To be appeared in 17th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD2005)

Cooperation of Neighboring PEs in Clustered Architectures

Yukinori Sato, Ken-ichi Suzuki and Tadao Nakamura
Graduate School of Information Sciences, Tohoku University
6-6-01, Aramaki Aza Aoba, Aoba-ku, Sendai, 981-8579, Japan
{yukinori,suzuki,nakamura}@archi.is.tohoku.ac.jp

Abstract

Clustered architectures which intend to process data within a localized PE are one of the approaches to increase the performance under the difficulties of the wire delay problems. The performance of clustered architectures depends on the amount of parallel execution of instructions and the amount of inter-PE communication to synchronize dependent instructions. In this paper, we propose an arrangement of PEs cooperating with the adjacent PEs by means of adding communication structures between the adjacent PEs in order to relax the inter-PE communication and workload imbalance in an effective manner. We evaluate the proposed configurations and compare them with the existing one so far considered. The results show that the proposed adjacent forwarding network configuration with the instruction steering scheme that concerns both the register fanout and available free register can achieve higher instructions per clock (IPC) with the small number of registers per PE than the other configurations.

1. Introduction

Considering today's advanced CMOS technology scaling that allows high transistor density, a novel paradigm of data processing is required to cover up the large wire delay. Dynamically-scheduled clustered architectures to process data locally will be able to fulfill this requirement [1], [13], [14]. In the clustered architectures, global structures are partitioned into simple smaller structures and each of them is arranged in a PE (processing element) called cluster in some papers. This partitioning makes the hardware simpler and its control and data paths faster because the number of entries and ports of the partitioned structures can be reduced.

The performance of clustered architectures depends on the amount of parallel execution of instructions and the amount of inter-PE communication to synchronize dependent instructions. If too many instructions are steered to a particular PE, then communication among PEs seldom occurs. However, PEs are deprived of working in parallel and the instructions in the overloaded PE cause resource conflicts, which degrade performance. This is referred to as workload imbalance. On the contrary, if instructions are steered to PEs evenly, the possibility of parallel processing is increased. However, the amount of inter-PE communication is also increased, which also degrades performance. Hence, we must design a clustered architecture that balances the workload and communication among PEs.

Many proposals for instruction steering schemes tried balancing the workload and communication across PEs [1], [14], [16]. However, just using the existing steering schemes, there are limitations in increasing the performance. In order to overcome these limitations of the existing steering schemes, we must redesign some hardware components in addition to instruction steering schemes. The key components that should be reconsidered are communication structures between PEs that affect the delay of the communication.

In this paper, we make every pair of neighboring PEs cooperate with each other in the clustered architecture. To achieve effective cooperation, we add direct communication structures between neighboring PEs and we propose novel instruction steering schemes suitable for the structures. The additional communication structure can reduce the latency of the communication between neighboring PEs. The load imbalance is also avoidable since instructions can be steered with more flexibility without extra inter-PE communication delay.

The rest of this paper is organized as follows. In section 2, we briefly show the overview of a baseline clustered architecture and a baseline instruction steering scheme. Then, we discuss the limitation of the existing steering schemes and consider making every neighboring PE cooperate with each other. Section 3 describes the experimental framework, the evaluation methodology and the results. Section 4 shows some related work. Section 5 concludes this paper.

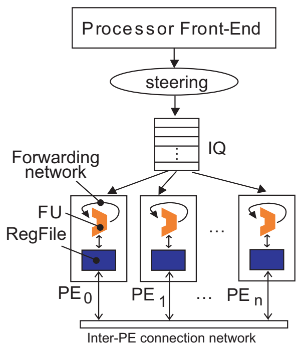


Figure 1. The overview of the clustered architecture.

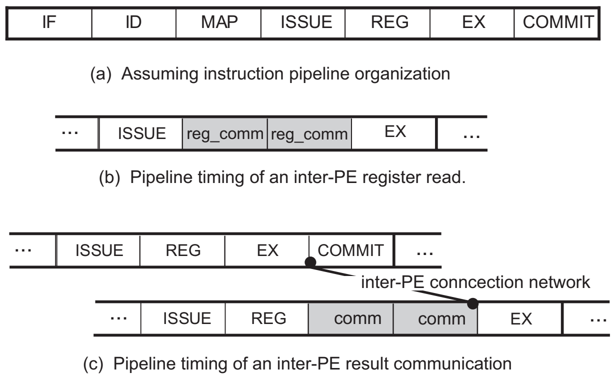

IF
ID
MAP
ISSUE
REG
EX
COMMIT
(a) Assuming instruction pipeline organization
ISSUE
reg_comm
reg_comm
EX
(b) Pipeline timing of an inter-PE register read.
ISSUE
REG
EX
COMMIT
inter-PE conncection network
ISSUE
REG
comm
comm
EX
(c) Pipeline timing of an inter-PE result communication

Figure 2. The timing of the pipeline.

2. Clustered architecture

2.1. Baseline microarchitecture

The microarchitecture of a clustered architecture is based on that of the aggressive out-of-order issue superscalar processors. Fig. 1 shows the overview of the baseline clustered architecture. The processor front-end fetches multiple instructions at once and decodes them. The decoded instructions are delivered to the steering logic. The steering logic chooses a PE for the execution of each instruction. Next, the steered instruction is dispatched to the IQ (issue queue) that observes whether the operand status of each instruction is ready or not. When required operands are ready, the instruction is waked up and the corresponding resources of the steered PE are checked. If the resources are available, the instruction is selected and issued to the PE and executed.

Most of current instruction set architectures are based on a single set of registers. However, the clustered architectures provide the physically partitioned set of registers in each PE in order to process data within a PE. Therefore, register mapping mechanism needs to map the architectural registers into the partitioned physical registers in an effective manner. One effective organization of the partitioned RegFiles is non-consistent RegFiles, where a register in each RegFile has its own register instance since a result is written into only one register [12]. In this configuration, any registers do not duplicate a register instance. On the other hand, in multiple-coherent RegFiles, register instances are replicated across PEs by writing all the results to each register of all RegFiles such as Alpha21264 [10]. The non-consistent RegFiles are composed of a smaller number of registers, so we assume clustered architectures with non-consistent RegFiles.

Fig. 2 (a) shows the pipeline stages assumed in this paper, which are based on those of Alpha21264 [10]. In MAP stage, the renaming logic allocates a destination register to an available free register and its mapping is recorded in the map table. In order to accommodate the renaming mechanism to the non-consistent RegFiles configuration, we prepare a free register list for each PE. The instruction steering mechanism must select a PE before the register renaming process because the selection of a free register list determines a PE where the instruction is executed. The register renaming mechanism we adopted is also based on that of Alpha21264. In the renaming process of Alpha21264, committed register instances are always stored in dedicated registers whose identifiable numbers are \$0-\$31, and those dedicated registers are also replicated across the partitioned RegFiles. On the other hand, in the non-consistent RegFile configuration, we assume that the committed registers are partitioned into PEs and each PE has the same number of committed registers to avoid converging register pressure on the particular PEs. For example, in 8 PE configuration, 32 committed registers (\$0-\$31) are partitioned as follows: PE_0 has \$0-\$3, PE_1 has \$4-\$7, ... , PE_7 has \$28-\$31.

In ISSUE stage, instructions in the IQ are checked whether their operands are ready and their corresponding functional units are available. If all the operands are ready and the corresponding functional unit is available, the operands are read from RegFiles in REG stage. When the operand is stored in the same PE, it can be read in a single cycle. When the operand is stored in a remote PE, we assume that the operand fetch requires one extra cycle for communication. Fig. 2 (b) shows the pipeline timing of an inter-PE communication due to the register read. After REG stage, the instruction is executed in EX stage in its given latency.

In the case where the instruction uses at least one unready operand, the instruction must wait until the results of the preceding instructions are provided. When the preceding dependent instruction is executed in the same PE as the waiting instruction, the waiting instruction is executed at the next cycle of the execution of the preceding instruction using a forwarding network. On the other hand, when the waiting instruction is allocated in a different PE from the preceding dependent instruction, we assume that it takes 2 extra cycles for the inter-PE communication as shown in Fig. 2 (c). This timing model of processing dependent instructions using inter-PE communication is the same as the model in [14] that inserts copy instructions dynamically.

We assume sufficient bandwidth for the interconnection network to isolate our results from possible communication bandwidth bottlenecks. Although the fully-connected network assumed here requires the same amount of hardware cost as that of multiple-coherent RegFiles, the configuration with non-consistent RegFile can reduce the number of registers than the multiple-coherent RegFiles. On the other hand, the The inter-PE communication latency affects the non-consistent RegFiles configuration more than the multiple-coherent RegFile configuration. This is because the non-consistent configuration requires inter-PE communication using the interconnection networks after a PE is found to use non-local registers or when a particular PE lacks available free registers. The effect of the organizations of interconnection networks and partitioned RegFiles will be evaluated in our future work.

In this paper, we assume all PEs share a single IQ as shown in Fig. 1 in order to concentrate on the effects of cooperation of neighboring PEs. If we partition the IQ across the PEs, we have to take into account the utilization of each queue. The effect of the partitioned IQ will be evaluated in our future work.

2.2. Baseline instruction steering scheme

We use a steering scheme based on the status of operands which is named !ready (not ready) as a baseline steering scheme in this work since this scheme is reported to show a good IPC [16]. This scheme gives priority to the instructions with at least one unready operand in order to prevent the undesirable inter-PE communications. The instructions with unready source operands are steered to the same PE as the preceding dependent instruction. The instructions with only ready source operands are steered to the minimum loaded PE in order to even up the loads of PEs. The heuristics to measure the loads of PEs is DCOUNT, which is the product of the number of PEs and the difference between the total number of instructions dispatched to the PE and the average number of instructions dispatched per PE [14].

Table 1 (a) shows the relationship between the status of source operands of an instruction and its steered PE on the !ready scheme. The first column and first row of each table denote the status of the two source operands of a consumer instruction, in1 and in2. The status of a source operand is classified as follows: dependent operand is nothing (null), operand is not ready (!ready), or operand is ready (ready). The rest of the table indicates which PE the instruction is

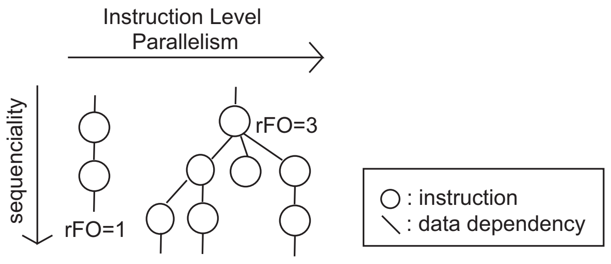


Figure 3. A snapshot of dynamic code.

steered to for each operand status. For example, if source operand in1 is ready and in2 is !ready, then the scheme steers the instruction to the PE having operand in2 in its RegFile.

2.3. Limitation of the existing steering schemes

The performance of clustered architectures depends on the amount of parallel execution of instructions and the amount of inter-PE communication to synchronize dependent instructions. In order to achieve both higher parallel execution and lower inter-PE communication, we focus on the metric called register fanout (rFO) [15]. The register fanout is represented by the number of times a particular register instance is used by subsequent instructions. Butts and Sohi showed that the portion of rFO=1 in dynamic code is 65%, rFO=2 is 14% and the average number of rFO is 1.7 in SPEC2000int benchmark suite [4]. This fact shows that most instructions are rFO$\leq$2 and sequentiality of codes are general characteristics of programs.

Fig. 3 shows parallelism and sequentiality of codes. Data-dependent instructions with rFO=1 must be processed in serial. Grouping instructions according to dependency leads to localized data processing within each PE. On the other hand, in a group of dependent instructions, instructions following the instruction whose register fanout exceeds one can be processed in parallel. In Fig. 3, the register fanout of the first instruction in the right dependent group is three. Therefore, the three instructions following the first one can be processed in parallel if they are assigned to individual PEs and the dependent data is given properly.

However, just using the existing steering schemes, the instructions waiting for the same result tend to be steered into the same PE and cause resource conflicts in the PE, which prevent the parallel execution. Therefore, a novel instruction steering scheme and a data handling mechanism are required to assign parallel-executable instructions to individual PEs and forward the data properly.

In order to overcome this limitation of the existing steering schemes, we must redesign some hardware components in addition to instruction steering schemes. The key components that should be reconsidered are communication struc-

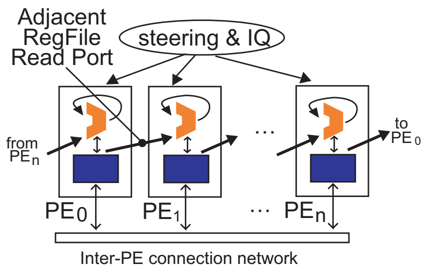


Figure 4. The adjacent read RegFile configuration.

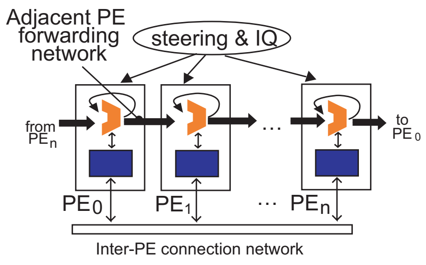


Figure 5. The adjacent PE forwarding configuration.

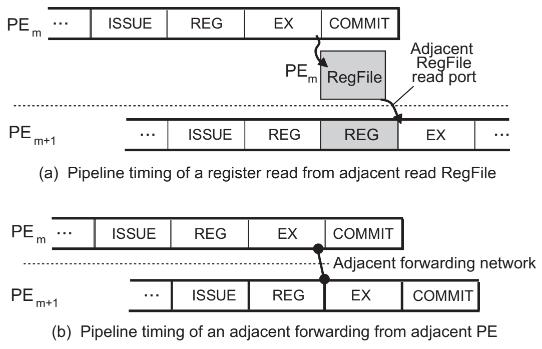

PE$_m$
ISSUE
REG
EX
COMMIT
PE$_m$ RegFile
Adjacent RegFile read port
PE$_{m+1}$
REG
(a) Pipeline timing of a register read from adjacent read RegFile
Adjacent forwarding network
(b) Pipeline timing of an adjacent forwarding from adjacent PE

Figure 6. Executions of dependent instructions in the proposed architecture.

tures between multiple RegFiles, the number of registers and instruction steering scheme suited to these structures. Communication structures between multiple RegFiles affect the delay of the communication. The number of registers in a PE decides hardware cost in a RegFile design and the number of available free registers for register renaming.

2.4. Proposed microarchitecture

In order to realize effective parallel execution by multiple PEs after the instructions of rFO>1, we make every neighboring PE cooperate with each other by means of additional structures that support direct communication between neighboring PEs. Here, we assume that the PEs are aligned to form a loop and the additional hardware is put between every two PEs to support communication of them, as shown in Figs. 4 and 5. The additional hardware can reduce the latency of the communication between adjacent PEs because physical distance between adjacent PEs might be short enough to cover up the wire delay. The load imbalance is also relaxed since instructions can be steered with more flexibility without extra communication delay.

In the cooperation of PEs, one way communication is enough to support rFO≤2 instructions that occupy most of dynamic instructions. We compare the following two communication structures to shorten the latency between adjacent PEs. The first structure enables a RegFile to be read from its right adjacent PE. This configuration can be implemented by adding one extra read port to the RegFile, as shown in Fig. 4. The second one is forwarding networks that can directly forward a result to the right adjacent PE. This configuration can be implemented by adding a direct forwarding network between adjacent PEs shown in Fig. 5. Both structures can be realized with less hardware cost than the conventional monolithic RegFile.

The RegFile structure which can be read by the adjacent PE has an effect in reducing the inter-PE communication latency in the operand fetch from the left adjacent RegFile. This enables a single cycle read from the left adjacent RegFile. In the conventional clustered architecture, it takes two cycles when the operand is stored in a remote PE, whether the PE is adjacent or not. Fig. 6 (a) shows the pipeline timing of executions of dependent instructions using this RegFile. When an instruction has waited the result from the adjacent PE, the instruction can be processed after one cycle delay, which comes from the latency of the register read.

The adjacent forwarding network structure allows contiguous executions of dependent instructions. Fig. 6 (b) shows the pipeline timing of executions of dependent instructions using the adjacent PE forwarding network. When an instruction has waited the result from the left adjacent PE, the instruction can be processed in the next cycle without transfer delay using the direct forwarding network.

2.5. Proposed instruction steering scheme

We present instruction steering heuristics based on the number of register fanouts in order to make full use of the proposed communication structures for the cooperation. The instruction steering scheme based on the heuristics that enables parallel execution of instructions after an rFO>1 instruction is named adjacent_rFO scheme.

Table 1(b) depicts the adjacent_rFO scheme. This scheme gives priority to instructions with unready operands

Table 1. The status of operands and its steered PE.

(a) !ready scheme

in1 \ in2	null	!ready	ready
null	Min_dcount	in2	Min_dcount
!ready	in1	in1	in1
ready	Min_dcount	in2	Min_dcount

(b) Adjacent_rFO scheme

in1 \ in2	null	!ready	ready
null	Min_dcount	rFO(in2)	Min_dcount
!ready	rFO(in1)	rFO(in1)	rFO(in1)
ready	Min_dcount	rFO(in2)	Min_dcount

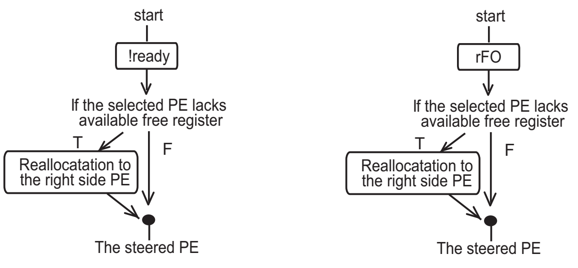
The Min_dcount indicates that the instruction is steered to the PE with minimum DCOUNT.
The in1 and in2 indicate that the instruction is steered to the producer PE of source operand in1 and in2, respectively.
The rFO(x) indicates that the instruction is steered based on the number of register fanouts of the source operand X.

similar to the !ready scheme. Instructions with unready operands are steered according to the number of rFO. The number of rFO is dynamically observed for each register. Instructions of rFO=2 or rFO=4 are steered to the right adjacent PE from the producer PE of a source operand, and the other instructions are steered to the producer PE of a source operand. The rFO(X) in the table indicates this steering for source operand X. This steering is useful for reducing the inter-PE communication or load imbalance after rFO>1 instructions. On the other hand, instructions with ready operands are steered to the minimum loaded PE for balancing workload as the !ready scheme does.

For example, let's assume that three successive instructions B, C, D are dependent only on an instruction A and the result of A has not been ready. B is assigned to the same PE as A, because it is the first successor of A (rFO=1). C is assigned to the right PE of A, because it is the second (rFO=2). Lastly, D is assigned to the A's PE again because it is the third (rFO=3).

2.6. The number of free registers in each PE

If the number of registers per PE is reduced, then the clustered architecture with small RegFiles can be realized. However, it may suffer from lack of available free registers. In the non-consistent RegFiles configuration, the lack of available free registers may cause changes of the decisions of instruction steering schemes. This is because the number of available free registers in a PE is different from the others since the number of in-flight instructions in a PE is different from the others. When the PE selected by the instruction steering scheme lacks available free registers, the instruction must be reallocated to one of the other PEs.


start
!ready
If the selected PE lacks available free register
T
F
Reallocatation to the right side PE
The steered PE
start
rFO
If the selected PE lacks available free register
T
F
Reallocatation to the right side PE
The steered PE

(a) Adjacent_freereg scheme (b) Adjacent_rFO_freereg scheme

Figure 7. The steering schemes based on the number of free register heuristics.

This reallocation causes extra inter-PE communication delay, which causes performance degradation. This behavior cannot be handled well in the existing steering schemes. In !ready and adjacent_rFO steering schemes, if the instruction must be reallocated due to the lack of available free registers, the instruction is reallocated to the PE with the largest number of available free registers regardless of the communication delay.

Then, we propose the heuristics for instruction steering based on communication delays and and the number of free registers in each PE. This can reduce the inter-PE communication delay due to the undesirable allocation by the cooperation of adjacent PEs. When a candidate PE selected by the steering scheme lacks free rename registers, we reallocate the instruction into its adjacent right PE.

The instruction steering schemes based on this heuristics are named adjacent_freereg and adjacent_rFO_freereg. Fig. 7 depicts these two schemes. These schemes work as follows: first, an instruction is tried to be steered in the same way as the !ready or adjacent_rFO scheme, respectively. Next, the schemes check whether the selected PE has available registers. If the instruction needs reallocation due to the lack of available registers, the instruction is steered to the right side PE of the PE originally selected. Therefore, if the instruction does not need to be reallocated, the steering scheme is the same as !ready or adjacent_rFO scheme.

The number of available free registers in a PE is affected by the number of registers in it. To realize an effective clustered architecture, we should try to process data with a reasonable amount of hardware. Therefore, we also focus on the number of registers per PE as an indicator of effectiveness in order to evaluate clustered architectures in terms of not only the performance but also the hardware cost.

Table 2. Main architectural parameters.

Fetch and decode	8 instructions per cycle
Branch predictor	Tournament branch predictor
IQ, FQ, LQ, SQ size	64
ROB size	256
Functional Units	1 ALU + 1 MUL per int. PE
Issue width	1 inst per PE
The number of PEs	8 int + 1 fp
Icache	128kB, 2way
Dcache	128kB, 2way

3. Experiments

3.1. Methodology

We developed a cycle-accurate execution-driven simulator to evaluate the proposed communication structures and instruction steering schemes. Baseline simulator is sim-alpha [6], which is one of the extention version of SimpleScalar tool set [3]. Sim-alpha models the detailed microarchitecture of Alpha21264, which is one of the clustered architectures composed of dual integer PEs (clusters).

We modified sim-alpha to model the clustered architecture with all the architectural features described in the previous section. In this paper, we assume an 8-way processor, which has 8 homogeneous integer PEs. The integer PE issues 1 instruction per cycle. The rest of the configuration such as out-of-order issue policy, latency of the caches and that of functional units are following that of Alpha21264. The main architectural parameters for the assuming architecture are shown in Table 2.

We have selected a subset of 4 benchmarks (djpeg, cjpeg, rawdaudio, rawcaudio) from the MediaBench benchmark suite [11]. This benchmark suite captures the main features of commercial multimedia applications. Benchmarks which tend to achieve high instruction-level parallelism have been selected. We have also selected a subset of 7 benchmarks (gzip, vpr, gcc, mcf, perlbmk, bzip, twolf) from the SPEC2000CPU int benchmark suite [9]. The rest of SPEC2000 benchmarks could not be adapted to the simulation environment used. All the benchmarks were compiled for the Alpha binary using Compaq's C compiler v6.5 on Tru64 UNIX V5.1B with -O4 -fast -non_shared options. Each program of the MediaBench was executed until the completion and 100 million instructions of each program of the SPEC2000int were executed after forwarding 1 billion instructions.

3.2. Results

Fig. 8 shows instructions per clock (IPC) for the four communication structures. The 'conventional' represents the configuration without any additional communication structures. The 'AF' represents the adjacent forwarding networks and the 'ARF' represents the adjacent read RegFiles. The 'AF+ARF' represents the configuration with both the adjacent forwarding networks and adjacent read RegFiles. The number of registers per PE is 40. The instruction steering schemes utilized are the !ready scheme for the conventional configuration and the adjacent_rFO_freereg scheme for the others.

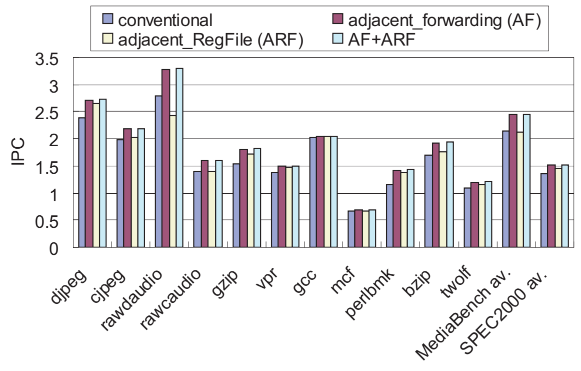


Figure 8. The IPC with 40 registers per PE configuration.

The results show that IPC of the configuration with AF is always larger than that of ARF configuration. The average IPC of the AF configuration is improved by 14% in MediaBench and 11% in SPEC2000int compared with the conventional configuration, respectively. On the other hand, the average IPC of the ARF configuration is not improved very much compared with the conventional configuration.

The reason why the ARF configuration cannot improve its IPC is because of dependencies with unready source operands. Our previous work showed that the communication delay between dependencies with unready source operands tend to be more responsible for the execution time than that with ready source operands stored in RegFiles [16]. This means that the dependencies with unready source operands should be processed as soon as the result is produced. However, in the ARF configuration, it takes one extra clock to process the dependencies with unready source operands from the adjacent PE due to the inter-PE register read compared with immediate executions of them.

In contrast, the AF configuration can process the dependencies with unready operands more effective way. The AF configuration can process an instruction with an unready operand from the adjacent PE in the next clock cycle of the execution of preceding dependent instruction. This can reduce the communication delay that seriously affects the performance. Moreover, workload balancing is improved since instructions can be steered with more flexibility avoiding extra inter-PE communication delay. Therefore, the AF configuration is superior to the ARF configuration.

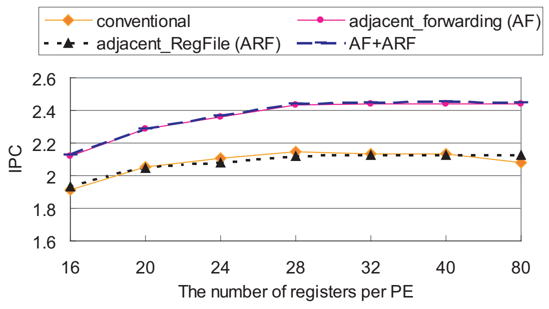


Figure 9. The effects of the number of registers.

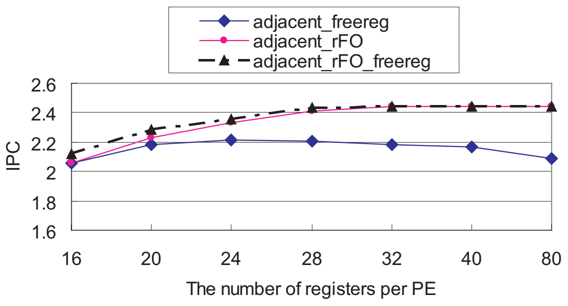


Figure 10. The differences among instruction steering schemes.

It is also observed that the difference of IPCs between the AF and AF+ARF are very small. Throughout the benchmarks, the IPC using AF+ARF is subtly superior to the IPC just using the AF. However, in terms of the effective design, the advantage of the configuration with the ARF is very small since we must add the extra ports to the adjacent RegFile in adjacent PE.

In order to discuss the effectiveness of the communication structures, we also vary the number of registers. Fig. 9 shows the IPC for each configuration. The IPC represented in the figure is the average number of MediaBench suite. The differences of the IPCs among the number of registers per PE are caused by the unbalanced register pressure among PEs. The lack of available free registers causes the reallocation of the PE and this causes the extra inter-PE communication compared to the case where the PE has a large enough number of registers.

It is also observed that the IPC is increased if the number of registers is increased. However, the increase of the IPC is saturated near the configuration with 28 registers per PE (in total 224 registers). This implies that the register pressure of the clustered architectures with non-consistent RegFiles is larger than that of conventional 8-way 64-entry IQ superscalars, which saturates the performance in 128 registers in total [7]. We note that the minimum number of registers per PE (16 registers per PE and in total 128 registers) comes from the saturation point of superscalars.

We can also understand that the minimum IPC of the configuration of AF is higher than the maximum IPC of conventional schemes. This means that even if we prepare a large number of registers, we cannot obtain much return in the conventional configuration. Instead of increasing the number of registers, we should prepare the adjacent PE forwarding networks.

For the sake of examining the effects of instruction steering schemes in the adjacent forwarding configuration, we compare the instruction steering schemes discussed in the previous section. Fig. 10 shows the differences among the instruction steering schemes. All the IPCs are obtained using the AF configuration. The IPC represented in the figure is the average number of MediaBench suite.

The effect of the adjacent_freereg steering scheme is significant in the configuration with small number of registers. However, if the number of registers is increased, the return of this scheme is decreased since each PE has enough number of registers and the lack of available free registers seldom happens. The adjacent_rFO steering scheme cannot achieve the distinguished IPC with small number of registers due to the lack of available free registers.

The adjacent_rFO_freereg scheme can compensate these two disadvantages. The combined scheme can realize the aim of the adjacent PE forwarding network configuration, which is to process the instructions following the instruction whose rFO exceeds one in parallel and to reduce the communication delay due to the lack of available registers. Therefore, we can understand the clustered architecture with the adjacent PE forwarding network and adjacent_rFO_freereg steering scheme is an effective design.

4. Related Work

Dynamically-scheduled clustered architectures can utilize information such as dependency and instruction status in order to distribute instructions into the suitable PEs. There are many proposals for instruction steering schemes and their comparisons in literature [1], [14]. The most basic instruction steering scheme for clustered architecture is dependence-based scheme [13]. This scheme assigns dependent instructions to the same PE. Therefore, this can minimize inter-PE communication penalties. However this suffers from load imbalance among the PEs.

Parcerisa and Gonzalez proposed a steering scheme of data dependence-based steering with load balancing named Advanced RMBS [5], [14] (In the papers, the dependence-based scheme is called RMBS, Register Mapping Based

Scheme). This scheme can improve the workload balance while this might increase undesirable communications among dependent instructions, which degrade the performance. To avoid both the undesirable inter-PE communications and load imbalance, a steering scheme based on the status of operands, which is named !ready (not ready), has been proposed [16]. This scheme can reduce the undesirable inter-PE communication with superior workload balance than the Advanced RMBS scheme.

To achieve higher performance on a clustered architecture, Aggarwal and Franklin proposed instruction replication [2]. Its basic idea is to selectively replicate instructions in those PEs where their results are required. The aim of the instruction replication is very similar to our work. However, our approach can realize the aim in much simpler way since it forwards only required results to the adjacent PE after the productions, in contrast with the replication approach, where required results are calculated in required PEs from the beginning of the result calculation process.

Farkas and Vranesic proposed Multicluster architecture [8] that makes use of the non-consistent RegFiles. However, Multicluster architecture must distribute instructions without run-time PE workload balance information, since allocations of PEs are determined by the compiler statically. In contrast, dynamic instruction steering can steer instructions more flexibly with keeping binary compatibility.

5. Conclusions

In this paper, we have made PEs cooperate with their neighboring PEs by means of adding communication structures between adjacent PEs in order to relax the losses due to inter-PE communication and workload imbalance in clustered architectures with non-consistent register files. The aim of this design is to process the instructions that use the same register instances in parallel and to reduce the inter-PE communication delay due to the lack of available registers. We have evaluated the performance of proposed configurations. We have found out that the forwarding networks which can forward the result to the right adjacent PE is effective in achieving higher IPC (instructions per clock) than the other structures. The reason why the adjacent forwarding network structure is effective is that it gives priority to the processing of dependencies that seriously affect the performance. We have also found that the instruction steering scheme that concerns both the register fanout and available free register could enhance the performance of the proposed architecture.

References

[1] A. Aggarwal and M. Franklin. An empirical study of the scalability aspects of instruction distribution algorithms for clustered processors. In *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software*, pages 172–179, 2001.

[2] A. Aggarwal and M. Franklin. Instruction replication: Reducing delays due to inter-pe communication latency. In *Proceedings of International Symposium on Parallel Architecture and Compiler Techniques*, pages 46–55, 2003.

[3] D. Burger and T. M. Austin. The simplescalar tool set, version 2.0. *Compututer Architecture News*, 25(3):13–25, 1997.

[4] J. A. Butts and G. S. Sohi. Characterizing and predicting value degree of use. In *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, pages 15–26, 2002.

[5] R. Canal, J.-M. Parcerisa, and A. González. A cost-effective clustered architecture. In *Proceedings of the 1999 International Conference on Parallel Architectures and Compilation Techniques*, pages 160–168, 1999.

[6] R. Desikan, D. Burger, and S. W. Keckler. Measuring experimental error in microprocessor simulation. In *Proceedings of the 28th annual international symposium on Computer architecture*, pages 266–277, 2001.

[7] K. Farkas, N. Jouppi, and P. Chow. Register file design considerations in dynamically scheduled processors. In *High-Performance Computer Architecture*, pages 40–51, 1995.

[8] K. I. Farkas, P. Chow, N. P. Jouppi, and Z. Vranesic. The multicluster architecture: reducing cycle time through partitioning. In *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*, pages 149–159, 1997.

[9] J. L. Henning. SPEC CPU2000: Measuring CPU Performance in the new millennium. *IEEE Computer*, 33(7):28–35, 2000.

[10] R. E. Kessler. The alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, 1999.

[11] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. MediaBench: a tool for evaluating and synthesizing multimedia and communicatons systems. In *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*, pages 330–335, 1997.

[12] J. Llosa, M. Valero, and E. Ayguade. Non-consistent dual register files to reduce register pressure. In *Proceedings of the 1st IEEE Symposium on High-Performance Computer Architecture*, pages 22–31, 1995.

[13] S. Palacharla, N. P. Jouppi, and J. E. Smith. Complexity-effective superscalar processors. In *Proceedings of the 24th annual international symposium on Computer architecture*, pages 206–218, 1997.

[14] J.-M. Parcerisa and A. González. Reducing wire delay penalty through value prediction. In *Proceedings of the 33rd annual international symposium on Microarchitecture*, pages 317–326, 2000.

[15] K. Sankaralingam, R. Nagarajan, S. Keckler, and D. Burger. A technology-scalable architecture for fast clocks and high ILP. In *5th Workshop on the Interaction Between Compilers and Computer Architectures (INTERACT-5)*, 2001.

[16] Y. Sato, K. Suzuki, and T. Nakamura. An operand status based instruction steering scheme for clustered architectures. In *Proceedings of the 2005 International Conference on Computer Design (CDES'05)*, pages 168–174, 2005.