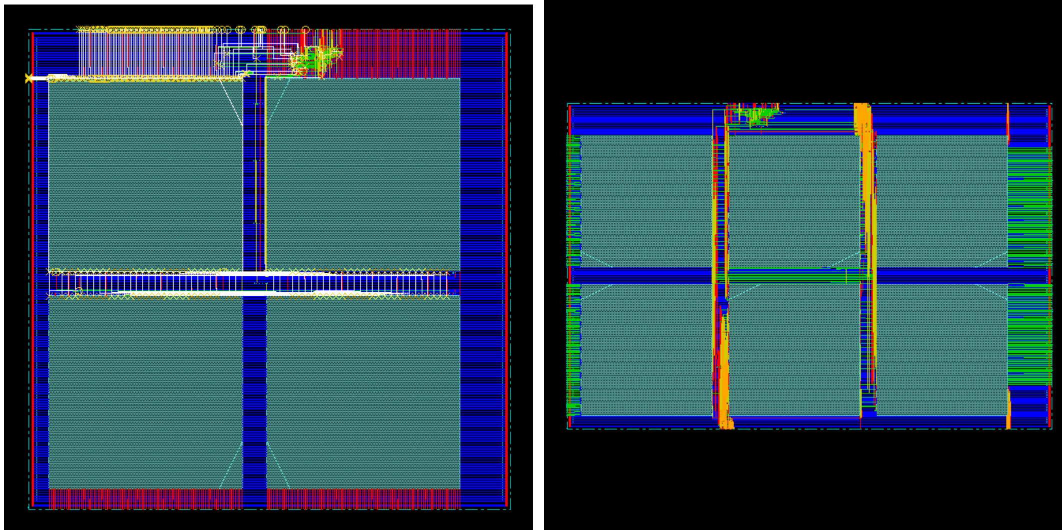
(e) RA.2 (f) RF

Figure 6.5: Pin Organization for Various Clusters

for various cluster configurations. For example in case of a two cluster configuration the ratio is 0.5 and in case of a four cluster configuration it is 1.0 etc. The *RF-to-RF* type of architectures are also organized in a similar fashion. The inter-cluster buses are laid out in between the clusters.

For *Write Across* type of architectures, shown in Figure 6.5, the *rf_wr*, wires are RF ports, which have directly been pulled out for adjacent cluster to be able to write to this RF. In case of bidirectional connectivity these are labeled with prefixes *rfl_* and *rfr_* which denote left and right cluster respectively. The *alu0_* and *alu1_* signals are mapped to write ports of left and right clusters respectively. For the *Read Across* type of architectures, the *_in_* type of ports are used for reading in data from adjacent clusters. If the architecture has bidirectional connectivity, then the prefixes, *rf_* and *rfr_*, denote the left and right clusters respectively. The *_out_* type of wires are used to provide data to adjacent cluster. They are directly mapped to a read port on the RF. *Read/Write Across* architecture, contains a subset of connections of *Read Across* and *Write Across*. The most simple connectivity is in case of *RF-to-RF* architectures. They only require bidirectional address and data buses, which are neatly organized towards the top of each cluster.

Figure 6.6, shows the obtained floorplans after running the floorplanning tool for *RF-to-RF* type architectures. The slits, as mentioned previously, denote the top in an upright position of individual clusters. It is clearly seen that the clusters have been rotated and shifted to obtain the best timing. For the two cluster configuration, the inter-cluster transfer buses have been arranged in between them vertically, which is the best possible way to connect. For four cluster configuration, the buses, run horizontally between all the clusters. For six cluster configuration, the buses have been arranged in the form of an *H*, which is one type of arrangement. The other



(a) Four Cluster Configuration (b) Six Cluster Configuration

Figure 6.6: Floorplans for RF-to-RF Architectures

option was to position the buses horizontally. However, the obtained timing in both these cases is same. The wires coming out of each cluster are the RAM signals from load/store units of clusters to off-chip data RAM blocks. The top level module (full processor) does not have any constraint on pin placements, so, the off-chip connectivity pins are extended to nearest chip boundary. The floorplan for other interconnect architectures, *Write Across*, *Read Across* and *Read/Write Across*, is similar to what has been obtained for *RF-to-RF* architecture. At most the vertical and horizontal alignments are slightly different to provide straight line connectivity between clusters.

6.3 Results of Clock Period Evaluation

We ran our experiments for two ASIC technologies, UMC's 0.13μ and 0.18μ Six metal layer process and one FPGA technology, Xilinx XC2V8000-FF1152-5. The Xilinx Device itself has been fabricated in a 0.15μ CMOS process. The results are presented in Tables 6.2, 6.3, 6.4, 6.5 and Figure 6.7. Tables 6.2 and 6.3 show variation of clock period for different interconnect mechanisms with an increase in the number of clusters. Tables 6.4 and 6.5, show variation of interconnect area as a percentage of the total chip area for different interconnect mechanisms with an increase in number of clusters. Figure 6.7, gives the variation of clock-period for Xilinx XC2V8000-FF1152-5 FPGA. Looking at the tables for clock period, a few observations are quite obvious.

1. The increase in clock period for RF-to-RF architecture, with global buses is very high as soon as the number of clusters increases beyond two.

2. For other interconnect architectures, the variation across different interconnect mechanisms is negligible.

3. There is a small increase in clock period with an increase in number of clusters for interconnect mechanisms other than *RF-to-RF*.

4. Bidirectional connectivity architectures, *WA.2* and *RA.2*, lose out somewhat in terms of clock period as compared to their unidirectional connectivity architectures and this impact is more in smaller cluster configurations.

To analyze why this is so, we traced the timing paths as reported after STA. For two cluster configuration the critical path (s) never run through the inter-cluster

$N_{Clusters}$	**WA.1**	**RA.1**	**WA.2**	**RA.2**	**WR.2**	**RF**
2	0.851	0.859	1.036	0.894	0.891	1.152
4	0.883	0.944	1.030	1.001	0.956	1.987
6	0.929	0.977	1.104	1.015	1.051	2.760
8	1.005	1.065	1.077	1.108	1.112	3.891
10	1.026	1.064	1.126	1.091	0.995	5.110

Table 6.2: Clock Period (ns) for UMC 0.13μ ASIC Tech.

$N_{Clusters}$	**WA.1**	**RA.1**	**WA.2**	**RA.2**	**WR.2**	**RF**
2	1.161	1.123	1.174	1.192	1.153	1.246
4	1.375	1.243	1.404	1.361	1.215	2.103
6	1.367	1.294	1.522	1.508	1.401	3.856
8	1.459	1.280	1.455	1.416	1.348	4.993
10	1.501	1.395	1.480	1.475	1.407	6.220

Table 6.3: Clock Period (ns) for UMC 0.18μ ASIC Tech.

interconnect. This is understandable as in all the cases these interconnects can be efficiently laid out. Even in the *RF-to-RF* type inter-cluster interconnect reduces to connectivity among adjacent clusters. As shown in Figure 6.6 this has been laid out very efficiently. The critical path is through the centralized part of microarchitecture, wires which carry condition code to the centralized control register. For higher cluster configuration (beyond 2), the critical path for *RF-to-RF* architectures is through the interconnect bus. As a consequence this increases monotonically with the increase in number of clusters. However, for other interconnect mechanisms the critical path is still through the condition code setting mechanism.

$N_{Clusters}$	**WA.1**	**RA.1**	**WA.2**	**RA.2**	**WR.2**	**RF**
2	39.45	39.34	39.42	39.16	39.33	43.43
4	36.71	36.65	36.63	36.53	36.65	36.67
6	35.69	35.66	35.73	35.61	35.55	35.71
8	34.90	34.62	35.02	34.82	34.84	34.98
10	34.89	34.84	34.95	34.73	34.81	34.89

Table 6.4: Interconnect Area (as % of Chip Area) for UMC 0.13μ ASIC Tech.

$N_{Clusters}$	**WA.1**	**RA.1**	**WA.2**	**RA.2**	**WR.2**	**RF**
2	37.01	37.05	37.11	36.89	37.01	37.01
4	34.92	35.00	35.02	34.90	34.95	34.97
6	34.21	34.17	34.26	34.13	34.19	34.17
8	33.60	33.54	33.63	33.53	33.59	33.63
10	33.55	33.56	33.64	33.55	33.53	33.62

Table 6.5: Interconnect Area (as % of Chip Area) for UMC 0.18μ ASIC Tech.

Tables 6.4 and 6.5, show the interconnect as percentage of chip area for two ASIC technologies. A few conclusions are immediately clear:

1. While the interconnect lengths do vary in case of unidirectional and bidirectional architectures. This increase, is in proportion to the increase in logic area.

2. The decrease in interconnect area with the increase in the number of clusters is around 4%.

3. The variation in interconnect area even with the increase in number of clusters is not much. This is due to a corresponding increase in the logic area. Basically the interconnect area per cluster is more or less constant.

We use our design-space exploration methodology, as discussed in Chapter 4 using the realistic bus latencies for multi-cycle bus configurations. Cumulative results using both number of cycles and cycle time itself are presented in the next section.

Figure 6.7, shows the variation of clock period for a FPGA Technology. Apart from the mostly monotonic increase in clock period there is not much inference which can be drawn. The FPGA synthesis as well as mapping, placement and routing tools use a flat structure. The critical path in all these cases never was through the inter-cluster interconnect, but rather some other ALU or control signal. Which effectively

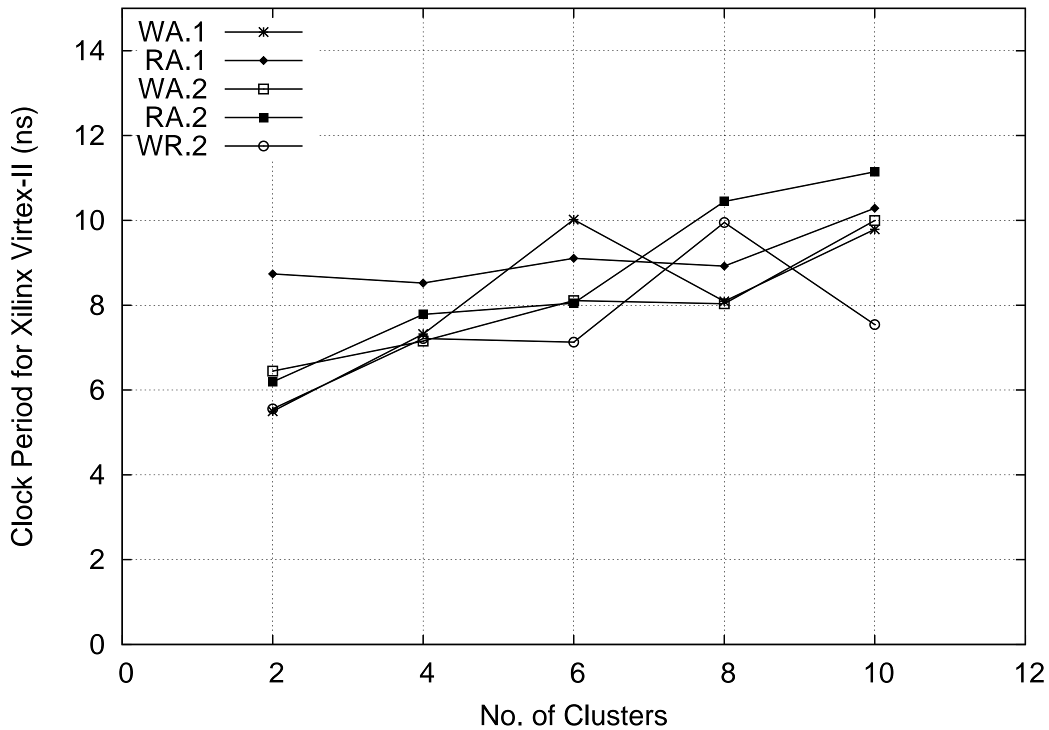


Figure 6.7: Variation of Clock Period for Xilinx XC2V8000-FF1152-5 Device

means that the tools did a very poor job of placement (LUT mapping). These results are presented to give a perspective for FPGA architectures which inherently have a very fast interconnect mechanisms. Also, the clock period values give a measure of speed difference between ASIC and FPGA technologies for very wide issue processors.

6.4 Cumulative Experimental Results and Observations

Multi-cycling and pipelining the buses to minimize their effect on clock-period is common practice. In order to evaluate RF-to-RF communication for such architectures, we again use the framework discussed in Chapter 4. These establish a more

relative performance comparison between various architectures as clock-period is also taken into account. Results are shown in Figures 6.8, 6.9 and 6.10. In Figure 6.8, results are grouped under numbers of the form $< xB, yL >$. Here x denotes the number of buses and y denotes the bus latency. From the figure it is clear that performance comparable to point to point connected architectures is achieved by $< 8, 1 >$, $< 16, 1 >$ and $< 16, 2 >$. configuration if the cycle time is not considered. However, such a solution is very expensive due to the extremely high number of buses required. Also, as shown in Section 6.3, global buses with a single cycle latency lead to a decrease in the overall clock period by up to a factor of five leading to a drastic decrease in overall performance (Figure 6.10). Another interesting observation is that the performance varies monotonically with x/y i.e. the average data transferred per cycle.

While running the buses at frequencies slower than the processor is one alternative, another one is to explicitly pipeline the buses taking clock-period into account. To simplify scheduling, we assume a homogeneous configuration of buses. Results for these configurations are shown in Figure 6.9. Here results are again grouped under various numbers of the form $< xB, yS, zL >$. Where, x denotes the number of buses, y the bus stride and z latency of each path between two consecutive appearance of registers in the bus. Bus stride is the number of clusters skipped after which a pipeline register is introduced. Whereas for a stride of two, the inter-cluster interconnect does not appear in the critical path, for a stride of four it does. Therefore, for this stride we have shown results only for a latency of two. Another consideration is the divisibility of $N_{clusters}$ by y. As a consequence we have only been able to show results for four, eight and sixteen cluster configurations.

From Figure 6.9 it is clear that the performance of pipelined buses is severely

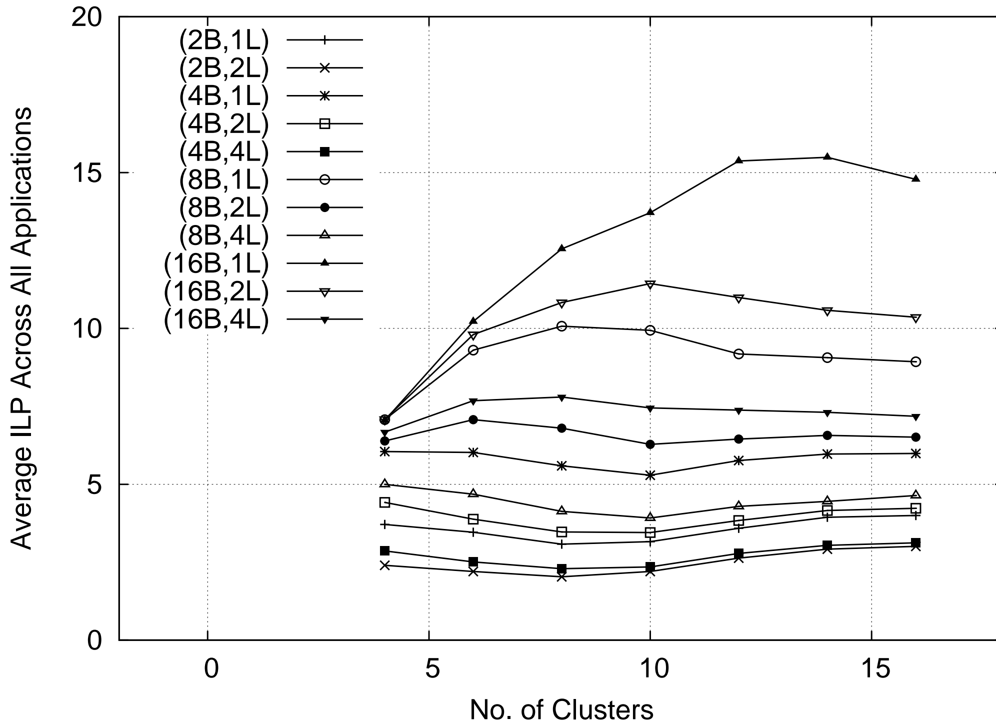


Figure 6.8: Bus Based Interconnects: Performance for Multi-cycle Configurations

limited. To analyze this we need to consider the factors which limit performance of clustered architectures. These are mainly related to communication bandwidth and communication latency. Bus based architectures invariably suffer from increased latencies as compared to direct communication. This is due to the fact that an additional transfer (copy) instruction needs to be scheduled if data has to be moved between clusters. Whereas introducing registers in buses allows the simultaneous usage of various *links* in buses for local communication, this does come with a penalty of one additional cycle latency. Correlating with results of multi-cycle buses in Figure 6.8, it is evident that a large bandwidth is required for local transfers, however this is advantageous if the transfer does not incur any additional penalty. However, this is not the case with configurations which we have shown here as that leads to penalty

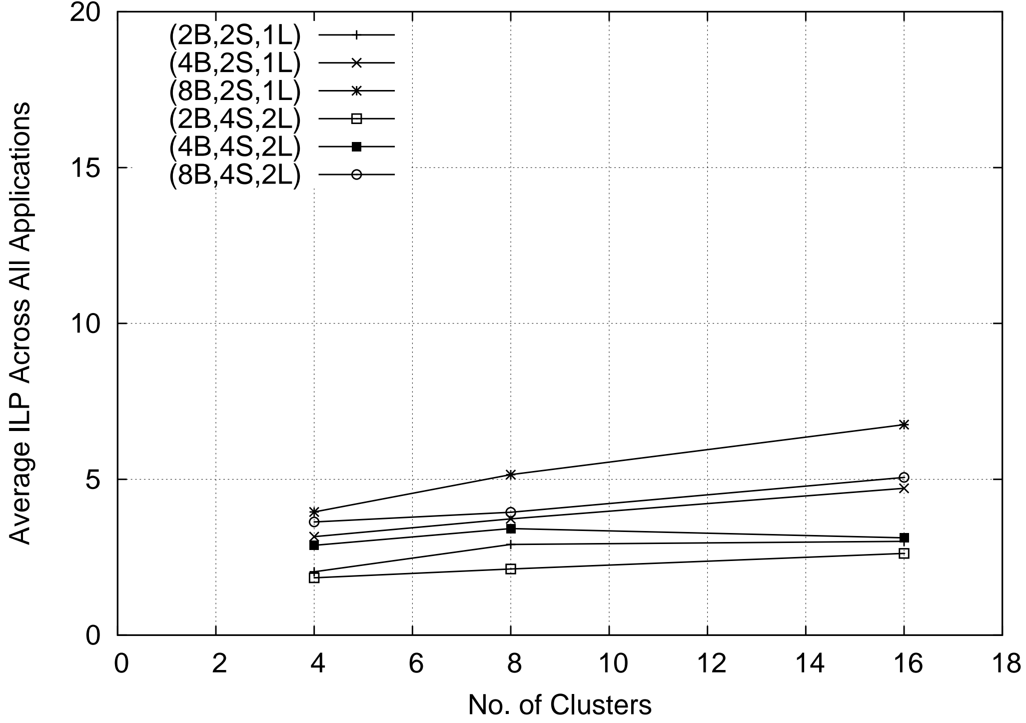


Figure 6.9: Average Variation in Performance for Pipelined Bus Configurations

in cycle time (see Table 6.2 for cycle time counts).

Using the clock-period results, it is possible to obtain a better picture of the final processor performance. Taking *Write Across - 1* architecture as the baseline case, we scale the reported ILPs using following formula:

$$ILP_{effective,\ x} = ILP_{original,\ x}\ *\ \frac{Clk\ Period_{WA.1}}{Clk\ Period_x}$$

Here x is the architecture type under consideration, $ILP_{original,\ x}$, is the original ILP reported in Chapter 5, $Clk\ Period_{WA.1}$ is the clock-period of *Write Across-1* type of architecture and $Clk\ Period_x$ is the clock period of architecture under consideration. This gives the relative performance of all other architectures vis-a-vis the *WA.1*.

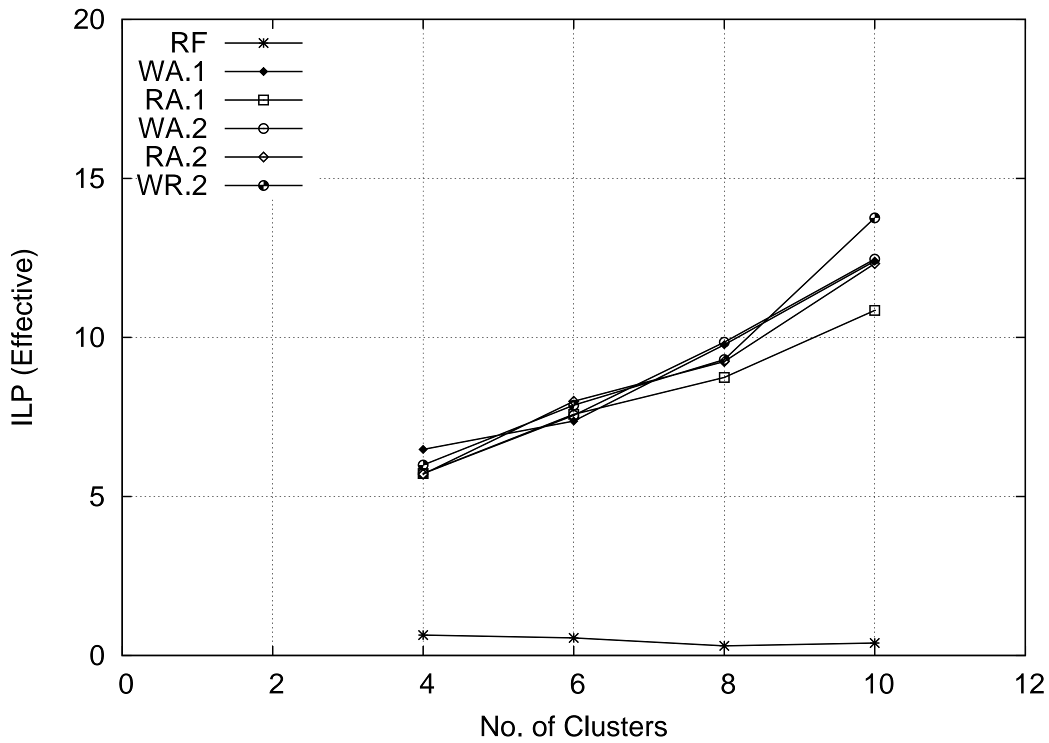


Figure 6.10: $ILP_{effective}$ for UMC 0.13μ ASIC Technology

Figure 6.10 shows effective average ILP. It is evident from this figure that global connectivity would annul an advantage which multiple buses may bring. The clock period scales too rapidly with global connectivity and no amount of increase in bandwidth can compensate for this effect. It is quite interesting to compare this figure with that obtained considering only number of cycles. However, it needs to be noted that there is considerable variation in performance if applications are considered individually.

6.5 Summary

In this chapter we have presented an exhaustive evaluation of bus based inter-cluster interconnects in clustered VLIW processor along with implementation characteristics of the various other point-to-point (direct) type ICNs. We have synthesized the processors with a variety of inter-cluster interconnects and obtained the achievable clock-period values. These values have in turn been used to arrive at realistic bus transfer latencies. The poor performance of such bus based interconnect is in sharp contrast to the direct-communication type of interconnects is established. These results become all the more significant if one takes into view the fact that almost all the research architectures and quite a few of the commercial ones have used bus based architectures though with a smaller number of clusters. The primary consideration for this seems to be a *simple* compiler. This implies greater focus on compiler technology for clustered VLIW processors.

Chapter 7

Conclusions

7.1 Summary

This research was taken up with an objective of developing a methodology for designing application specific high issue rate VLIW processors. We felt that it was essential to first establish that the domain of applications of our interest, namely, media applications, could indeed benefit from high issue rate or high ILP processors. Finding the results reported in the literature about available and achievable ILP rather confusing, we started with a classification of the different ILP measurement techniques as well as discussed their utility. While detecting ILP in media applications, we took a different approach. We bypassed typical compiler limitations in dealing with pointer aliasing by working on the instruction trace, thus focusing on what we termed as the *achievable-H* ILP. Further we made use of existing ILP enhancing compiler optimizations by obtaining the trace from a state of the art VLIW compiler infrastructure, namely, Trimaran. Results clearly showed that for a statically scheduled VLIW processor the average ILP in media applications could be as

much as 20 or higher.

Using the high achievable ILP as basis for exploring high issue-rate processors, we tackled the hitherto unsolved problem of inter-cluster ICN exploration. We started off with the classification of the inter-cluster ICN design-space. We gave an overview of the various inter-cluster ICNs which have been previously used in literature, the previous attempts at classification and finally our own classification. Our classification mechanism is able to capture almost all the architectures which have been previously reported in literature. We also presented a design-space exploration methodology which can be used for exploration.

We ran our design-space exploration methodology on a wide range of clustered VLIW architectures. The results conclusively showed that application dependent evaluation is critical to make the right choice of an interconnection mechanism. Also, it is quite evident from the results that all the architectures are inter-cluster communication bandwidth constrained. Once the communication delay starts dominating, the different inter-cluster transfer mechanisms fail to achieve performance close to Pure VLIW. This happens even though the supported concurrency has increases due to an increase in the number of available FUs. Also, the RF-to-RF type of architectures considered here, fail to perform to expected levels because even for adjacent clusters they have to gain access to global buses which are shared.

To present a comprehensive performance estimate as well as study the implementation characteristics of the various inter-cluster ICNs we carried out physical implementations of these processor. We developed parameterized VHDL models for processors having these interconnects, synthesized, placed and routed the designs to obtain the achievable clock-period values. We in turn used these clock-period results to arrive at realistic bus transfer latencies for RF-to-RF type architectures. We es-

tablished the poor performance of such bus based interconnect in sharp contrast to the direct-communication type of interconnects.

7.2 Main Contributions and Highlights of the Results

Following are the main contributions of this thesis:

- Definition and classification of available and achievable ILP
- Results obtained showed that the ILP in media applications could be as much as 20 or even higher.
- Tool chain for measuring *achievable-H* ILP in media applications
- Classification of design-space for inter-cluster communication architectures
- A clustering algorithm which can be used for performance estimation as well as code generation
- Methodology and tool-chain for inter-cluster architecture based design-space exploration
- Results for the design-space exploration showed that application dependent evaluation is critical to the final obtained performance. Also, the considered RF-to-RF architectures were severely performance constrained, even when the number of buses was increased.
- Parameterized VLIW design generator which can produce range of clustered VLIW architectures at logical and physical level

- Results from clock-period evaluation showed that there is very little variation in the clock-period for the point-to-point type architectures. Also, the RF-to-RF architectures, beyond two cluster configuration severely limit the cycle time. In these cases neither pipelining nor multi-cycle configuration configurations obtained results close to point-to-point type architectures.

7.3 Future Work

This thesis focused on homogeneous cluster configurations during design-space exploration. The work reported here could be extended for heterogeneous configurations. Code generation for clustered architectures is always a very complex task. The work reported in this thesis can be used as a basis for writing a generic code generator which can target the entire design-space.

Clustered architectures, bypass the requirement of an expensive RF by breaking it up into smaller chunks. However, high issue-rate processors, in turn need a sufficient number of ports on the data and instruction caches. They rely on the underlying memory architecture to supply the instruction as well as data. A possible extension could be the exploration of these memory architectures. Finally, this thesis focused on performance driven design-space exploration, a power driven methodology could be explored, wherein the architecture has constraints on both power consumption and performance requirements.

Bibliography

[Aditya et al., 1998] Aditya, S., Kathail, V., and Rau, B. R. (1998). Elcor's Machine Description System: Version 3.0. Technical Report HPL-1998-128, Hewlett-Packard Laboratories.

[Aditya et al., 1999] Aditya, S., Rau, B. R., and Kathail, V. (1999). Automatic architectural synthesis of VLIW and EPIC processors. In *International Symposium on System Synthesis*. November.

[Balakrishnan et al., 2004] Balakrishnan, M., Kumar, A., Panda, P. R., and Paul, K. (2004). Srijan: A Methodology for Synthesis of ASIP based Multiprocessor SoCs. Technical report, Department of Computer Science and Engineering, Indian Institute of Technology Delhi.

[Balasubramonian, 2004] Balasubramonian, R. (2004). Cluster Prefetch: Tolerating On-Chip Wire Delays in Clustered Microarchitectures. In *International Conference on Supercomputing*, pages 326–335.

[Banerjia et al., 1997] Banerjia, S., Havanki, W. A., and Conte, T. M. (1997). Treegion scheduling for highly parallel processors. In *European Conference on Parallel Processing*, pages 1074–1078.

[Bhargava and John, 2003] Bhargava, R. and John, L. K. (2003). Improving dynamic cluster assignment for clustered trace cache processors. In *Proceedings of the 30th annual international symposium on Computer architecture*, pages 264–274.

[Chang et al., 1991] Chang, P. P., Mahlke, S. A., Chen, W. Y., Warter, N. J., and Hwu, W. W. (1991). IMPACT: An architectural framework for multiple-instruction-issue processors. In *Proceedings of the 18th International Symposium on Computer Architecture (ISCA)*, volume 19, pages 266–275, New York, NY. ACM Press.

[Codina et al., 2001] Codina, J. M., Sanchez, J., and Gonzalez, A. (2001). A unified modulo scheduling and register allocation technique for clustered processors. In *International Conference on Parallel Architectures and Compilation Techniques (PACT 2001).*

[Conte et al., 1997] Conte, T. M., Dubey, P. K., Jennings, M. D., Lee, R. B., Peleg, A., Rathnam, S., Schlansker, M., Song, P., and Wolfe, A. (1997). Challenges to combining general-purpose and multimedia processors. In *IEEE Computer*.

[Cruz et al., 2000] Cruz, J.-L., Gonzalez, A., and Valero, M. (2000). Multiple-banked register file architectures. In *International Symposium on Computer Architecture (ISCA-2000).*

[Desoli, 1998] Desoli, G. (1998). Instruction Assignment for Clustered VLIW DSP Compilers: A New Approach. Technical Report HPL-98-13, Hewlett-Packard Laboratories.

[Doug Burger and James R. Goodman, 2004] Doug Burger and James R. Goodman (2004). Billion-Transistor Architectures: There and Back Again. *IEEE Computer*, 37:22–28.

[Ebcioglu et al., 1999] Ebcioglu, K., Altman, E., Sathaye, S., and Gschwind, M. (1999). Optimizations and oracle parallelism with dynamic translation. In *Proc. of the 32nd Annual International Symposium on Microarchitecture*, pages 284–295. ACM Press.

[Faraboschi et al., 2000] Faraboschi, P., Brown, G., Fisher, J. A., Desoli, G., and Homewood, F. M. O. (2000). Lx: A technology platform for customizable VLIW embedded processing. In *International Symposium on Computer Architecture (ISCA'2000)*. ACM Press.

[Faraboschi et al., 1998] Faraboschi, P., Desoli, G., and Fisher, J. A. (1998). Clustered Instruction-Level Parallel Processors. Technical Report HPL-98-204, Hewlett-Packard Laboratories.

[Fisher et al., 1996] Fisher, J. A., Faraboschi, P., and Desoli, G. (1996). Custom-fit processors: Letting applications define architectures. *IEEE Symposium on Microarchitectures.*

[Fritts and Mangione-Smith, 2002] Fritts, J. and Mangione-Smith, B. (2002). MediaBench II - Technology, Status, and Cooperation. In *Workshop on Media and Stream Processors, Istanbul, Turkey.*

[Fritts and Wolf, 2000] Fritts, J. and Wolf, W. (2000). Evaluation of static and dynamic scheduling for media processors. In *2nd Workshop on Media Processors and*

DSPs in conjunction with 33rd *Annual International Symposium on Microarchitecture*. ACM Press.

[Fritts et al., 1999] Fritts, J., Wu, Z., and Wolf, W. (1999). Parallel Media Processors for the Billion-Transistor Era. In *International Conference on Parallel Processing*, pages 354–362.

[Gaisler, 2001] Gaisler, J. (2001). LEON: A Sparc V8 Compliant Soft Uniprocessor Core. http://www.gaisler.com/leon.html.

[Gangwar et al., 2005] Gangwar, A., Balakrishnan, M., Panda, P. R., and Kumar, A. (2005). Evaluation of Bus Based Interconnect Mechanisms in Clustered VLIW Architectures. In *Proc. Design, Automation and Test in Europe (DATE-2005)*, pages 730–735.

[Gibert et al., 2003] Gibert, E., Sanchez, J., and Gonzalez, A. (2003). Flexible Compiler-Managed L0 Buffers for Clustered VLIW Processors. In *36th Annual International Symposium on Microarchitecture*, page 315.

[Glaskowsky, 1998] Glaskowsky, P. (1998). MAP1000 unfolds at Equator. *Microprocessor Report*, 12(16).

[Huang and Lilja, 1998] Huang, J. and Lilja, D. J. (1998). Improving Instruction-Level Parallelism by Exploiting Global Value Locality. Technical Report HPPC-98-12, High-Performance Parallel Computing Research Group, Univ. of Minnesota.

[Hwu et al., 1993] Hwu, W. W., Mahlke, A., S., Chen, W. Y., Chang, P. P., Warter, N. J., Bringmann, R. A., Ouellette, R. G., Hank, R. E., Kiyohara, T., Haab, G. E.,

Holm, J. G., and Lavery, D. M. (1993). The Superblock: An effective technique for VLIW and superscalar compilation. *J. Supercomputing.*

[Jacome and de Veciana, 2000] Jacome, M. and de Veciana, G. (2000). Design challenges for new application specific processors. In *IEEE Design and Test of Computers*, number 2, pages 40–50.

[Jacome et al., 2000] Jacome, M. F., de Veciana, G., and Lapinskii, V. (2000). Exploring performance tradeoffs for clustered VLIW ASIPs. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'2000).*

[Jouppi and Wall, 1989] Jouppi, N. P. and Wall, D. D. (1989). Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines. Technical Report 89/7, Western Research Laboratory.

[Kailas et al., 2001] Kailas, K., Ebcioglu, K., and Agrawala, A. K. (2001). CARS: A new code generation framework for clustered ILP processors. In *HPCA*, pages 133–144.

[Kathail et al., 2000] Kathail, V., Schlansker, M. S., and Rau, B. R. (2000). HPL-PD Architecture Specification Version 1.1. Technical Report HPL-1993-80, Hewlett-Packard Laboratories.

[Kozyrakis et al., 1997] Kozyrakis, C. E., Perissakis, S., Patterson, D., Anderson, T., Asanovic, K., Cardwell, N., Fromm, R., Golbus, J., Gribstad, B., Keeton, K., Thomas, R., Treuhaft, N., and Yelick, K. (1997). Scalable processors in the billion-transistor era: IRAM. *IEEE Computer*, (9):75–78.

[Lam and Wilson, 1992] Lam, M. S. and Wilson, R. P. (1992). Limits of control flow on parallelism. In *19th Annual International Symposium on Computer Architecture*, pages 46–57, Gold Coast, Australia.

[Lapinskii et al., 2001] Lapinskii, V., Jacome, M. F., and de Veciana, G. (2001). High quality operation binding for clustered VLIW datapaths. In *IEEE/ACM Design Automation Conference (DAC'2001).*

[Lee et al., 1997] Lee, C., Potkonjak, M., and Mangione-Smith, W. H. (1997). Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems. In *International Symposium on Microarchitecture*, pages 330–335.

[Lee et al., 2000] Lee, H.-H., Wu, Y., and Tyson, G. (2000). Quantifying instruction-level parallelism limits on an EPIC architecture. In *International Symposium on Performance Analysis of Systems and Software.*

[Lee et al., 1998] Lee, W., Barua, R., Frank, M., Srikrishna, D., Babb, J., Sarkar, V., and Amarasinghe, S. P. (1998). Space-time scheduling of instruction-level parallelism on a raw machine. In *Architectural Support for Programming Languages and Operating Systems*, pages 46–57.

[Leupers, 2000] Leupers, R. (2000). Instruction scheduling for clustered VLIW DSPs. In *IEEE PACT*, pages 291–300.

[Lewis et al., 1997] Lewis, D., Galloway, D., Ierssel, M., Rose, J., and Chow, P. (1997). The transmogrifier-2: A 1-million gate rapid prototyping system.

[Liao and Wolfe, 1997] Liao, H. and Wolfe, A. (1997). Available parallelism in video applications. In *Annual International Symposium on Microarchitecture*, pages 321–329.

[Mahlke et al., 1992] Mahlke, S. A., Lin, D. C., Chen, W. Y., Hank, R. E., and Bringmann, R. A. (1992). Effective compiler support for predicated execution using the hyperblock. In *25th Annual International Symposium on Microarchitecture.*

[Mattson et al., 2001] Mattson, P., Dally, W. J., Rixner, S., Kapasi, U. J., and Owens, J. D. (2001). Communication scheduling. In *Proceedings of the ninth international conference on Architectural support for programming languages and operating system*, pages 82–92.

[Middha et al., 2002] Middha, B., Raj, V., Gangwar, A., Kumar, A., Balakrishnan, M., and Ienne, P. (2002). A Trimaran Based Framework for Exploring the Design Space of VLIW ASIPs With Coarse Grain Functional Units. In *15th International Symposium on System Synthesis (ISSS'02).*

[Nakra et al., 1999] Nakra, T., Gupta, R., and Soffa, M. L. (1999). Value prediction in VLIW machines. In *International Symposium on Computer Architecture*, pages 258–269.

[Nicolau and Fisher, 1981] Nicolau, A. and Fisher, J. A. (1981). Using an oracle to measure potential parallelism in single instruction stream programs. In *14th Annual Workshop on Microprogramming*, pages 171–182.

[Ozer et al., 1998] Ozer, E., Banerjia, S., and Conte, T. M. (1998). Unified assign and schedule: A new approach to scheduling for clustered register file microarchitectures. In *International Symposium on Microarchitecture*, pages 308–315.

[Patt et al., 1997] Patt, Y. N., Patel, S. J., Evers, M., Friendly, D. H., and Stark, J. (1997). One billion transistors one uniprocessor one chip. In *IEEE Computer*.

[Rau and Schlansker, 2001] Rau, B. R. and Schlansker, M. S. (2001). Embedded computer architecture and automation. *IEEE Computer*, 34(4):75–83.

[Rixner et al., 2000] Rixner, S., Dally, W. J., Khailany, B., Mattson, P. R., Kapasi, U. J., and Owens, J. D. (2000). Register organization for media processing. In *Proceedings of 6th International Symposium on High Performance Computer Architecture*, pages 375–386.

[Sanchez et al., 2002] Sanchez, J., Gibert, E., and Gonzalez, A. (2002). An interleaved cache clustered VLIW processor. In *ACM International Conference on Supercomputing (ICS'2002)*.

[Sanchez and Gonzalez, 2000] Sanchez, J. and Gonzalez, A. (2000). Instruction scheduling for clustered VLIW architectures. In *International Symposium on System Synthesis (ISSS'2000)*.

[Schlansker and Rau, 2000] Schlansker, M. S. and Rau, B. R. (2000). EPIC: An Architecture for Instruction-Level Parallel Processors. Technical Report HPL-1999-111, Hewlett-Packard Laboratories.

[Siroyan, 2002] Siroyan (2002). http://www.siroyan.com.

[Smith et al., 1989] Smith, M. D., Johnson, M., and Horowitz, M. A. (1989). Limits on multiple instruction issue. In *Proceedings of the 3rd International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS)*, volume 24, pages 290–302, New York, NY.

[Smits, 2001] Smits, J. E. (2001). Instruction-level distributed processing. *IEEE Computer*, 34(4):59–65.

[Song, 1998] Song, P. (1998). Demystifying EPIC and IA-64. *Microprocessor Report*.

[Stefanovic and Martonosi, 2001] Stefanovic, D. and Martonosi, M. (2001). Limits and graph structure of available instruction-level parallelism (research note). *Lecture Notes in Computer Science*, 1900:1018–1022.

[Talla et al., 2000] Talla, D., John, L., Lapinskii, V., and Evans, B. (2000). Evaluating Signal Processing and Multimedia Applications on SIMD, VLIW and Superscalar Architectures. In *International Conference on Computer Design (ICCD)*, pages 163–174.

[Tencilica, 2003] Tencilica (2003). http://www.tensilica.com.

[Terechko et al., 2003] Terechko, A., Thenaff, E. L., Garg, M., van Eijndhoven, J., and Corporaal, H. (2003). Inter-Cluster Communication Models for Clustered VLIW Processors. In *9th International Symposium on High Performance Computer Architecture, Anaheim, California*, pages 298–309.

[Texas Instruments, 2000] Texas Instruments (2000). TMS3206000 CPU and Instruction Set Reference Guide.

[Trimaran Consortium, 1998] Trimaran Consortium (1998). The Trimaran Compiler Infrastructure, http://www.trimaran.org.

[Tseng and Asanovi, 2003] Tseng, J. H. and Asanovi, K. (2003). Banked multiported register files for high-frequency superscalar microprocessors. In *International Symposium on Computer Architecture (ISCA-2003)*, pages 62–71.

[Zalamea et al., 2001] Zalamea, J., Llosa, J., Ayguade, E., and Valero, M. (2001). Modulo scheduling with integrated register spilling for clustered VLIW architectures. In *Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitectur*, pages 160–169.

[Zivojinovic et al., 1994] Zivojinovic, V., Velarde, J. M., Schlager, C., and Meyr, H. (1994). DSPStone – A DSP-oriented Benchmarking methodology. In *International Conference on Signal Processing Application Technology, Dallas, TX*, pages 715–720.

Technical Biography of Author

Anup Gangwar received a B. E. in Electronics and Telecommunication Engineering from B.I.T. Bhilai (India) in 1998, and an M. Tech. in VLSI Design Tools and Technology (VDTT) from Indian Institute of Technology Delhi (India) in 2000. From January 2001 to November 2004, he worked towards his Ph.D. on "A Methodology for Exploring Communication Architectures of Clustered VLIW Processors". Since November 2004, he has been employed at Calypto Design Systems (I) Pvt. Ltd., Noida, as a Senior Member of Technical Staff.

Refereed Publications

1. Anup Gangwar, Jos T. J. van Eijndhoven, M. Balakrishnan and Anshul Kumar, "Multi-Processor Multi-Tasking Performance Data Measurement and Visualization", Nat. Lab. (Philips Research Laboratories Eindhoven) Technical Note 2001/9, Jan 2001, Eindhoven, The Netherlands

2. Bhuvan Middha, Varun Raj, Anup Gangwar, Anshul Kumar, M. Balakrishnan and Paole Ienne, "A Trimaran Based Framework for Exploring Design Space of VLIW ASIPs With Coarse Grain Functional Units", 15th IEEE/ACM International Symposium on System Synthesis (ISSS'02), Oct 2002, Kyoto, Japan

3. Amarjeet Singh, Amit Chhabra, Anup Gangwar, Basant K. Dwivedi, M. Balakrishnan and Anshul Kumar, "SoC Synthesis With Automatic Interface Gen-

eration", 16th IEEE/ACM International Conference on VLSI Design (VLSI-2003), Jan 2003, New Delhi, India

4. Anup Gangwar, M. Balakrishnan and Anshul Kumar. "Impact of Inter-cluster Communication Mechanisms on ILP in Clustered VLIW Architectures", 2nd Workshop on Application Specific Processors (WASP-2), in conjuction with 36th IEEE/ACM Annual International Symposium on Microarchitecture (MICRO-36), Dec 2003, San Diego, USA

5. Anup Gangwar, M. Balakrishnan, Preeti Ranjan Panda and Anshul Kumar. "Evaluation of Bus Based Interconnect Mechanisms in Clustered VLIW Architectures", IEEE/ACM Design Automation and Test in Europe (DATE05), Mar 2005, Munich, Germany.

Refereed Presentations

1. Anup Gangwar, M. Balakrishnan and Anshul Kumar. "Customizing Clustered VLIW Architectures with Focus on Interconnects and Functional Units", SIGDA Ph.D. Forum at 41st IEEE/ACM Design Automation Conference (DAC-41), Jun 2004, San Diego, USA.

2. Ankit Mathur, Mayank Agarwal, Soumyadeb Mitra, Anup Gangwar, M. Balakrishnan, and Subhashis Banerjee. "SMPS: An FPGA-based Prototyping Environment for Multiprocessor Embedded Systems", IEEE/ACM Thirteenth International Symposium on Field Programmable Gate Arrays (FPGA-2005), Feb 2005, Monterey, USA.