國 立 交 通 大 學

資訊工程學系

碩 士 論 文

應用在叢集式超長指令集處理器的叢集間低成本通訊方法

A Low-Cost Inter-cluster Communication Scheme
for Clustered VLIW Processors

研 究 生: 吳 智 斌
指導教授: 鍾崇斌教授

中華民國九十三年六月

應用在叢集式超長指令集處理器的叢集間低成本通訊方法

A Low-Cost Inter-cluster Communication Scheme for Clustered VLIW Processors

研 究 生: 吳 智 斌　　Student: Chih-pin Wu
指導教授: 鍾崇斌博士　　Adviser: Dr. Chung-Ping Chung

國 立 交 通 大 學
資 訊 工 程 學 系
碩 士 論 文

A Thesis
Submitted to Department of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer Science and Information Engineering

June, 2004

Hsinchu, Taiwan, Republic of China

中華民國九十三年六月

應用在叢集式超長指令集處理器的叢集間低成本通訊方法

學生：吳智斌　　指導教授：鍾崇斌博士

國立交通大學資訊工程學系碩士班

摘　　要

在超長指令集處理器上，叢集化是一個常被利用來幫助實作 Register File 的一個方法。若沒有叢集化，當加入了大量的 Function Units 到超長指令集處理器上後, 對 Register File 的讀寫埠需求會使得 Register File 的存取延遲、面積和耗電成為 一大問題。然而即使是使用了叢集化的技巧，現存的幾個叢集間通訊方法仍需要依賴額外的讀寫埠讓各叢集交換資料。本文提出了一個不需要額外的讀寫埠也可以達到叢集間通訊的方法；我們以將暫存器實作成讀寫埠分別面對不同的叢集的方式來達成叢集間通訊。由於不用增加額外的讀寫埠和投資新的硬體資源，這種方法可以在存取延遲、面積和耗電方面上較現有的方法具有優勢，並讓叢集式超長指令集處理器可以用一個較低成本的方法達到叢 集間通訊的目的。

A Low-Cost Inter-cluster Communication Scheme for Clustered VLIW Processors

Student: Chih-pin Wu Adviser: Dr. Chung-Ping Chung

Department of Computer Science and Information Engineering
National Chiao Tung University

ABSTRACT

Clustering is a well-known technique to improve the implementation of register file on VLIW processors. Without clustering, high demands on read/write ports will bring many issues on power dissipation, area and delay, and thus makes the hardware scalability problematic. However, most inter-cluster communication models rely on extra read/write ports to access register values between clusters. The objective of the thesis is to propose an inter-cluster communication model which demands no extra read/write ports but using a kind of *single-way* special register. We evaluated the performance by hand-optimized codes and code rewriting generation approach. Simulated results showed that for several generic computation kernels, only a few extra cycles and special registers will be needed, but sacrificing no extra delay on register file access. Thus improvements on execution time can be achieved. The design will also benefit from less power dissipation and fewer silicon area, making the approach an efficient and economic communication scheme for clustered VLIW processors.

Contents

List of Figures

List of Tables

Acknowledgements

The author wishes to acknowledge the valuable suggestions from Professor Chung-Ping Chung, Professor Jean Jyh-Jiun Shann and the constructive comments from all the dears in Computer System Laboratory.

Chapter 1

Introduction

The VLIW[1] processors have been chosen by many hardware architects as an effective approach for applications demanding large ILP[2]. However the large number of function units will bring some impact on high read/write ports on register files. And clustering is proposed to minimize the impact.[1]

For clustered processors, the inter-cluster communication models is so important because it is the key for function units located in different clusters to work together. Most of existed inter-cluster communication models rely on extra read and/or write ports to achieve this objective. We propose a low-cost inter-cluster communication scheme, which requiring no extra read or write ports in this thesis.

In this chapter, we first state the research observations from VLIW processors to clustering. And then the motivation will be presented. Second, we explain the basic concept of the proposed design. Finally, the organization of the thesis is introduced.

[1]Very Long Instruction Word

[2]Instruction-Level Parallelism

1.1 VLIW Processors

VLIW nowadays is a well-known approach while processor designers seeking for large ILP. It gets several advantages on instruction fetching, decoding and scheduling issues compared to the superscalar processor, due to the prevention of complex and large logic. The hardware scalability of VLIW is also excellent because that adding a new function unit will not modify the existed logic a lot but some duplicated logic. However it is programmer's and/or compilers' responsibility to find out the parallelism of programs, making it important of skilled programmers to write a highly parallelized programs for VLIW machines. If the executed VLIW codes is not dense enough to take use of the large parallelism, the VLIW machine is just a power-inefficient machine because that the unused function units will be just idling.

1.2 Clustering VLIW

We've presented the advantages of VLIW approach, including the hardware scalability. However the claimed hardware scalability on VLIW machines is problematic. We can notice the nature that as the number of function units increases, the more demands on number of register ports will be. But high port number of register file will make it difficult to implement. The delay, area and power dissipation will increase dramatically. There has been some research[2] showed that as ALU number grows to N times more, area and power dissipation will grow as N^2 times, and delay will grow as N times. The large demands on port number will make the stage related to register file access much longer than other stages, thus register file access latency will start to dominate the processor frequency. Thus the more function units the VLIW has, the slower it is. Then the performance brought by VLIW's large ILP will be neutralized by the slow executing frequency.

Thus clustering is proposed to improve the implementation of register file and min-

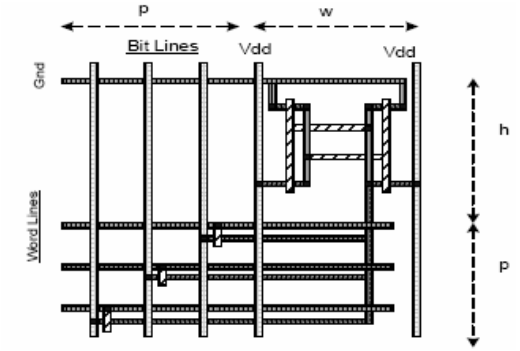

Figure 1.1: A classical register cell

We can notice that more ports will lead to longer wire length, more fan-out and larger area.

imize the impact of high read/write ports demands. With clustering, function units now can only access local registers, leading to the reduction of register file port number. Here we must notice that clustering is different from SMP[3]. All function units of clustered VLIW are sharing the same clock signal, and stalls due to any function units will make all other function units located in all clusters stalled as the nature of VLIW, while this is not necessary for SMP machines.

The data located in different clusters may need to be operated together. In order to make all clusters working corporately, the data exchanges between clusters is necessary. Thus copies among clusters are taken as a basic idea for the purpose.

According to the difference of inter-cluster communication schemes in a higher level point of view, we can basically categorize them into several types:

- Main Memory
  use memory(i.e., load and store instructions) to exchange data among clusters
- System Bus
  if system bus supports point-to-point transactions, clusters can communicate with

[3]Symmetric Multi-Processing

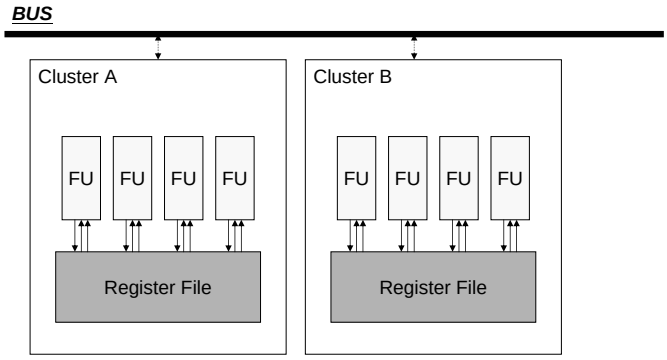

Figure 1.2: Clustered processor

each other via bus transactions

- Register File

  providing a special interconnect and bypass network among the register files to provide communication mechanism

Both main memory and system bus approaches are simpler to implement, and they don't need to modify the core logic of the cluster. However, due to the modern VLSI[4] technology, the speed gap between core logic and external I/O causes that both of the approaches inefficient. And due to the existence of bus contention and memory coherence, both of the approaches must be asynchronous, while asynchoronous may be a fatal issue to a static scheduling machine. Asynchornous means that the operation is not cycle-accurate predictable, and when the operation is completed is unknown. For most VLIW machines, the cycle-accurate scheduling is the key to the execution efficiency and code density. These operations will lead to stalls with unpredicted length, thus the paralleism may be compromised due to the stalls. Using main memory as inter-cluster communication also brings some side effect on cache subsystem besides the latency problem. So most architects thought it clever to choose a register file approach rather than others for inter-cluster communication.

[4]Very Large Scale Integration

In the following section, we will discuss about several existed researches on register file-based inter-cluster communication approach.

1.2.1 Register File-based Inter-cluster Communication

Most inter-cluster communication schemes are based on register files because that it's simplest for programmer to use and no major modification to ISA[5] is needed. Inter-cluster communication schemes based on register file usually provide partial accessbility to remote register file via instruction operands, results or dedicated copies. Types of instructions which are able to access remote register file may be restricted. The instruction which access remote register file may take longer latency than ordinary instructions. Also the available inter-cluster communication quantity per issue frame may be also constrainted. The difference and characteristics introduces the various researches in this field.

1.3 Related Work

In the following sections, we presented several existed researches on inter-cluster communication models based on the Philips' publication[3].

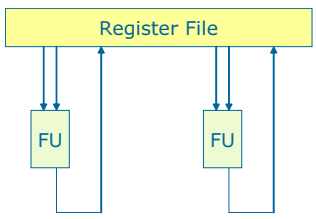

Figure 1.3: Unicluster

[5]Instruction Set Architecture

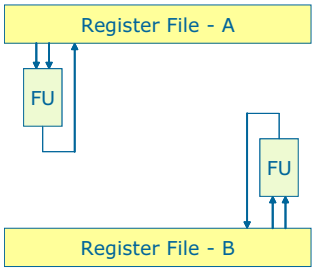

Figure 1.4: Clustered

The unicluster and clustered model are presented as references to related works.

1.3.1 Copy Operation

Inter-cluster communication in this model is specified as copy operations in regular VLIW issue slots. In the operand read stage of a copy operation the value is read from the local RF, passed through the bypass network, and then sent to remote register file. It can be issued in any slots as long as corresponding function units can read value from local register file. But how many copy operations can be issued per clock is constrained.

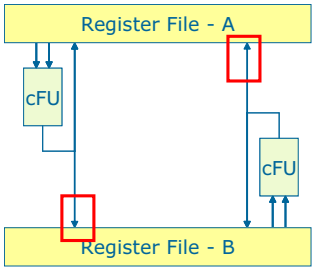

Figure 1.5: Copy Operation

Extra write ports are necessary for this inter-cluster communication model.

1.3.2 Dedicated Issue Slot

In this model inter-cluster communication is executed in extra dedicated issue slots of the VLIW instruction. The operation is restricted to copies. It's like copy operation model but needed to be in the dedicated slot.

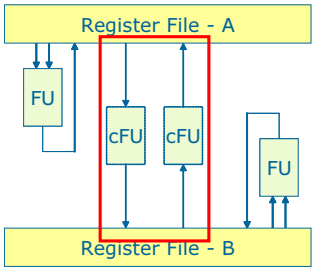

Figure 1.6: Dedicated Issue Slot

Due to the extra dedicated issue slots, extra read and write ports will be needed in this model.

1.3.3 Extended Operands

The source operands in this model are extended with cluster identification, so that the source operands can be remote registers. This model allows using a value from a remote register file without storing it in the local register file, which lessens the register pressure.

For the extended source operands, extra read ports will be needed to serve remote cluster accesses.

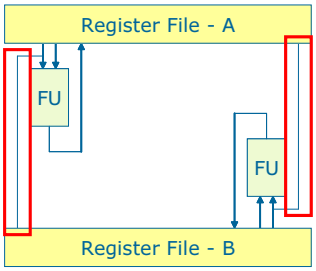

Figure 1.7: Extended Operands

1.3.4 Extended Results

In this model, destination operands are extended with cluster identification, so that the operation's result can be stored remotely. Different from copy operation, this model implement inter-cluster moves rather than copies.

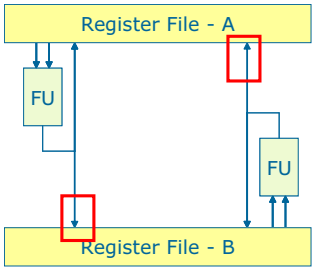

Figure 1.8: Extended Results

Extra write ports are necessary for the the remote destination operands.

1.3.5 Share Registers

This model can use shared register addresses to communicate between clusters. The registers corresponding to the shared addresses can be both read and written in all clusters. By reducing the shared resources to partial addresses of whole register file, this model is prevented to be a unicluster model but partially unified.

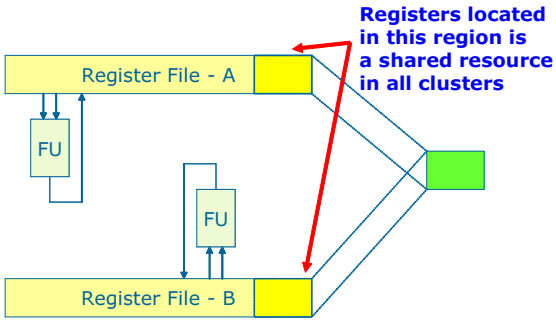

Figure 1.9: Shared Registers

For the shared registers, read and write ports corresponding to each cluster will be the necessary extra hardware.

1.4 Observations and Motivations

Existed researches on inter-cluster communication rely on extra ports and additional bypass networks for exchanging data among clusters, and the increase of register file ports will affect the implementation of register file though. As register file ports increases, access to register file will be slow, the power and area of register file will grows dramatically. Thus most machines constraint the inter-cluster communication per issue frame to reduce the impact on register file ports. However that means there are only some instructions located in the same issue frame are able to access the remote data, and this will affect the performance somehow. But it is impossible to provide full accessibility to the remote data, or the processor will be a unicluster(non-clustered) architecture, which suffers from the access latency, power dissipation and area impact the most.

The objective of the thesis is to exploit a new method of inter-cluster communication, which demands on no extra ports on register file, and thus no extra access latency to

register file will be increased. Detailed design will be introduced in next chapter.

1.5 Organization of the Thesis

The thesis is organized as follows. Chapter 2 introduce design of proposed inter-cluster communication scheme. Works related to inter-cluster communication are also compared in several points of view in the chapter. And in Chapter 3, we evaluated the performance of the baseline machine and proposed design with hand-optimized codes and code rewriting approach. Finally the conclusion and future works are discussed in Chpater 4.

Chapter 2

Design

In this chapter, details of the proposed design will be presented.

2.1 Basic Concepts

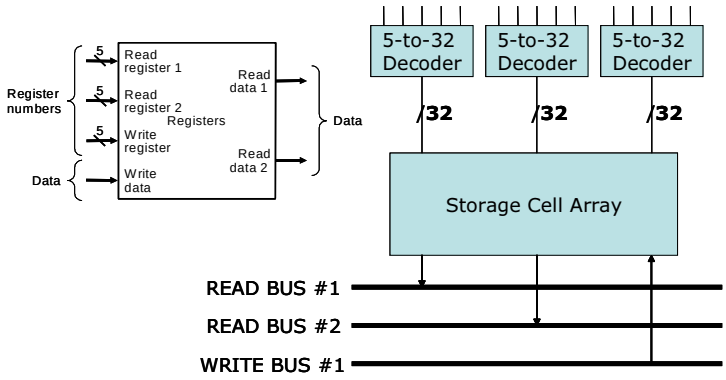

Figure 2.1: Classical Register File

By observing the classical register file, we can notice that the decoder and bus can be rewired to different storage cells. With a little little modification, we can rewire the register file to make its read ports and write ports connect to FU which belongs to different clusters. To be short, some registers can be read by function units of cluster #A and be written by those of cluster #B. Thus communication between clusters can

be achieved by these special register rather than extra ports.

2.2 Design of Register File

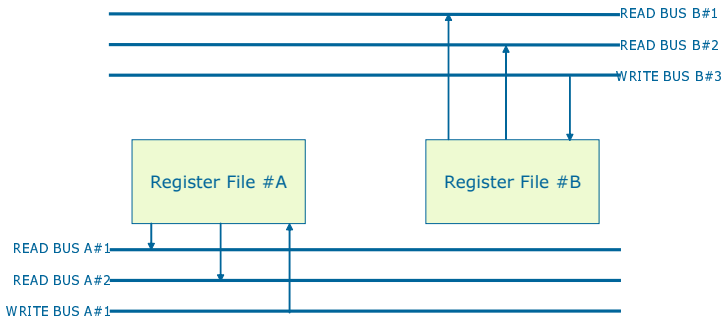

Figure 2.2: Original Register File of Clustered Processors

The figure shown above is a generic organization view of clustered processors. We assume that each register file owns 16 registers. The 2-read and 1-write bus is connected to a normal function unit, say ALU. And we can notice it is a clustered processor because the function units can only access its local register file.

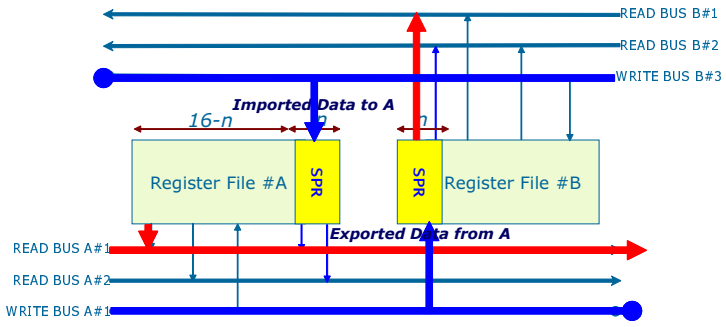

Figure 2.3: Modified Register File

With rewiring some registers, we can notice that some registers are used to import

data to cluster #A and some are used to export data from cluster #A. The import and export are defined in the view of cluster #A. The *import* means that cluster #A can access data from remote cluster, while export are used by cluster #A to output data to remote cluster.

Registers for importing are done by rewiring their write ports toward cluster #B, but remains their read ports same, while registers for exporting are done in the same concept. Rewiring the write ports of registers to cluster #B makes that these registers be able to written by any function units located in cluster #B. In simpler words, their writing is driven by cluster #B but reading is still driven by cluster #A.

The rewiring includes redirecting the decoder's select(enable) signal and register cell output wires. However the rewiring is not adding new logic, thus keep the register file fast, smaller and power-efficient as the original clustered register file is.

The number of register being rewired must be symmetric in both clusters to keep the register index range legal and same as before.

The registers for importing values act as channels for cluster #B to transfer data to cluster #A, then all function units at cluster #A can read these values from the registers. The export registers act as virtual channels for cluster #A to transfer data to cluster #B, and all function units belongs to cluster #B can read these values as well. So that the inter-cluster communications between clusters are done by transferring value into these special registers.

In the above case we replaced n register of total 16 with SPR[1], and the number n is decided by hardware designer. More SPR brings more channels for inter-cluster com-

[1]Special Purpose Register, those marked as import and export registers

munication. However due to the replacement, more SPR equals to fewer GPR[2], and it will certainly increase the register pressure. Deciding a proper number of SPR is necessary to minimize the impact of increased register pressure and satisfy the ICC demands.

Because the inter-cluster communication is done by these special purpose registers, no extra read or write ports are needed. And these SPR is implemented by *redirecting* their ports to another cluster, data exchanges between clusters can be done without extra hardware. Every FU can read or write (depends on where the FU is located) these SPR without constrained, that is there is no restriction on how many FU can access these SPR. Thus multiple copies or data exchanges between cluster per cycle is easily done.

2.3 Hardware Characteristics Comparison

In this section, we compare the proposed design with other ICC models, and examined them together to see the difference of hardware characteristics.

We used following arguments in the comparison table:

B_R : Reading Accessibility

B_W : Writing Accessibility

C : Cluster Number

Unicluster model doesn't apply the clustering technique. Thus as function unit number grows, the register file must prepare corresponding number of ports to satisfy the function units. And most function units are 2-read and 1-write style, so three times

[2]General Purpose Register

Table 2.1: Hardware Characteristics Comparison

Model	Extra Read Port	Extra Write Port	Extra VLIW Instruction Field	Access Latency
Unicluster	Not Applicable		0	slowest
Copy Operation	0	B_W	0	slower
Dedicated slots	B_R	B_W	0	slower
Extended results	0	B_W	$\log(C)$	slower
Extended operands	B_R	0	$\log(C)$	slower
Share Registers	(C-1) times		0	much slower
Proposed Design	0	0	0	faster

ports of function units are needed.

Copy operation model uses normal function units to do copy operation, but it needs to write to remote cluster to copy the value. Thus one more write port for each cluster per access is needed.

Dedicated slots model is similar to copy operation models, however it uses dedicated slot so that one extra read and on extra write ports are needed for each cluster per access.

Extended results model needs one extra write port for each cluster per access because of the extended destination operand field, while extended operands model need one extra read port for the similar reason.

Shared registers model uses some uniclustered registers. For those shared registers, the port demand is similar to unicluster model. They must have enough ports to serve every function units located in every cluster.

Above models use extra ports to communicate between clusters, so the access latency is slowed depends on the number of extra port number. Proposed design relies no extra ports to accomplish the inter-cluster communication, and the access latency will not be slowed at all.

2.4 Usage Limitation Comparison

In this section, the differences between these models are re-examined from the programmer's(user's) view.

Table 2.2: Usage Limitation Comparison

Model	writing results for remote cluster	writing results for remote and local clusters	ICC per cycle
Unicluster	Not Applicable		
Copy Operation	Extra Copy	Extra Copy	Extra Port Number
Dedicated slots	Extra Copy	Extra Copy	Extra Port Number
Extended results	$\surd$	Extra Copy	Extra Port Number
Extended operands	Not Applicable		Extra Port Number
Share Registers	$\surd$	$\surd$	SPR Amount
Proposed Design	$\surd$	Extra Copy	SPR Amount

The table is arranged according to an operation and its succeeding dependent operation's relationship. If one operation generated a value which will be used only by the remote cluster, the accessibility is shown in the 2nd column. *Not Applicable* means that it is not necessary or not available within the model. A $\surd$ means that it can be solved by the model without any extra instructions or cycles. *Extra Copy* means that it must be done by adding another copy operation, which will consume one slot and maybe one more cycle.

The *Not Applicable* shown in Extended operands states that the model cannot write any value to remote cluster. However the remote cluster has the ability to access the local register, thus it's not necessary to send data to remote cluster and remote cluster just access it when the remote cluster needs the data.

According to this table, we can notice that extended operands model is advantaged compared to other models. However even with extended operands model, the inter-cluster communication is also constrained by the extra port number. If more than prepared extra port number of function units requesting ICC[3] at the same time, some of them will be postponed to later cycles due to that no available ports can be used to serve all function units. The shared register model and proposed design will not be restricted in such condition, and all function units can access those special purpose registers at the same time(cycle). These two models are only constrained when one cluster wants to send data more than SPR numbers, for example if there are 2 SPR and cluster #A wants to send 3 data to cluster #B. In such cases, some of the communication requests must be postponed and waiting for the release of the SPR.

[3]Inter-Cluster Communication

2.5 Summary

In this chapter, we introduced the details of proposed design and how the modified register file organization works as a inter-cluster communication scheme. And in following chapter, we will evaluate its performance with the baseline machine.

Chapter 3

Evaluation

This chapter evaluates the performance between proposed model and the baseline machine. We evaluated these two model with hand-optimized codes and an automatic code rewriting method. Both experiments shows that proposed design performs better than baseline machine on several computation kernels.

3.1 Evaluation Approach

The evaluation part is proceeded in two approaches, hand-optimized codes and code rewriting.

The hand-optimized codes tries to maximize the code performance by human beings, and can be seen as the optimum. However the optimization is extremely time-consuming and cannot be easily repeated on each benchmark.

The code rewriting is a code transformation scheme applied on codes for other models. With peep-hole approach, code rewriting *patches* the original codes with applying proposed inter-cluster communication. In other words, instructions related to ICC in original codes will be replaced by instructions which adopting proposed ICC. Thus codes after rewriting can be executed on proposed model and evaluated the performance

difference.

3.2 Introduction to the Baseline Machine

We choose TI TMS320C6200[4] as our baseline machine. It's a 8-way VLIW, 2-cluster commercial digital signal processor and very popular in the commercial market. Based on the extended operands model, the manufacturer claims its overwhelming power on digital signal processing applications. It is able to perform one inter-cluster communication per cluster per cycle on. The use of ICC will not induce any stalls. However on its advanced version TMS320C6400, whose architecture is mostly similar to C6200 but running at higher frequency(1GHz versus 200MHz) will penalty from one stall cycle when ICC is used. We can notice that the register file access will become critical path when the processor need to run at higher speed.

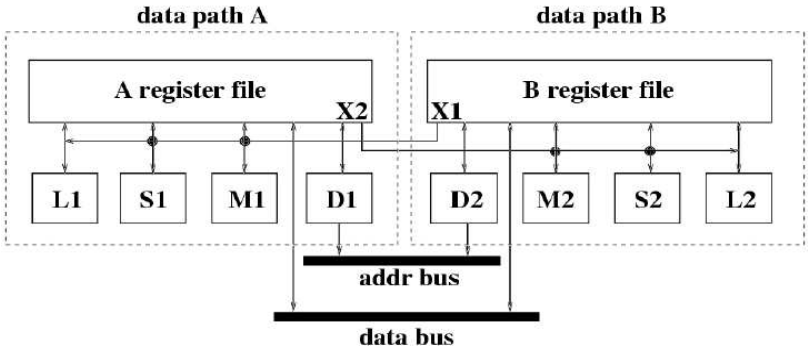

Figure 3.1: TMS320C6200: Organization Overview

We modified its ICC model from extended operands model to proposed design, however deciding the number of SPR is a major issue. We decided the number in hand-optimized codes section according to the ICC demands. In the code rewriting section, due to the original register allocation result, we didn't replace any of total 32 registers into SPR, but add the SPR besides the original registers. However the costs of added registers has taken into concern, and the costs includes time latency, silicon area and

power dissipation. By taking these costs into concern we make the comparison more fair.

3.3 Hand-Optimized Codes

In this section, we examined the finite impulse response filter performance on baseline and modified machines. Finite impulse response filter (FIR Filter) is globally used in various applications, and is also a good measure for compare performance between processors.

The optimized FIR codes for baseline machine are available from TI's official web site, and the FIR codes optimized for proposed design is presented in appendix.

Both versions of the FIR filter performs the same on cycle count: M*(N+8)/2+6 cycles, while M stands for outer loop count and N stands for inner loop count. And the whole 16 registers per cluster is partitioned into 11 GPRs plus 5 SPRs for the machine adopting proposed design. Under the same performance on cycle count, proposed model performs better on time latency, area and power dissipation thanks for the reduction of one read port originally for the ICC.

This shows that on FIR filter applications, proposed design can performs as well as extended operands model, and even better.

3.4 Code Rewriting

The goal of code rewriting is to transform codes using other ICC models into codes using proposed models. The code rewriting should be an automatic and systematic approach. Programmers can use this approach to transform their existed codes to migrate

to other ICC models to seek better performance rapidly and easily.

In this section, we use TI TMS320C6X C/C++ Compiler ver4.32[5] to generate codes for baseline machine. Then we use a modified tool based on a TMS320C6200 instruction set simulator[6] to accomplish code rewriting and evaluate the cycle count.

The experiment examines generated transformed codes performance on

- Additional SPR Number and
- Extra Execution Time

In order to keep original register allocation result, the inter-cluster communication is done by new SPR, which is additional hardware. Adding these new registers and reduce the ports for ICC in extended operands model make access latency change. The *patching code* might also increase some extra execution cycles, and both of the access latency and extra cycles will be considered together as a time performance metric.

Following benchmarks have been chosen to be evaluated, and they are all computation kernels because that they have higher parallelism and frequently-utilizing the inter-cluster communication.

- Block Move
- FIR Filter
- Complex FIR
- IIR Filter
- Vector Sum of Squares
- Weighted Vector Sum
- DCT

- IDCT

3.4.1 DEF-USE Model

Definition(DEF) and Use model is globally used in the compilation techniques. *DEF* means the generation of data, equals to the destination operand of operation. *USE* means the references of data, equals to source operands of operation. By converting all ICC-related instructions into DEF-USE models as a intermediate form, we can analyze and trace the inter-cluster communications, and then transform them to other ICC models easily.

3.4.2 Comparing to Other Inter-cluster Communication Schemes with DEF-USE Model

Table 3.1: Comparison between different models

	USE	DEF
Unicluster	Local	Local
Copy Operation and Dedicated Issue Slots	Local	Local
Extended Operands	Local	Local/Remote
Extended Results	Local/Remote	Local
Shared Registers	Local/Remote	Local/Remote
Proposed Design	Local/Remote	Local

We can categorize DEF and USE into local and remote two different kinds. A local DEF acts just as normal operation, writing a value to local register file, while a local USE acts in the same concept of reading a value from local register file. A remote DEF means that it's writing a destination to remote cluster like extended results model, while

A remote USE is that referencing a value from remote cluster like extended operands model. Proposed design can write a value to export registers which can be read from remote cluster, thus it supports remote DEF.

By observing the difference between models in DEF/USE model, we can convert a remote DEF into remote USE or in the opposite way. By converting between remote DEF and remote USE we can accomplish the goal of rewriting the codes for other models.

3.4.3 Code Rewriting Principles

The simulation is based on Vinodh Cuppu's cycle accurate simulator[6], with some modification to fit experiment requirement. Both of code rewriting and cycle count evaluation are done in the modified simulator.

The code rewriting is based on the following principles

1. Keeping Correctness,
2. Minimizing Extra Execution Cycles, and
3. Using Least Special Registers

In order to keep correctness, we need to insert copies before remote use, to make it copied to import register to be accessible by local function units. For minimizing extra cycles, we tries to insert the copy in the non-full VLIW issue frames as hard as possible. Only when there is no empty issue slots between DEF and USE we will insert a new issue frame for the copy.

The details and major issues of designing a code-rewriting engine are discussed in the appendix.

3.4.4 Simulated Results and Performance Comparison

We use performance metrics to judge the goodness within different ICC models. Performance metrics are composed on 3 points of view

- Time
- Area
- Power Dissipation

And we used p for port number and R for register amount in the following discussion. The formulas are cited from [2] and have been well-reviewed.

The time performance metric is based on the total execution time, including executed cycles and access latency. The access latency is proportional to $pR^{0.5}$.

$$\text{Execution Time} = \text{cycle counts} * \text{access latency} \propto \text{cycle counts} * pR^{0.5}$$
$$P_{TIME} = \text{cycle counts} * pR^{0.5}$$

The register file area is composed by the summation of decoder area and storage cells area. By some research, ordinary 32-bit register file used about 20% of area for decoder and the rest for storage cells. Decoder area is proportional to $\lceil \log(R) \rceil$, and area of storage cells is proportional to P^2R.

$$P_{AREA} = 0.2 * \lceil \log(R) \rceil + 0.8 * P^2R$$

Power dissipation is dominated by the wire capacitance and grows as p^2R.

$$P_{POWER} = P^2R$$

Following is the execution cycle counts of the original(left) and after code rewriting(right). We can notice only on *Complex FIR* there is a slight amount of extra cycles(about 1%).

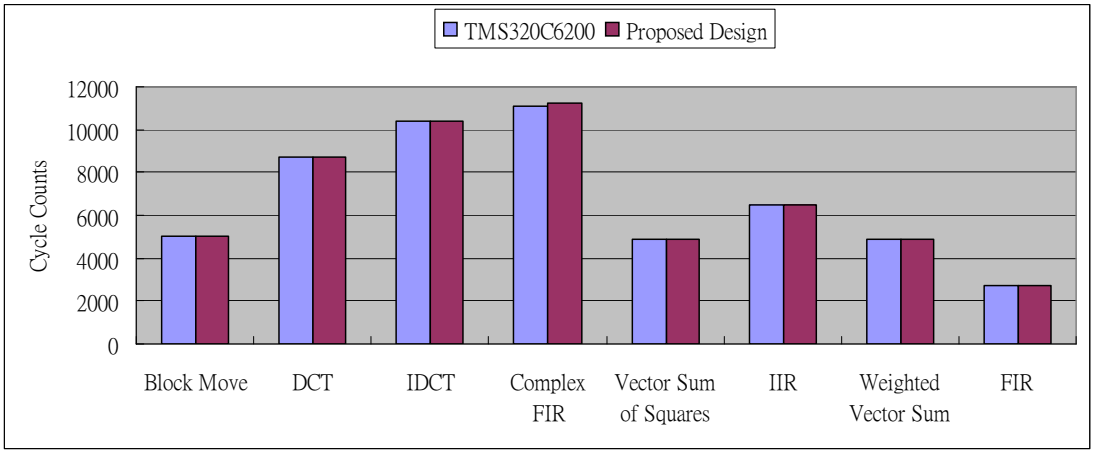

Figure 3.2: Executed Cycle Counts

And only up to 3 SPRs is needed to satisfy the ICC demands on these computation kernels.

Based on the extra cycles and amount of added SPRs, we apply the performance metrics. With the figure below, we can notice that either on time, area or power dissipation proposed design get better performance than the baseline machine. Proposed design consumes less time, fewer silicon area and less power dissipation than the baseline.

3.5 Results Summary

In the hand-optimized codes section, the FIR filter application performs the same on cycle counts for both the extended operands and proposed design perform , and pro-

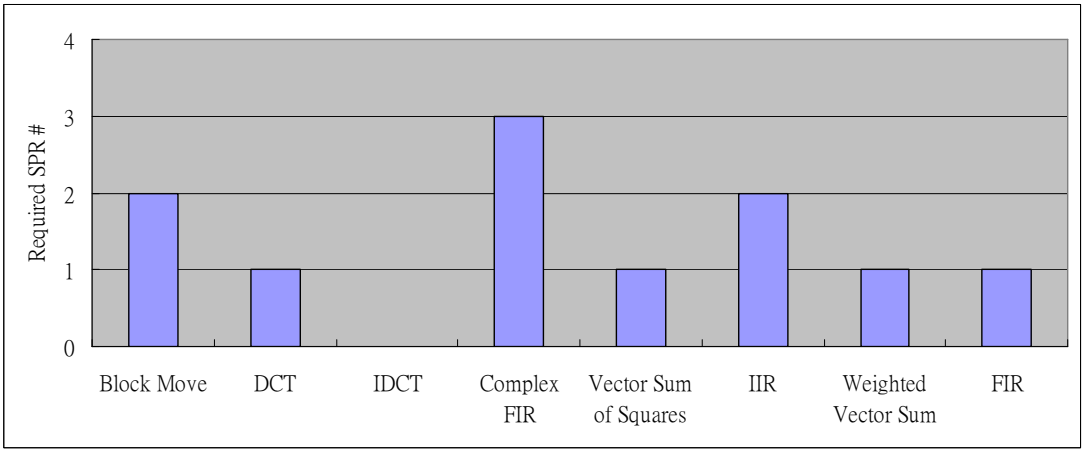

Figure 3.3: Required SPR amount

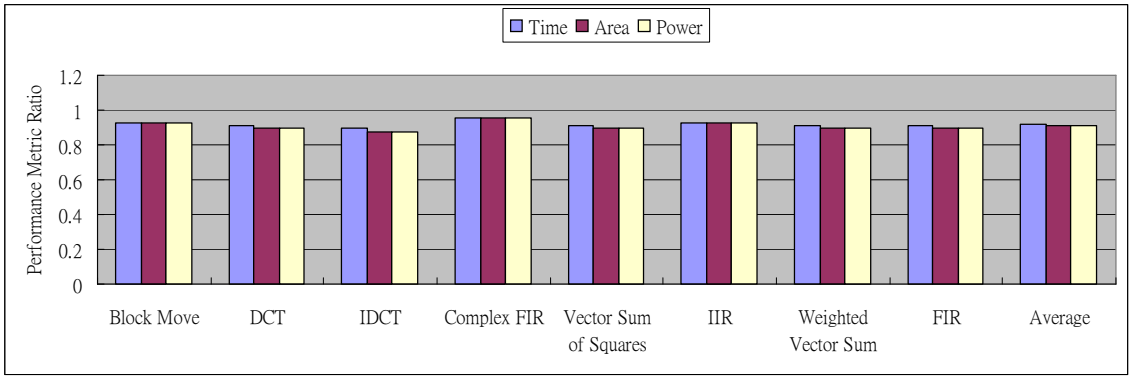

Figure 3.4: Summary of Performance Comparison

posed design can be executed more faster thanks for the port reduction of ICC. In the code-rewriting section, the results showed that for compiled codes optimized for extended operands models, transforming to proposed design can bring better performance, about averaged 10% improvement, on time, area and power dissipation with very low cost, up to 3 special registers is needed to add. Proposed design is a more efficient and economic way for inter-cluster communication.

3.6 Discussion

The code rewriting scheme can be used to transform compiled codes rapidly rather than designing a new compiler and corresponding optimization mechanisms. However on ultra-dense codes the rewriting may be not performing very well. For ultra-dense codes, inserting a copy for patching may also bring another extra cycles due to lack of available non-full issue slots in the existed issue frames. And for small and dense loop bodies, adding an extra issue frame and a corresponding extra cycle may bring dramatic performance impact on cycle counts. Thus for ultra-dense codes, the peep-hole rewriting approach may perform not so well than on sparse codes.

The performance improvement is brought by reducing the extra ports for inter-cluster communication. For machines equipped with larger part of ports for ICC, the advantages brought by port reduction will be more evident than those equipped with smaller part of ports for ICC. For example, for a 8-ports register file, reduction of 1 port makes the access latency reduced of 87.5% of original. But for a 20-ports register file, 1-port reduction reduces the access latency only to 95%.

The proposed design owns fewer-port advantages over other models, but may suffer from the increased register pressure. The design can be modified to dynamically replace some registers into SPR by inserting multiplexers in front of write ports. Thus the processor can replace more registers into SPR when high ICC demands are encountered and then restore them back to normal registers when register pressure is high. However the access latency will be also increased because signals need to pass through the extra multiplexers compared to the hard-wired approach.

On some cases, if the register files are very far from each other, the wires for accessing remote register file will dominate the delay over port numbers. However this issue also occurs on other models, and proposed design will suffer from the access latency

less thanks to the reduction of extra ports too.

Chapter 4

Conclusion and Future Works

4.1 Conclusion

In this thesis, we proposed the low-cost inter-cluster communication design. On top of that, we evaluate the performance of the well-known FIR filter on proposed design and the baseline machine. We also evaluated the cycle count overheads and required SPR numbers on transforming the native codes for baseline machine(extended operands) into the proposed design. The evaluation reveals that transforming causes up to 1% extra cycle counts and requires up to extra 3 special purpose registers for the generic common computation kernels. But access latency, area and power dissipation can be improved by eliminating the extra ports for inter-cluster communication on the baseline machine. That means adopting the propose design can make the processor run at a higher speed, be implemented by smaller area and consume less energy. Different from existed approaches using extra hardware to accomplish inter-cluster communication, proposed design only modifying existed wires, making it performs better on access latency, area and power dissipation.

4.2 Future Works

The idea of proposed design can be further examined with different approaches, and under a larger cluster configuration. We also arranged the hardware resource requirement compared to other ICC models under several classical topology in the following section.

4.2.1 Evaluation with Compilation Approach

The research can be improved by native compilation evaluation. Evaluating by hand-optimized codes is precise, but it's nearly impossible to ask programmers to write their code in assembly. In most cases programmers need a clever compiler to do the complicated low-level optimization for them. However the code rewriting transformation is a peephole technique and constrained by the original codes characteristics, making it limited to take full use of the proposed design. It is necessary to take cluster assignment, instruction scheduling and register assignment into concern for compilation approach. And taking all of them into concern together will certainly make the code quality better. We can also evaluate other models and larger clustered VLIW rapidly if the compiler with cluster support is available.

4.2.2 Larger Cluster

When cluster number is larger than 2, following factors dominates the resource requirement on inter-cluster communication:

1. Topology
2. Connectivity

The topology decides the shape of connected clusters, and connectivity decides required resource for each point-to-point connection.

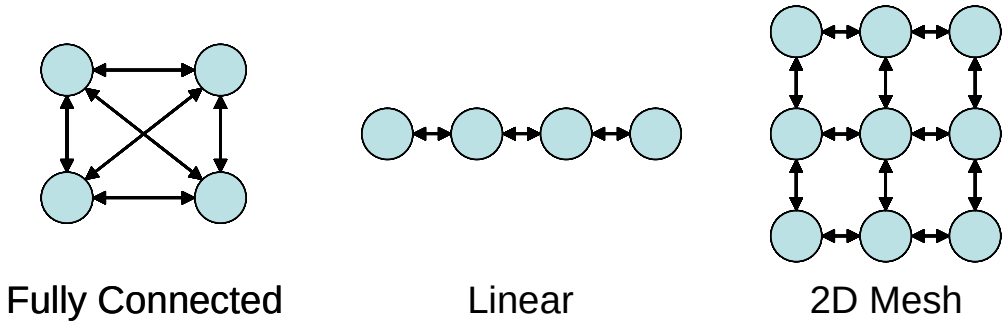

Figure 4.1: Different Topologies for Larger Cluster

The performance difference for various inter-cluster communication is a complicated problem though, we arranged the required resources on three common topologies between different communication models.

We can observed that as connectivity grows, resources for other models increase dramatically. However proposed design deploys registers rather than extra ports for inter-cluster communication, and it will improve the implementation of register files for larger cluster. And most important of all, proposed design will not slow down the register file access latency and can guarantee the executing frequency when clusters grow larger.

Table 4.1: Connectivity and Resource Requirement on Larger Cluster

Topology	Fully Connected	Linear	Mesh
Total Cluster Number	N		
Clusters to connect for each cluster	(N-1)	2	4
Extra Hardware Resource for Inter-Cluster Communication			
Copy Operation	(N-1)M ports	2M ports	4M ports
Dedicated Slots	(2N-2)M ports	4M ports	8M ports
Extended Results	(N-1)M ports	2M ports	4M ports
Extended Operands	(N-1)M ports	2M ports	4M ports
Extra Cluster Index Bit	$\lceil \log(N) \rceil$	2	3
Shared Registers	(3N-3)M ports	6M ports	12M ports
Proposed Design	(N-1)I Registers	2I Registers	4I Registers
Extra Cluster Index Bit	0	0	0
M stands for function unit numbers, and I stands for SPR number to each cluster			

Appendix A

Source codes of FIR4 for Modified TI TMS320C6200

```
* Modified FIR4 Filter,
* Inner loop is unrolled four times, and outer is unrolled 2 times
*
* CYCLE PERFORMANCE = M*(N+8)/2+6,
* M stands for outer loop times, N stands for Inner loop counts
*
* REMARK: *NO EXTRA STACK PROCESSING
* A4: &(X[]) ; input array
* A6: &(Y[]) ; output array
* A8: M ; output number
* B4: &(H[]) ; coefficient array
* B6: N ; coefficient number
*
* REGISTER ALLOCATION:*
* A0: B0: I
* A1: J B1: tmp
```

```
* A2: X[1] B2: X[0]
* A3: X[3] B3: X[2]
* A4: &X B4: &H
* A5: H[2] B5: (&Y)+1
* A6: &Y              B6: N
*
* A8: M->tmp          B8: SPECIAL X[0]
* A9: Sum0 B9: Sum1
* A10: BASE[X1/X3] B10: BASE[X0/X2]
* A11: BASE[H0/H2] B11: BASE[H1/H3]
*
* SPR Domain:
*
* A7: H[2] B7: H[3]
* A12: H[1] B12:  H[0]
* A13: H[3] B13:  H[2]
* A14: H[0] B14:  H[1]
* A15: tmp_b2a B15:  tmp_a2b

_FIR4:
B_START:
SHR .S1 A8, 1, A1 ; A1 is outer loop counter J
|| MV .L1 A6, B14 ; COPY &Y to Remote Cluster
|| MV .S1 A4, B15 ; B15 = X
|| MV  .S2 B4, A15 ; A15 = &H

ADD .D2 B14, 1, B5 ; B5 is (&Y)+1
|| MV .S1 A15, A11 ; Reset BASE[H0/H2]
|| ADD .S2 B4, 1, B11 ; Reset BASE[H1/H3]
```

```
|| ADD .L1 A4, 0, A10 ; Reset BASE[X0/X2]
|| ADD .L2 B15, 1, B10 ; Reset BASE[X1/X3]

SUB .S1 A1, 1, A1 ; OUTLOOP Counter j--
|| SHR .S2 B6, 2, B0 ; Set I = N/4

LDH .D1 *A10++[2], A2 ; LOAD X1
|| LDH .D2 *B10++[2], B3 ; LOAD X2

LDH .D1 *A11++[2], B12 ; LOAD H0[SP]
|| LDH .D2 *B11++[2], A12 ; LOAD H1[SP]

LDH .D1 *A10++[2], A3 ; LOAD X3
|| LDH .D2 *B10++[2], B2 ; LOAD X0

OUTLOOP:

ZERO .L1 A9 ; Sum0 = 0
|| ZERO .L2 B9 ; Sum1 = 0
|| LDH .D1 *A11++[2], B13 ; LOAD H2
|| LDH .D2 *B11++[2], A13 ; LOAD H3

LDH .D1 *A10++[2], A2 ; LOAD X1
|| LDH .D2 *B10++[2], B3 ; LOAD X2
|| SHR .S2 B6, 2, B0 ; Set Inner Loop Counter: I = N/4

LDH .D1 *A11++[2], B12 ; LOAD H0[SP]
|| LDH .D2 *B11++[2], A12 ; LOAD H1[SP]
```

```
MPY .M1 A2, A12, A8 ;  x1*h1 => [A8]
|| MPY .M2 B8, B12, A15 ;  x0*h0 => [A15]
|| LDH .D1 *A10++[2], A3 ; LOAD X3
|| LDH .D2 *B10++[2], B2 ; LOAD X0
|| MV .S1 A12, B14 ; COPY H[1]
|| MV .S2 B12, A14 ; COPY H[0]

[B0] B .S2 INNLOOP ; inner branch
||[B0] SUB .L2 B0, 1, B0 ; decrement loop counter
|| MPY .M1 A2, A14, B15 ;  x1*h0 => [B15]
|| MPY .M2 B3, B14, B1 ;  x2*h1 => [B1]
|| LDH .D1 *A11++[2], B13 ; LOAD H2
|| LDH .D2 *B11++[2], A13 ; LOAD H3

ADD .L1 A15, A9, A9 ; sum0 += x0*h0
|| MPY .M1 A3, A13, A8 ;  x3*h3 => [A8]
|| MPY .M2 B3, B13, A15 ;  x2*h2 => [A15]
|| LDH .D1 *A10++[2], A2 ; LOAD X1
|| LDH .D2 *B10++[2], B3 ; LOAD X2
|| MV .S1 A13, B7 ; COPY H[3]
|| MV .S2 B13, A7 ; COPY H[2]

INNLOOP:

ADD .L1 A8, A9, A9 ; sum0 += x1*h1
|| ADD .L2 B15, B9, B9 ; sum1 += x1*h0
|| MPY .M1 A3, A7, B15 ;  x3*h2 => [B15]
|| MPY .M2 B2, B7, B1 ;  x0*h3 => [B1]
|| LDH .D1 *A11++[2], B12 ; LOAD H0[SP]
```

```
|| LDH .D2 *B11++[2], A12 ; LOAD H1[SP]

ADD .L1 A15, A9, A9 ; sum0 += x2*h2
|| ADD .L2 B1, B9, B9 ; sum1 += x2*h1
|| MPY .M1 A2, A12, A8 ;  x1*h1 => [A8]
|| MPY .M2 B2, B12, A15 ;  x0*h0 => [A15]
|| LDH .D1 *A10++[2], A3 ; LOAD X3
|| LDH .D2 *B10++[2], B2 ; LOAD X0
|| MV .S1 A12, B14 ; COPY H[1]
|| MV .S2 B12, A14 ; COPY H[0]

ADD .L1 A8, A9, A9 ; sum0 += x3*h3
|| ADD .L2 B15, B9, B9 ; sum1 += x3*h2
|| MPY .M1 A2, A14, B15 ;  x1*h0 => [B15]
|| MPY .M2 B3, B14, B1 ;  x2*h1 => [B1]
|| LDH .D1 *A11++[2], B13 ; LOAD H2
|| LDH .D2 *B11++[2], A13 ; LOAD H3
||[B0] B .S1 INNLOOP
||[B0] SUB .S2 B0, 1, B0 ; INNER LOOP Counter i--

ADD .L1 A15, A9, A9 ; sum0 += x0*h0
|| ADD .L2 B1, B9, B9 ; sum1 += x0*h3
|| MPY .M1 A3, A13, A8 ;  x3*h3 => [A8]
|| MPY .M2 B3, B13, A15 ;  x2*h2 => [A15]
|| LDH .D1 *A10++[2], A2 ; LOAD X1
|| LDH .D2 *B10++[2], B3 ; LOAD X2
|| MV .S1 A13, B7 ; COPY H[3]
|| MV .S2 B13, A7 ; COPY H[2]
```

```
; then INNLOOP branch take effect here

ADD .L1 A8, A9, A9 ; sum0 += x1*h1
|| ADD .L2 B15, B9, B9 ; sum1 += x1*h0
|| MPY .M1 A3, A7, B15 ;  x3*h2 => [B15]
|| MPY .M2 B2, B7, B1 ;  x0*h3 => [B1]
|| ADD .S1 A4, 2, A4 ; [RESET] X   = X+2
||[A1] B .S2 OUTLOOP
|| ADD .D1 A4, 3, B15 ; [RESET] B15 = (X+2)+1
|| MV  .D2 B4, A15 ; [RESET] A15 = &H

ADD .L1 A15, A9, A9 ; sum0 += x2*h2
|| ADD .L2 B1, B9, B9 ; sum1 += x2*h1
|| LDH .D2 *A4, B8 ; [PROLOGUE] LOAD X[0] -> X[J]
|| ADD .S1 A4, 0, A10 ; [RESET] BASE[X0/X2] MUST
|| MV .S2 B15, B10 ; [RESET] BASE[X1/X3]
|| MV .D1 A15, A11 ; [RESET] BASE[H0/H2] OK

ADD .L1 A8, A9, A9 ; sum0 += x3*h3
|| ADD .L2 B15, B9, B9 ; sum1 += x3*h2
|| LDH .D1 *A10++[2], A2 ; [PROLOGUE] LOAD X1
|| LDH .D2 *B10++[2], B3 ; [PROLOGUE] LOAD X2
||[A1] SUB .S1 A1, 1, A1 ; OUTLOOP Counter j--
|| ADD .S2 B4, 1, B11 ; [RESET] BASE[H1/H3] OK

SHR .S1 A9, 15, A9 ; sum0>>15
|| ADD .S2 B1, B9, B9 ; sum1 += x0*h3
|| LDH .D1 *A11++[2], B12 ; LOAD H0[SP]
|| LDH .D2 *B11++[2], A12 ; LOAD H1[SP]
```

```
SHR .S2 B9, 15, B9 ; sum1>>15
|| LDH .D1 *A10++[2], A3 ; LOAD X3
|| LDH .D2 *B10++[2], B2 ; LOAD X0

STH .D1 A9, *A6++[2] ; *(y0++) = sum0
|| STH .D2 B9, *B5++[2] ; *(y1++) = sum1

; then OUTLOOP branch take effect here

B_END:
```

Appendix B

Code Rewriting on TMS320C6200 Instruction Set Architecture

Details on code rewriting technique for TMS320C6200 codes is presented in this chapter.

The rewriting relies on DEF-USE model as the intermediate to carry necessary information. DEF-USE Model is globally used in data-flow analysis, register allocation and related research. We can define DEF and USE with REMOTE and LOCAL extended attribute, thus we can describe all inter-cluster communication schemes in another view.

Rewriting TMS320C6200 codes into proposed design is based on following steps:

1. Removing all remote USE,
2. Insert a remote DEF to copy the value to export registers, and
3. Use a local USE to access the export registers on remote clusters.

Above cases can be simplified to remove remote USE and transform into only one remote DEF when the DEF is only referred by the remote cluster. Because only the remote cluster needed the value, the DEF can write the result directly to export registers rather than an extra copy to prevent to waste another issue slot. In this case we

identify the USE as a *Pure Remote Use*. And for those DEF will be referred by both local and remote clusters, we identify them as *Mixed Use*.

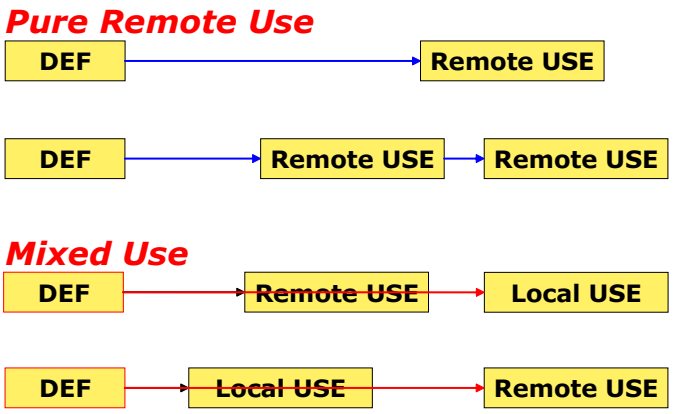

Figure B.1: Pure Remote Use and Mixed Use

When a Pure Remote Use is encountered, the corresponding DEF can be replaced by a R-DEF to write the result to export registers.

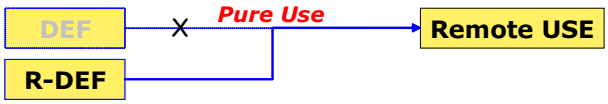

Figure B.2: Rewriting On Pure Remote Use

A Mixed Use can not be rewritten in the same way though, an extra copy must be inserted before the first USE is encountered. And it is also necessary to modify the USE to access the export registers instead of access remote register file for the copied value.

Based on the above concepts, the correctness can be maintained easily. However, the case may become complicated when the DEF-USE pairs go through different control paths(across several basic blocks). As the control path spans, some USE may be

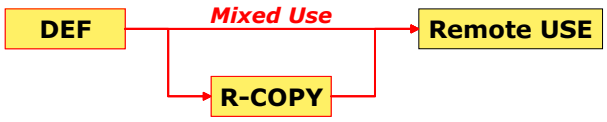

Figure B.3: Rewriting On Mixed Use

corresponding to several DEF statements, and we identify that as a multi-DEF case. Also one DEF statement may provide its operation results to several USE located in different control paths, in that case we identify it as multi-USE case.

When a multi-USE case is encountered, we insert the copy in the basic block where DEF located. If not, we may need to insert copies corresponding to each USE in different basic blocks. By insert the copy in the basic block where DEF located can reduce the copies and obey the 2nd rewriting principle which stated in chapter 4. The case is shown in the following figure, and the dashed lines is used to separate basic blocks.

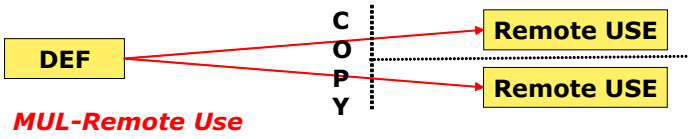

Figure B.4: Multiple USE in different basic blocks

For the multi-DEF, the situation is similar. However in order to not make the condition of multi-DEF-multi-USE too complicated to be handled, the multi-DEF cases are treated as multiple single-DEF statements to maintain the rewriting logic simple.

After handling above cases, we can guarantee all remote-USE statements can be rewritten into accessing the special purpose registers, which is our proposed design for ICC. In order to keep SPR usage fewer, we insert the induced copies as late as possible(ALAP). The induced copy will try its best to find a empty issue slot in the existed

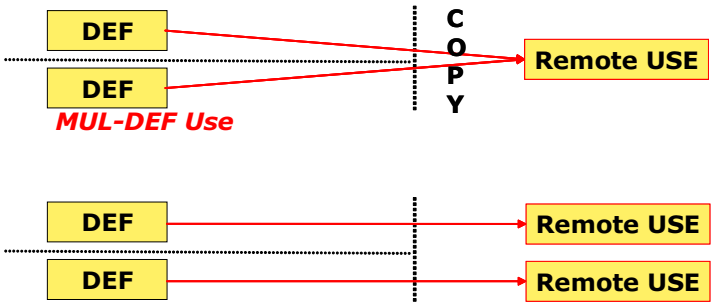

Figure B.5: Multiple DEF in different basic blocks

issue frames[1]. However if there is no empty issue slots available to insert the copy, a new issue frame will be inserted and thus an extra execution cycle will be penalized.

[1]also called *Issue Packets*

Bibliography

[1] D. Puppin, "Scheduling for vliw," 2001.

[2] S. Rixner, W. J. Dally, B. Khailany, P. R. Mattson, U. J. Kapasi, and J. D. Owens, "Register organization for media processing," in *HPCA*, pp. 375–386, 2000.

[3] A. T. Erwan, "Inter-cluster communication models for clustered vliw processors," 2003.

[4] T. Instruments, "Tms320c6000 cpu and instruction set reference guide," 2000.

[5] T. Instruments, "Tms320c6000 optimizing compiler user's guide," 2002.

[6] V. Cuppu, "Cycle accurate simulator for tms320c62x, 8 way vliw dsp processor," 1999.