# A Distributed Register File Architecture Based on Dynamic Scheduling for VLIW Machine

Yang Guo[1 2], Donglin Wang[1], Zijun Liu[1], Hongyu Meng[1 2],
[1] Institute of Automation, Chinese Academy of Sciences (CASIA), Beijing, China
[2] University of Chinese Academy of Sciences (UCAS), Beijing, China
E-mail: {guoyang2014, donglin.wang, zijun.liu, menghongyu2014}@ia.a.cn

*Abstract*—**Distributed register files have the advantages of area and power compared with centralized register files in very long instruction word (VLIW) machine. But it may complicate the internetwork. A distributed register file architecture based on dynamic scheduling is proposed in our work. A portion of latency on internetwork is shifted to the latency among registers by latency transfer. And the decrease in the fanout of writing ports leads to a reduction in the power of the internetwork. To ensure the external compatibility of the distributed register file interface, the register was redirection designed. A dynamic scheduling strategy is designed to avoid conflicts between physical entities and logical entities that redirect registers. The optimization was evaluated on our in-house processor. The register file based on distributed dynamic scheduling reduces 15.48% of latency. The system frequency is raised from 1.19 GHz to 1.41 GHz. The power decreased by 27.42%.**

***Keywords- distributed register file; VLIW machine; dynamic scheduling; system frequency; power***

## I. INTRODUCTION

Extremely expanding application’s requirements to computational performance is endless. Nowadays the demand for the ability to handle intensive computing in lots of areas such as communications, multimedia and artificial intelligence is growing. VLIW (Very Long Instruction Word) [1] architecture is the combination of multiple instructions that can be operated in parallel by the compiler after mining the potential parallelism between instructions to form a very long instruction word with more than one opcode field which improves the performance. Organization of traditional VLIW machine register file is that a number of functional units share a centralized global register file [2]. Because the centralized global register files are flawed in terms of area and power, the structure of distributed register file is proposed. It allocates traditional register files to different functional units. Each functional unit has a distributed local register file. Scott Rixner and others have done related research on centralized global register file and distributed register file [3].

The VLIW architecture in [4] uses a distributed register file implementation. Due to the characteristics of the distributed register file, the function units should be able to access the register files to each other, which makes the interconnection between the function units more complex. As shown in Fig.1, (1) is the interconnection structure of the functional unit with the centralized global register file, and (2) is the interconnection structure of the functional units with the distributed register file. It can be seen that the number of connections to functional units in the distributed registers has increased significantly.

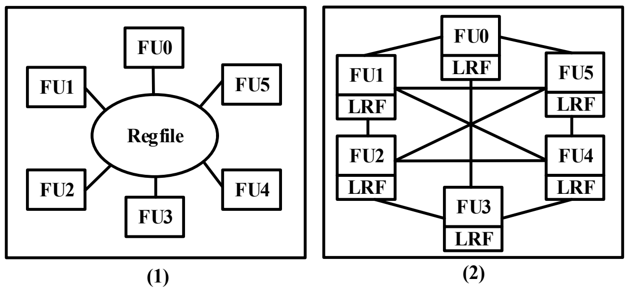

Figure 1. The internetwork of two register file architectures. (1): The centralized global register file. (2): The distributed register file. The connection of distributed register file is much more than centralized global register file.

With the advent of nanometer times, interconnects are becoming one of the most critical issues in the design of electronic circuits and systems. Performance, power and area are deeply affected by the interconnection, especially for the global internetwork [5]. It is reported that interconnection is more than 50% of the overall dynamic power in a microprocessor [6]. How to reduce internetwork latency and power has been a key issue in VLIW processor core design. The entire system structure is divided into several logical clusters called islands in [7-8]. Each island contains input logic, local register files, and functional units. The local register file is used to store the calculation results generated by the internal functional unit. It is also responsible for providing operands to the internal functional units and external functional units located on other islands. And an algorithm is proposed to optimize inter-island interconnects and limit register reading ports to reduce system power. However the latency between islands is ignored in [7-8]. A novel type of single-compartment latency model was proposed in [9].

At this stage most of the researches proceed from the structure of the internetwork and find the optimal clustering method through algorithm optimization to realize the optimization of latency and power. Rarely research the hardware structure of the register file to optimize. The register file is the connection of the internetwork to the functional unit. The latency and power of the internetwork are closely related to the interface to the register file. Our work focus on the

The Strategic Priority Research Program of Chinese Academy of Sciences (under Grant XDA-06010402) and Youth Innovation Promotion Association of Chinese Academy of Sciences (Y5S7061G51)

hardware structure and access patterns for register file to optimize the power and latency of the internetwork.

## II. NOVEL DISTRIBUTED REGISTER FILE ARCHITECTURE

### *A. Latency Transfer*

The novel distributed register file architecture reduces latency and power that the internetwork creates at the interface to register files by latency transfer. The traditional writing structure of the distributed register as shown in dashed box of Fig.2 (1). The amounts of register file's writing port, reading port and depth are assumed as 8, 4 and 8 respectively in the paper. Distributed register files connect external networks and functional units. The external network transfers data to the register file by writing operation. The local functional unit get the data in the register file to execute the operation in the instruction by reading operation. The functional unit can send the processed result to the internet to write to the register file of other unit, and can also write the result back to the local register file. The functional unit can send the processed result to the register file of other unit through internetwork, and can also write the result back to the local register file. The writing logic is in the dashed box, we can see that there is a multi-layer selection logic. This greatly increases the latency of the internetwork at the register file interface. And the long wire of the internetwork at writing ports of the high fan-out also brought considerable power.

To simplify the writing port logic of the register file, the register file of each writing port binds a register input. In the current cycle, if the writing port is enabled, the corresponding data will be unconditionally written to the register. The existing data in the register is judged. If the data are logically overwritten, no scheduling is needed. Otherwise it must be dispatched to other registers. The destination register to be dispatched must be a register whose write enable is not valid for the current cycle and original reserved data can be overwritten.

The idea of latency transfer is to transfer the multi-level selection logic before register file to register-to-register data scheduling. As Fig.2 (2) shows, the selection logic between the internetwork and the register file is reduced to only one level of latency and there is no high fan-out. Instead, there is a scheduling logic between the registers inside the register file. The network is inside the register file. so there is no long wire. The power of scheduling internal is much lower relatively.

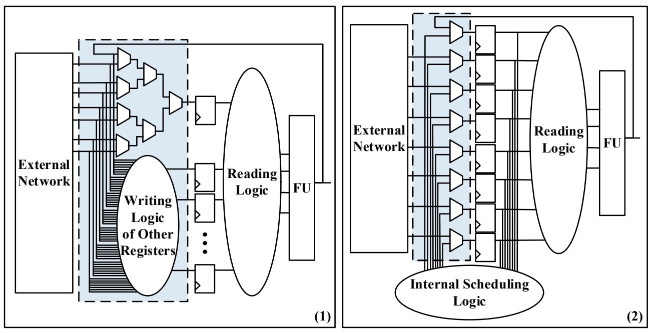

Figure 2. The structure of two register files. (1): The traditional structure of register file. (2): The redirection register file. The writing logic of redirection register file is much simpler than traditional one.

because its timing path is among registers, it does not take the internetwork timing latency, eased the timing pressure of the internetwork. And there is no other inherent latency in the timing path among the registers except for the scheduling logic. So it can not be the critical path that affect system frequency.

### *B. Register Redirection*

According to the idea of latency transfer and the structure of novel register file described above, the meaning of each register changes. Each register's physical and logical meaning of the traditional register file in Fig. 2 (1) is the same, that is, the data stored in each register physical represents the data stored in the logical register. The same property applied to the structure in Fig.2 (2) obviously leads to data storage errors. Therefore, the structure of register and the way of data stored must be changed. The concept of register redirection is proposed.

The $i^{th}$ physical register is denoted as $R_i$, $0 \leq i \leq 7$. Three bits are added to each physical register to indicate its logical meaning. Fig.3 shows the structure after register redirection. High-K bits are logical labels that represent the label of the logical register where the data in this physical register resides, denoted $R_i$[K]. The lower M bits represent the data stored in the logical register represented by the logical label of the physical register. The lower M bits are the same as the data in the traditional register file, denoted by $R_i$[M]. That is to say, a K-bit logical label is added to redirection register compared to the traditional register. The K is related to the depth of the register file. If the depth of the register file is N, $K = \log_2 N$. In this paper, K = 3. Since VLIW machines generally handle vector operations, M is large generally. Distributed registers are local registers for each functional unit. Their depths is shallow. The overhead of K-bit after taking the logarithm is minimal compared to M.

### *C. Reading Logic*

According to the structure of the redirection register, it can be seen that the logic of reading and writing changes. The logical meaning of each physical register is different in different clock cycle. Changes in writing logic involve the dynamic scheduling strategy, which is discussed in section 3. The characteristics of reading logic is discussed in this section.

The reading logic of traditional register file is that the reading address is regarded as the address of multiplexer directly. However, the physical number of the redirection register does not represent the logical number of the data stored. So in the process of reading, the reading address must be compared with the logical label of each register. The physical register whose logical label matchs is selected. The stored data is read out.

## III. DYNAMIC SCHEDULING STRATEGY

The novel register file structure described above reduces the latency of the internetwork and the power of long wire fan-

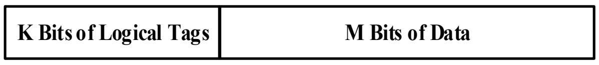

Figure 3. The structure of single redirection register.

outs. But to ensure that requirement of reading, writing and storage, the interface of the novel register file must be the same as traditional register file. A strategy for dynamic scheduling of redirection registers is presented in this chapter. Under the premise of normal access, we try to make a minimal amount of overhead.

## A. Register Classification

To assure the correctness, two types of registers must be identified before scheduling. One is the physical registers whose data stored have to be scheduled. These registers are call as α-type. When a logical register is written to a new value, the original value will be overwritten. Thus, in the novel register file architecture, the necessary and sufficient conditions for the physical register $R_i$ is an α-type are as follows.

$$(WEn_i = 1)\ \&\ (\nexists \text{j} \in \text{P}\ s.t.(WEn_j = 1)\&(WID_j = R_i[\text{K}])) \quad (1)$$

The registers whose stored data can be erased is the second type. It also means that the register can be used to accept other data. These registers are call as β-type. When the data in the register is overwritten by the newly written data, the old value of the data is meaningless and can be erased. Therefore, the necessary and sufficient conditions for the physical register $R_i$ is a β-type register are as follows.

$$(WEn_i = 0)\ \&\ (\exists \text{j} \in \text{P}\ s.t.(WEn_j = 1)\&(WID_j = R_i[\text{K}])) \quad (2)$$

## B. Conflict Analysis

Scheduling may conflict. A conflict occurs when multiple scheduled data are selected to the same physical register to be received. Conflicts can cause data storage errors. The focus of the scheduling strategy is to resolve conflicts. When the number of physical registers is 8, the maximum amount of registers to be scheduled is analysed.

The amount of the enabled writing port is denoted as n. When $1 \le n \le 4$, it is possible to schedule data only when writing port is enabled according to expression (1). So the amount of physical registers that need to be scheduled is at most n. When $5 \le n \le 8$, it can be analysed that if the amount of the data scheduled is 4, there must be 4 writing port is enabled, and there are 4 new data to be written to the register file. Then the 4 scheduled data and the 4 newly written data constitute all the data stored in the 8 logical registers. So the amount of physical registers that need to be scheduled is 4 at most in this case. In summary, the maximum value $S_{max}$ of physical registers that need to be scheduled is as follows.

$$S_{max} = \begin{cases} n & 1 \le n \le 4 \\ 4 & 5 \le n \le 8 \end{cases} \quad (3)$$

According to the equation (3), the maximum amount of data to be scheduled clock cycle is 4. When there are 4 data to be scheduled at the same time, any two of these data will not be scheduled to the same physical register have to be guaranteed. Each data scheduled must finds its own unique register to be accepted.

## C. Priority Scheduling Channel

According to the conflict analysis in the previous section, the maximum amount of data to be scheduled in each cycle is 4. Therefore, four scheduled channels can meet all the

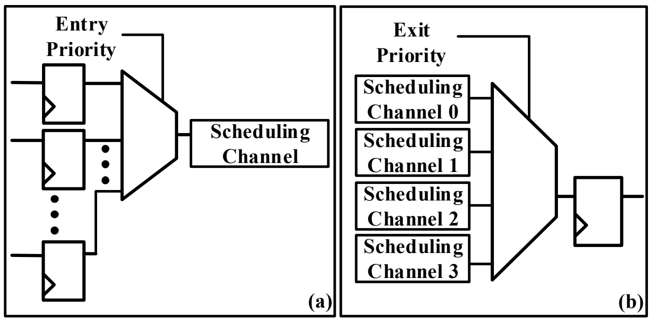

Figure 4. (1): The structure of entry logic. (2): The structure of exit logic.

conditions. There are two ports for each channel. The entry is connected to the source register for scheduled data. And the exit is connected to the destination register for scheduled data. The process of scheduling is divided into two steps. The first step is entry logic. Each channel chooses to receive a data to be scheduled in α-type registers according to its priority as shown in Fig.4 (1). The second step is exit logic. Each β-type register chooses to receive a data in four channels according to its priority as shown in Fig.4 (2). The address signal of multiplexer is controlled by priority. The priority determines directly whether the process of scheduling can avoid conflicts.

The entry priority for Fig.4 (1) is specified as $R_0$ to $R_7$ in descending order for channel 0. $R_0$ has the highest priority. Signal Schedule [7:0] is used to indicate whether $R_7$~$R_0$ is an α-type register. If Schedule [0] is true, the data in $R_0$ goes directly to channel 0. If Schedule [0] is false, $R_1$ is considered. If Schedule [1] is true, the data in $R_1$ goes directly to channel 0. In accordance with the law, the $R_0$ to $R_7$ are considered sequentially. The entry priority is specified as $R_1$ to $R_7$ in descending order for channel 1. Do not care about $R_0$. Because if Schedule [0] is true, the data in $R_0$ goes directly to channel 0 and not to channel 1. However, not only Schedule [i] to be judged, but also the data in $R_i$ has entered channel 0 needs to be considered. Accordingly, the entry priority is specified as $R_2$ to $R_7$ in descending order for channel 2. Not only Schedule [i] to be judged, but also the data in $R_i$ has entered channel 0 and channel 1 needs to be considered. The priority for channel 3 is to follow this law. The behavioural description of priority judgment discussed above is expressed as algorithm 1.

The logic of high-priority requires fewer selectors. The highest priority logic only needs one address signal to be judged. If the condition is met, the data is selected. Otherwise,

**Algorithm 1:** Entry priority

**Input:** Scheduling signal **Schedule** [7:0] and the value in registers **reg** [7:0].
**Output:** Channel value **channel** [3:0].

```
1 Initialize channel [3:0] ← 0 and FlagIn [7:0] ← 0;
    //FlagIn indicates whether the value in register has found a channel
    //to be scheduled
2 for i = 0 : 3
3   for j = i : 7
4     if Schedule [j] = 1 and FlagIn [j] = 0
5       channel [i]  ←  reg [j]
6       FlagIn [j]   ←  1
7     end if
8   end for
9 end for
```

**Algorithm 2:** Exit priority

**Input:** Accepting signal **Accept** [7:0] and the value in channel **channel** [3:0].
**Output:** Register value **Reg**[7:0].

```
1  Initialize   FlagOut[3:0] ← 0;
      // FlagOut indicates whether the value of the channel has found the
      //target register
2  for i = 7 : 0
3    for j = 0 : 3
4      if Accept [i] = 1 and FlagOut [j] = 0
5         channel[i] ←  reg[j]
6         FlagOut [j] ←  1
7      end if
8    end for
9  end for
```

the next priority is judged. So the higher priority, the lower latency. From the entry logic, it can be seen that the latency of the timing path starting with $R_0$ is the lowest, and the timing path with $R_7$ as the starting point has the highest latency. The longest timing path is the critical path that determines the system frequency. So in determining the exit priority, the timing balance to be noticed.

Signal Accept [7:0] is used to indicate whether $R_7$~$R_0$ is a β-type register. For the exit priority of Fig.4 (2), the $R_7$ with the longest entry logic is considered first. If Accept [7] is true, channel 0 data is directly written to $R_7$. Otherwise, $R_6$ is considered. The exit priority is specified as $R_7$ to $R_0$ in descending order. Each register judges channel 0 to channel 3 successively. The condition for writing data in channel to $R_i$ is that accept [i] is true and the data in channel is not written to a register of higher priority than $R_i$. The behavioural description is expressed as algorithm 2.

The scheduling strategy discussed above can guarantee the compatibility of the novel register file to the external interface. And it make little latency and low power overhead.

## IV. Result and Discussion

Mathematic processing unit (MaPU) is our in-house VLIW processor whose data format is 512-bit single instruction multiple data (SIMD). Its Dpath module is used as an experimental object to compare timing and power in this chapter. The MaPU core is composed of 8 Dpath modules. Each Dpath is 64 bits. There are 4 funtion units in a Dpath module. They are two 64-bit arithmetic logical units (ALUs) and two 64-bit multiply-accumulators (MACs). And there are 4 input channels and the interconnection among functional units and input channels. Each FU has a local distributed register file whose depth is 8. And it has 8 writing ports. Two identical Dpath modules were used to experiment. FUs of one were equipped with the traditional register file. FUs of the other were equipped with the register file based on dynamical scheduling. The result is evaluated after synthesizing with Design Compiler and place & route with IC Compiler, with a 16-nm logic library. The data of experiment are shown in Table 1 below.

It can be seen from the data in Table 1, the register file based on dynamic scheduling optimizes the latency of the internetwork from 0.84 ns to 0.71 ns. The 15.48% optimization in latency raised the system frequency from 1.19 GHz to 1.41 GHz. The DPath module power is reduced from 43.23 mW to 31.78 mW. It is optimized by 27.42%. It can be seen from the experiment that the register file based on dynamic scheduling can effectively reduce the latency and power in the processor.

TABLE I. EXPERIMENT COMPARISON

| | **Frequency (GHz)** | **Latency (ns)** | **Power (mW)** |
|---|---|---|---|
| Traditional register file | 1.19 | 0.84 | 43.24 |
| Register file based on dynamic scheduling | 1.41 | 0.71 | 31.38 |
| Absolute optimization | - | 0.13 | 11.86 |
| Relative optimization | - | 15.48% | 27.42% |

## Conclusion

For the problem of excessive latency and power in the internetwork, a distributed file architecture based on dynamic scheduling is proposed. The architecture transfers a portion of logic on the internetwork to the close range scheduling logic among the registers which relieves timing pressure and power overhead on the internetwork. And a register redirection architecture and a set of dynamic scheduling strategy are designed to ensure the compatibility of the interface. The architecture is evaluated for the timing and power information by synthesis and place & route. The register file based on dynamic scheduling optimizes 15.48% latency compared with the traditional distributed register file which improves system frequency from 1.19 GHz to 1.41 GHz. System power reduces by 27.42%. The register file architecture is suitable for any VLIW processor. Dynamic scheduling strategy for different parameters of the register file is valid.

## References

[1] Techcon, Arm. "Implementing the Viterbi algorithm in modern digital communications systems." Eetimes Com (2009).

[2] Yang, Yan. "Design and Implementation of VLIW Processor System Level Verification Platform." Journal of Electronic Measurement & Instrument 21.2(2007):81-85.

[3] Rixner, Scott, et al. Register organization for media processing. Register Organization for Media Processing. 2000:375-386.

[4] Wang, Donglin, et al. "MaPU: A novel mathematical computing architecture." IEEE International Symposium on High PERFORMANCE Computer Architecture IEEE, 2016:457-468.

[5] Bottoms, Bill. "The International Roadmap for Semiconductors 2007." International Conference on Electronic Packaging Technology IEEE, 2007:1-1.

[6] Magen, Nir, et al. "Interconnect-power dissipation in a microprocessor. " System Level Interconnect Prediction (2004):7-13.

[7] Cong, Jason, Y. Fan, and J. Xu. "Simultaneous resource binding and interconnection optimization based on a distributed register-file microarchitecture." Acm Transactions on Design Automation of Electronic Systems 14.3(2006):824-833.

[8] Huang, Juinn Dar, et al. "Communication Synthesis for Interconnect Minimization Targeting Distributed Register-File Microarchitecture." Ieice Transactions on Fundamentals of Electronics Communications & Computer Sciences 94-A.4(2011):1151-1155.

[9] Huang, Juinn Dar, et al. "Performance-driven architectural synthesis for distributed register-file microarchitecture considering inter-island delay." International Symposium on Vlsi Design Automation and Test IEEE, 2012:169-172.