Systems and Computers in Japan, Vol. 34, No. 12, 2003
Translated from Denshi Joho Tsushin Gakkai Ronbunshi, Vol. J85-D-I, No. 12, December 2002, pp. 1103–1113

Combining Variable Latency Pipeline with Instruction Reuse for Execution Latency Reduction

Toshinori Sato and Itsujiro Arita

Department of Artificial Intelligence, Kyushu Institute of Technology, Iizuka, 820-8502 Japan

SUMMARY

Operand bypass logic is likely to be one of the critical structures for future microprocessors to achieve high clock speed. The logic delay causes the execution time budget to be reduced significantly, so that the execution stage is divided into several stages. The variable latency pipeline (VLP) structure has the advantages of pipelining and pseudo-asynchronous design. According to the source operands delivered to arithmetic units, VLP changes execution latency and thus achieves both high speed and low latency for most operands. In this paper we evaluate VLP with dynamically scheduled superscalar processors, using a cycle-by-cycle simulator. Our experimental results show that VLP successfully reduces the effective execution time, and thus relaxes the constraints on the operand bypass logic. We also evaluate the instruction reuse technique in order to support VLP. © 2003 Wiley Periodicals, Inc. Syst Comp Jpn, 34(12): 11–21, 2003; Published online in Wiley InterScience (www.interscience.wiley.com). DOI 10.1002/scj.10498

Key words: instruction level parallelism; superscalar processors; operand bypass logic; variable latency pipeline structure; instruction reuse.

Contract grant sponsor: Supported in part by Grant #13558030 from the Japan Society for the Promotion of Science.

1. Introduction

Future microprocessors are likely to rely on higher clock speeds, wider instruction issue width, deeper pipeline stages, and larger instruction windows in order to improve performance. As the width and size increase, scaling of the clock speed becomes difficult to attain. The execution stage consists of execution latency and resulting bus drive time for operand bypassing [6]. Recently, the wire delay has become larger than the gate delay, and thus it is expected that the delay of longer bypass logic will be serious. For example, Horowitz and colleagues [8] predict that the FO4 delay of a wire crossing a 50K-gate circuit will become 2.8 times as large and that the FO4 delay of a wire connecting the two sides of a chip will become 153 times as great when the process technology improves from 0.25 to 0.05 μm. The delay of the bypass logic causes the execution time budget to be reduced significantly, so that the execution stage is divided into several stages. This is because it is difficult to design ALUs which can execute in one clock cycle [12]. Pipelining is one of the techniques for realizing high-speed circuits, and it can improve the throughput of a function. However, deep pipelines cannot improve processor performance if they entail a longer branch misprediction penalty or the disallowing of back-to-back bypassing of dependent instructions. Therefore, careful consideration is required when applying pipelining to bypass logic. Thus, the ALUs must be faster. Asynchronous and pseudo-asynchronous circuits are other techniques for realizing high-speed circuits, but they have the following problems. The asynchronous and pseudo-asynchronous circuits need a completion detector for each operation. The detector be-

comes the critical path of the circuits and increases the processor cycle time. Furthermore, the throughput can be diminished for specific operands. From these considerations, techniques to reduce execution latencies including operand bypassing are required. The variable latency pipeline (VLP) structure [10] was proposed in order to realize gigahertz-class microprocessors in 1996. It is a microarchitectural technique which has the advantages of pipelining and pseudo-asynchronous design. VLP is applicable to all logic circuits as well as ALUs. This design technique exploits the fact that the longest path for an individual operation of a logic circuit is generally much shorter than the critical path of the circuit. Furthermore, it utilizes the fact that the source operands which govern the critical path are limited to a few variations. Considering the characteristics of logic circuits and their critical paths, the circuits could be designed so that the longest path governed by most of the operands for a function is shorter than the expected processor cycle time. In this paper, *operation* and *function* are used properly as follows. A function is one of the mainpulations such as addition and subtraction. An operation is an individual function executed with its own operands. For operations which cannot be executed in one cycle, pipelined circuits are provided, but their execution latencies are increased. This combination of two kinds of circuits makes possible high-speed, short-latency execution for most of the operations. Kondo and colleagues evaluated the usefulness of VLP on a platform of an in-order execution scalar processor and found a 30% performance improvement [10], but the improvement is not clear for instruction-level parallel (ILP) processors. Thus, in this paper, we will apply VLP to ILP processors [17] and will evaluate its usefulness based on detailed simulations. One of the other techniques for execution latency reduction is instruction reuse [3, 16, 21]. Instruction reuse is a technique which eliminates redundant computations by utilizing instruction redundancy. Instruction redundancy is a characteristic of programs: dynamic instances of a static instruction are executed with the same operand values many times. Thus, if the previous computation is kept in a table, the next identical computation can be eliminated by table lookup. One example of the exploitation of instruction reuse is the result cache [16]. In this paper, in order to enhance VLP, we propose to utilize the instruction reuse technique for operations which cannot be covered by VLP. The combination of VLP and instruction reuse is also evaluated using a simulator.

This paper is organized as follows. In Section 2 the evaluation methodology is described. Section 3 evaluates the impact of long execution latency on processor performance. Section 4 explains VLP and instruction reuse. Section 5 presents simulation results. In Section 6, previously proposed related work is surveyed. Section 7 concludes the paper.

2. Evaluation Methodology

This section introduces our processor model and benchmark programs.

2.1. Processor model

An execution-driven simulator which models wrong path execution caused by branch misprediction is used for this study. We implemented the simulator using the SimpleScalar tool set (version 2.0) [2]. The SimpleScalar/PISA instruction set architecture (ISA) is based on MIPS ISA.

The simulator models realistic 4- and 8-way out-of-order execution superscalar processors based on register update units (RUU) [22] which have 64 and 128 entries, respectively.

Each functional unit can execute all functions. The latency for execution is 1 cycle except in the cases of multiplication (4 cycles) and division (12 cycles).

A 4-port, nonblocking, 128KB, 32B block, 2-way set-associative L1 data cache is used for data supply. It has a load latency of 1 cycle after the data address is calculated and a miss latency of 6 cycles. It has a backup 8MB, 64B block, direct-mapped L2 cache which has a miss latency of 18 cycles for the first word plus 2 cycles for each additional word. A memory operation that follows a store whose data address is unknown cannot be executed. A 128KB, 32B block, 2-way set-associative L1 instruction cache is used for instruction supply and also has a backup L2 cache which is shared with the L1 data cache. The hit latency of 1 cycle is smaller than that of modern microprocessors such as the Intel Pentium 4 and HP 21264 [7, 9], and thus it should be noted that it may emphasize the influence of ALU latency. However, several techniques for cache access latency reduction have been used, such as load address prediction [5] and load value prediction [11].

For control prediction, a 1K-entry 4-way set-associative BTB, a 4K-entry gshare-type 2-level adaptive branch predictor [13], and an 8-entry return address stack are used. The BTB keeps only taken branch instructions. The branch predictor is updated at the instruction commit stage.

2.2. Benchmark programs

The SPECint95 benchmark suite is used for this study. The `test` input files provided by SPEC are used. Table 1 lists the benchmarks and the input sets. We used the object files provided by the University of Wisconsin at Madison [2], except for `132.ijpeg` which was compiled by GNU GCC (version 2.6.3) with the optimization option `-O3`. Each program is executed to completion or for the first 100 million instructions. We count only committed instructions.

Table 1. Benchmark program and input file

program	input file
099.go	null.in
124.m88ksim	ctl.in
126.gcc	cccp.i
129.compress	test.in
130.li	test.lsp
132.ijpeg	specmun.ppm
134.perl	primes.in
147.vortex	vortex.in

We focus on the performance of only integer programs because it tends to be more difficult to obtain high levels of parallelism in such programs than in floating-point programs.

3. Impact of Long Latency

This section evaluates the impact of long execution latency on processor performance. In order to evaluate the impact, the latency for ALU execution other than cases of multiplication and division is increased to two cycles from one cycle. The ALU operation includes address calculation of load and store instructions and conditional branch instructions. The latency of logical instructions including shift instructions is maintained as one cycle.

Table 2 presents the percentage of operations executed in two latencies. Note that it is correct that 4- and 8-way processors have different percentages. This is because instructions flushed by branch misprediction are considered in calculating the percentages. As can be easily observed, a significant part of the operations (70 to 80%) falls into the class executed in two cycles. Since most of the operations are executed in two cycles, it is expected that the throughput of the ALUs will become lower and thus that processor performance will be degraded even if instructions are scheduled in out-of-order fashion.

Table 2. Two-cycle latency operations (%)

program	percentage	
	4-way	8-way
099.go	73.22	73.26
124.m88ksim	74.00	73.73
126.gcc	83.78	83.58
129.compress	88.82	87.63
130.li	85.18	84.58
132.ijpeg	80.59	79.99
134.perl	81.70	80.84
147.vortex	89.29	89.07

Figure 1 shows the impact of the long execution latency on processor performance. We use committed instructions per cycle (IPC) as a metric for evaluating performance. The processor performance is normalized to that of a processor whose ALU latency is always one. Thus, the lower the bar, the more severe the performance degradation is. For each group of two bars, the left bar indicates the performance of the 4-way processor and the right bar that of the 8-way processor. Note that the bars for 4- and 8-way processors cannot be compared with each other: they indicate only the difference from the baseline processor models. The following observations are made.

First, the processor performance is significantly degraded. Second, the impact of long latency is generally much more severe for an 8-way processor than for a 4-way processor. The degradation is as much as 30%. This is because the 8-way processor can execute more instructions in parallel than the 4-way processor and because the demand for scheduling becomes higher. In addition, because the instruction issue width is larger, the branch resolution time which is longer than that of the baseline model has a serious impact on processor performance.

Third, there is no explicit relationship between the percentage of two cycle latency operations shown in Table 2 and the performance degradation rate. For example, the percentage of `129.compress` is larger than that of `132.ijpeg` but `132.ijpeg` is much more severely affected by the long latency than `129.compress`.

Branch prediction plays one of the essential roles in modern superscalar processors. Table 3 summarizes branch prediction accuracy. The first column gives the program name. The remaining four groups of three columns indicate the results for the 4-way baseline, 4-way two-latency, 8-way baseline, and 8-way two-latency models, respectively.

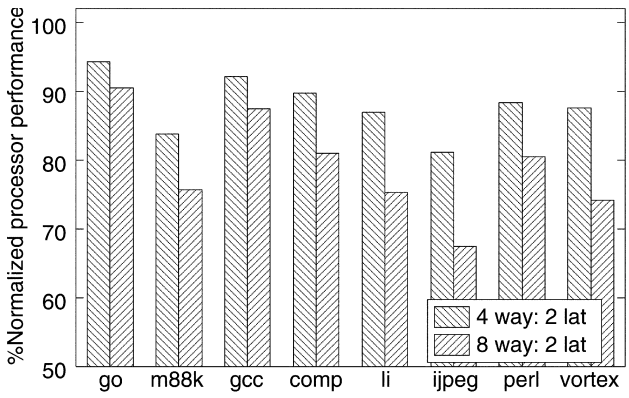

Fig. 1. Impact of long latency (%).

Table 3. Branch prediction accuracy (%)

	4-way						8-way					
	latency = 1			latency = 2			latency = 1			latency = 2		
program	addr	dir	jr	addr	dir	jr	addr	dir	jr	addr	dir	jr
099.go	68.71	72.24	56.49	68.56	72.16	55.54	67.64	71.72	49.06	67.43	71.71	46.24
124.m88ksim	91.92	95.98	45.52	91.19	95.64	40.50	90.32	95.84	26.66	89.95	95.62	24.71
126.gcc	78.78	83.30	50.33	78.30	83.08	47.87	76.80	82.53	38.66	76.66	82.64	36.21
129.compress	94.85	96.49	77.55	94.21	96.34	70.74	92.69	96.34	49.12	91.97	95.95	44.70
130.li	87.73	93.78	57.70	86.33	94.02	46.25	85.44	93.41	44.30	83.65	92.89	35.45
132.ijpeg	89.67	90.24	80.65	89.46	90.10	77.91	89.17	89.95	72.83	88.80	90.00	57.09
134.perl	88.74	95.76	44.11	87.95	95.56	39.59	86.26	95.74	25.27	85.01	95.48	17.74
147.vortex	85.11	92.68	37.72	84.06	92.72	28.71	82.39	92.40	17.48	83.11	92.68	21.14

For each group, the three columns present the accuracy of target address prediction (`addr`), that of branch direction prediction (`dir`), and that of indirect jump address prediction (`jr`), respectively. Note that the prediction accuracy is not the BTB prediction accuracy, but the prediction accuracy of branch direction. Thus, the former is always smaller than the latter. For most of the cases other than `147.vortex` on the 8-way processor, the target address and branch direction prediction accuracy is slightly reduced as the ALU latency increases. Even a slight decrease of branch prediction accuracy is reported to severely affect processor performance. Thus, the results indicate a negative impact on processor performance. Furthermore, the jump address prediction accuracy is considerably degraded. In summary, a serious impact on processor performance is produced if the branch resolution time is late due to long-latency ALUs.

4. Variable Latency Pipeline Structure

The previous section revealed that the impact of long execution latency on processor performance is very severe. This section proposes two solutions.

4.1. VLP technique

Figure 2 shows the concept of the VLP [10]. A function can be implemented by several kinds of circuits whose design policies differ from each other. In Fig. 2, two circuits are used for implementing the function. Circuit A is designed so that most of the longest path of each operation is shorter than a processor cycle time. Circuit B is designed so that the critical path of the circuit is shorter than the cycle time, and is pipelined. Combining these two kinds of circuits reduces the effective latency of function execution and also maintains the throughput of the function even for operations which are executed in two cycles. The execution stage consists of the execution latency and the result drive time for operand bypassing [6]. Since the effective execution latency can be reduced by VLP, the constraints on the operand bypass logic are considerably relaxed.

In order to select one of two results, a completion detector for circuit A is required. The detector can be pipelined, since pipelined circuit B works as the backup of circuit A. Thus, the detector does not increase the critical path. From the above explanation, it can be seen that a high-speed, short-latency function is implemented. It is true that the total transistor count is increased in order to implement two circuits. However, it is possible to reduce the count if circuits A and B share circuitry.

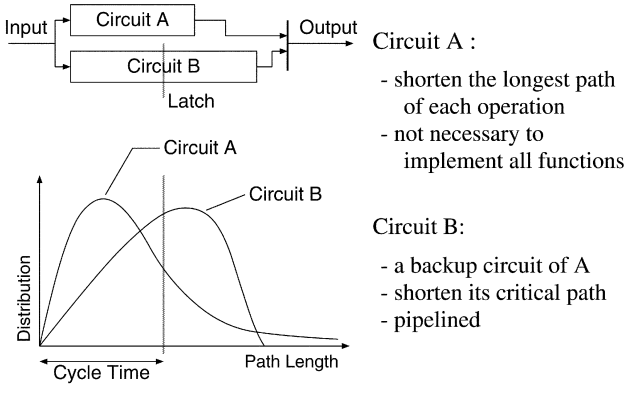

Fig. 2. Variable latency pipeline structure.

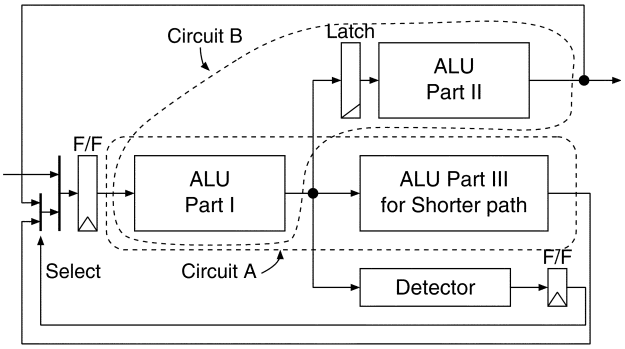

Fig. 3. ALU utilizing VLP.

Figure 3 shows an example of VLP applied to an ALU [10]. There are two kinds of circuits. One is a two-stage pipelined circuit consisting of ALU parts I and II, which is used for high speed and high throughput. This circuit is used as circuit B in Fig. 2. The other consists of ALU parts I and III and is used as circuit A in Fig. 2. That is, its execution latency is 1. As can be seen, ALU part I is shared by two circuits. Circuit A in Fig. 3 is designed so that no carry propagation over 16 bits is considered [10]. If the carry propagation for an operation is less than 16 bits, circuit A consisting of ALU parts I and III supplies its correct result, and most of the operations are included in this case. When the propagation is larger than 16 bits, circuit A may generate a wrong result and the circuit B acts as its backup. Across SPECint95 benchmark programs, over 50% of integer operands are 16 bits or less [1], and thus the percentage of carry propagations larger than 16 bits is even smaller. The correct result is obtained in two cycles. Therefore, there is a completion detector for circuit A which selects one of the two results. As described above, the detector has no impact on the critical path.

Instruction scheduling is handled as follows. Most schedulers rely on the knowledge that a simple ALU operation will always take one cycle, and they perform wakeup and selection with this knowledge in advance. VLP breaks this assumption, since now some ALU operations take one cycle and some take two cycles. In VLP, the execution latency of the ALU operations is assumed to be one cycle. Therefore, when an instruction is executed, dependent instructions will be scheduled or may already be scheduled. If the instruction in the ALU takes two cycles, the dependent instructions must be rejected or eliminated from the functional unit and rescheduled. This process is easily implemented. Since the execution latency is at most two cycles, the height of dependent instructions is 1. There are no instructions which are indirectly dependent on any two-cycle latency operations and are already scheduled. That is, only instructions which are directly dependent on and located immediately after two-cycle latency operations may violate data dependencies. These instructions can be easily detected and invalidated. Rescheduling the instructions is also easy. Since functional units are already assigned to the instructions, each of them only has to be executed again using the correct operands on the same functional unit to which it has already been assigned. This scheduling scheme resembles that implemented in modern processors [7, 9] and thus is likely to be practical.

4.2. Result cache

The result cache [16] has a structure similar to cache memories and keeps calculations once executed. Figure 4 depicts a direct-mapped result cache. It is provided for each function, is indexed by operand values, and supplies the execution result of the operation. Since the index to the result cache includes operand values, its execution result is not available until the operands are obtained.

In order to enhance VLP, we propose to utilize the instruction reuse technique for operations which are executed in two cycles. Since candidates for instruction reuse are limited, the utilization of the result cache might be improved. Figure 5 explains the case in which the result cache is attached to the VLP ALU. When an operation and its operand values are provided to the ALU, they also index the result cache in parallel. Most of the time, circuit A supplies correct results, just as in VLP. When circuit A cannot supply a correct result for an operation, it is supported by the result cache. Its correct result is obtained from the result cache if the access is a hit. In such a case, the execution latency is reduced to 1 even though circuit A is not applicable. When a result cache miss occurs, the execution result from circuit B is used. As described above, the

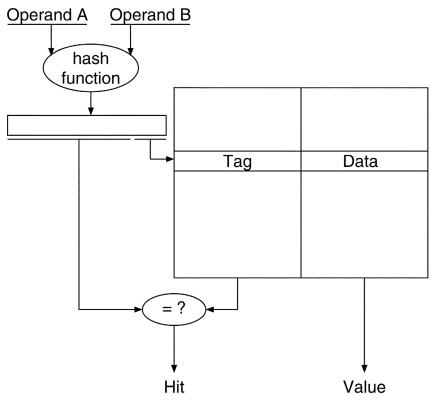

Fig. 4. Result cache.

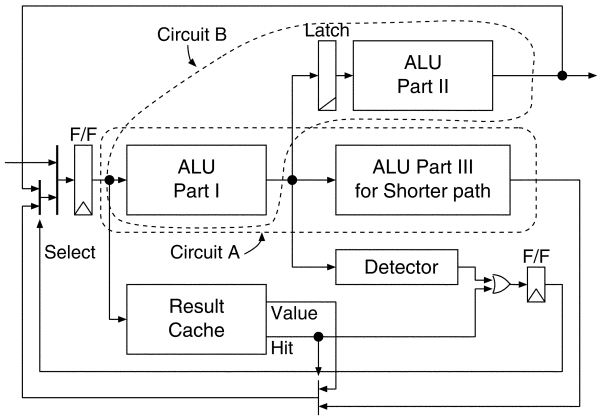

Fig. 5. Enhancing VLP using result cache.

effective latency is further reduced from the case in which only VLP is utilized.

Only operations which cannot be covered by VLP are held in the result cache. This registration policy limiting the candidates for the result cache is likely to improve its utilization. Thus, the result cache hit rate is likely to increase, improving processor performance more efficiently. Note that it is impossible to keep all cases which cause two-cycle latency in a lookup table because there are many combinations which produce carry propagation of over 16 bits. However, the execution results have locality [11] and thus the result cache considered in this paper can store useful combinations efficiently.

5. Simulation Results

This section presents simulation results.

5.1. Effectiveness of VLP

We evaluate the VLP integer ALUs [10] explained in Section 4 by comparing them with ALUs whose execution latencies are one and two cycles, respectively. That is, we focus on the execution stage of the integer pipelines, considering the operand bypass logic. The latency of the ALU utilizing VLP is 2 cycles when a carry propagation of over 16 bits occurs. Otherwise it is 1 cycle. Candidate functions for VLP include not only arithmetic and logical instructions but also load, store, and branch instructions. Multipliers and dividers do not utilize the VLP technique. The latency for multiplication is 4 cycles and that for division is 12 cycles.

Table 4 shows the percentage of operations whose execution latencies are reduced by VLP. That is, it presents the VLP hit ratio. Note that it is correct that 4- and 8-way processors have different VLP hit ratios. This is because

Table 4. VLP hit ratio (%)

program	hit ratio	
	4-way	8-way
099.go	82.00	81.84
124.m88ksim	88.10	88.04
126.gcc	88.86	89.04
129.compress	29.14	33.87
130.li	81.23	81.99
132.ijpeg	93.08	93.02
134.perl	92.28	92.86
147.vortex	87.02	86.93

instructions flushed due to branch misprediction are considered for calculating the hit ratios. For most of the programs, approximately 90% of the two-cycle latency operations are executed in one cycle by utilizing the VLP. Therefore, less than 10% of operations must be executed in two cycles. The effective latency is reduced and the performance degradation will be compensated. For `129.compress`, the VLP hit ratio is considerably lower than the others, and hence the effectiveness of VLP is likely to be smaller than for the other programs.

The reason why the VLP hit rate is small in the case of `129.compress` is as follows. The execution of `129.compress` is dominated by a function *cl_hash()*, which occupies 61.3% of all executions. In *cl_hash()*, an integer array which has 69001 values is cleared. This process includes many store operations which involve carry propagations of over 16 bits when calculating data addresses. In addition, there are many subtracts with carry propagations of over 16 bits when decrementing index variables.

Figure 6 depicts the impact of VLP on processor performance. For each group of four bars, the first two bars indicate the results for the 4-way processor and the remain-

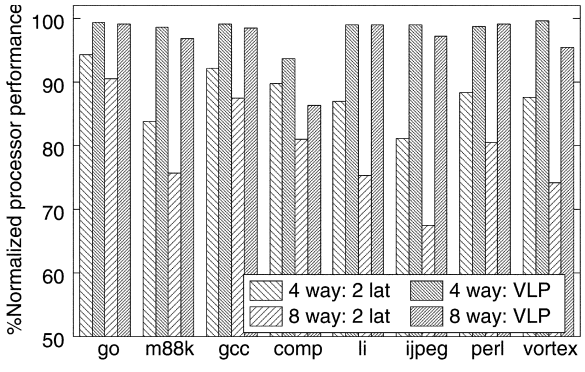

Fig. 6. Effect of VLP (%).

ing two bars indicate those for the 8-way processor. The left bar is for a processor consisting of two-latency pipelined ALUs. The right bar is for that consisting of the variable latency pipelined ALUs. We can make the following observations. First, when utilizing VLP, the processor performance for most of the programs is comparable to that of the baseline processors. This indicates that VLP is effective for ILP processors. Second, for `129.compress` there is still a significant difference between the performance of the processor consisting of one-cycle-latency ALUs and that of the processor consisting of VLP ALUs. This is because the VLP hit ratio for the program is very small. Third, for `147.vortex`, the VLP hit ratio is relatively high and the percentage of two-cycle-latency operations is also large. Thus, the effect of VLP becomes moderate. In summary, the performance gain is 13.8 and 15.0% on average for the 4- and 8-way processor, respectively, when VLP is utilized.

Table 5 summarizes the branch prediction accuracy. The layout and content of Table 5 are the same as for Table 3. The first column shows the program name. The remaining four groups of three columns give the results for the 4-way baseline, 4-way VLP, 8-way baseline, and 8-way VLP models, respectively. As easily seen, the branch prediction accuracy is considerably improved over two-cycle-latency models except for `147.vortex` on the 8-way processor, especially for jump address prediction. The results also confirm the usefulness of VLP.

5.2. Effectiveness of result cache

We have seen that VLP does not always bridge the performance gap between one- and two-cycle-latency pipelined ALUs. In this section, we evaluate the result cache.

We implemented the result cache in the simulator explained in Section 2. It is assumed that the result cache is fully associative and has 64 entries. It is also assumed that the result cache is shared by all functions, and thus it is indexed by opcodes as well as operand values.

Table 6. R-cache hit rate (%)

program	hit rate	
	4-way	8-way
099.go	70.53	69.41
124.m88ksim	89.21	81.73
126.gcc	52.04	51.40
129.compress	15.73	16.04
130.li	78.25	76.43
132.ijpeg	17.78	20.29
134.perl	54.76	46.12
147.vortex	24.15	24.19

Table 6 shows the result cache hit rate. Contrary to our expectation that the result cache hit rate was high, it is quite low. This is a discouraging result, especially for `129.compress`, where the VLP hit ratio is small, and thus it is expected that the contribution of the result cache to processor performance will be modest.

Figure 7 depicts the impact on processor performance. For each group of four bars, the first two bars indicate the results for the 4-way processor and the remaining two bars indicate those for the 8-way processor. The left bar is for the processor consisting of the VLP ALUs. The right bar is for that consisting of the VLP ALUs and the result cache. The primary observation is that the performance gap between the baseline model and the VLP model with the result cache is still large for `129.compress`. For the 8-way processor, it is approximately 10%, even though the result cache improves the performance efficiency by 4% over the VLP model. Since the result cache hit rate for `129.com-`

Table 5. Branch prediction accuracy (VLP) (%)

	4-way						8-way					
	latency = 1			variable latency			latency = 1			variable latency		
program	addr	dir	jr	addr	dir	jr	addr	dir	jr	addr	dir	jr
099.go	68.71	72.24	56.49	68.64	72.18	56.44	67.64	71.72	49.06	67.64	71.72	48.92
124.m88ksim	91.92	95.98	45.52	91.91	96.00	45.33	90.32	95.84	26.66	90.11	95.77	24.85
126.gcc	78.78	83.30	50.33	78.75	83.30	50.09	76.80	82.53	38.66	76.80	82.55	38.43
129.compress	94.85	96.49	77.55	94.74	96.63	74.07	92.69	96.34	49.12	92.48	96.22	47.98
130.li	87.73	93.78	57.70	87.64	93.99	55.63	85.44	93.41	44.30	84.82	93.59	38.73
132.ijpeg	89.67	90.24	80.65	89.67	90.24	80.50	89.17	89.95	72.83	89.15	89.94	72.13
134.perl	88.74	95.76	44.11	88.53	95.57	43.91	86.26	95.74	25.27	86.19	95.69	25.18
147.vortex	85.11	92.68	37.72	85.35	92.87	38.16	82.39	92.40	17.48	82.37	92.48	16.69

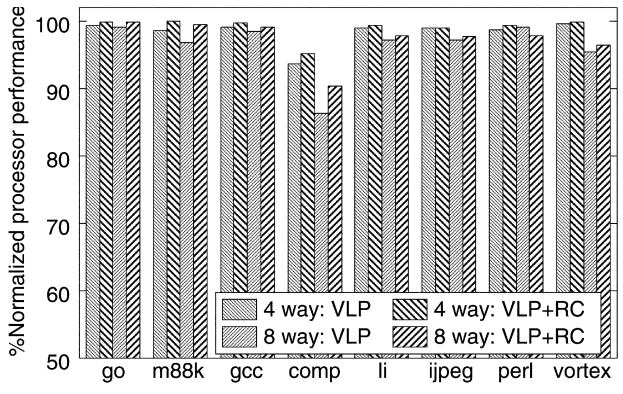

Fig. 7. Effect of result cache (%).

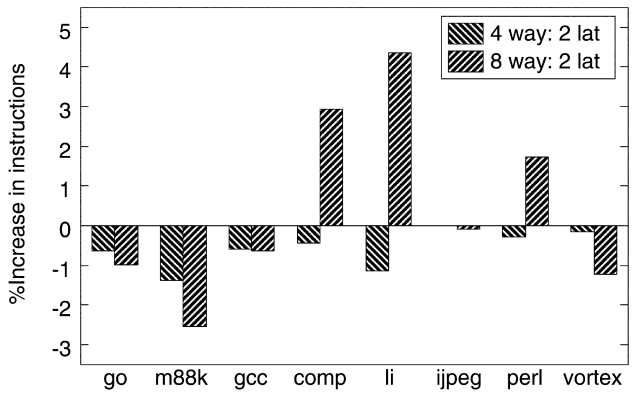

Fig. 8. Increase of total instruction count (%).

`press` is smallest, this simulation result matches our expectation.

The second observation is that the result cache enhances VLP in most cases. The 4- and 8-way processors gain 1.25 and 1.12% on average, respectively. The only exception is `134.perl` on the 8-way processor. The reason is as follows. Figure 8 shows the increase in the total instruction count, which includes instructions flushed by branch misprediction when the result cache is attached. We define the increase by the total number of instructions for VLP with the result cache model compared with that for the VLP model. For each group of two bars, the left bar indicates the increase of the 4-way processor and the right bar indicates that of the 8-way processor. When the effective execution latency is reduced, the number of instructions executed while branch misprediction is unresolved is increased. On the other hand, the branch resolution time is also reduced. Thus, most of the time, the total instruction count can be reduced. From Fig. 8, in only the cases of `132.ijpeg` on the 4-way processor, and `129.compress`, `130.li`, and `134.perl` on the 8-way processor, the total instruction increases. For `132.ijpeg`, the increase is small, and thus it does not affect processor performance. For `130.li`, the result cache hit rate is large, and thus its contribution to processor performance is bigger than the negative impact of the increase in the total number of instructions. For `129.compress` and `134.perl`, the increase severely affects performance. Thus, the processor performance of `134.perl` is degraded. In any case, the result cache makes a modest contribution in supporting the VLP technique.

Table 7 summarizes branch prediction accuracy. The layout and content of Table 7 are the same as for Table 3. The first column shows the program name. The remaining four groups of three columns indicate the results for the 4-way VLP, the 4-way VLP with the result cache, the 8-way VLP, and the 8-way VLP and the result cache models,

Table 7. Branch prediction accuracy (VLP + RC) (%)

	4-way						8-way					
	variable latency			VLP + RC			variable latency			VLP + RC		
program	addr	dir	jr	addr	dir	jr	addr	dir	jr	addr	dir	jr
099.go	68.64	72.18	56.44	68.70	72.23	56.52	67.64	71.72	48.92	67.65	71.72	49.00
124.m88ksim	91.91	96.00	45.33	91.93	95.99	45.61	90.11	95.77	24.85	90.30	95.81	26.84
126.gcc	78.75	83.30	50.09	78.77	83.30	50.25	76.80	82.55	38.43	76.80	82.53	38.69
129.compress	94.74	96.63	74.07	94.80	96.46	77.23	92.48	96.22	47.98	92.73	96.24	51.20
130.li	87.64	93.99	55.63	87.79	93.85	57.59	84.82	93.59	38.73	85.30	93.43	43.18
132.ijpeg	89.67	90.24	80.50	89.65	90.22	80.57	89.15	89.94	72.13	89.15	89.94	72.64
134.perl	88.53	95.57	43.91	88.53	95.58	43.90	86.19	95.69	25.18	86.26	95.76	25.14
147.vortex	85.35	92.87	38.16	85.28	92.81	38.07	82.37	92.48	16.69	82.39	92.46	17.07

respectively. It is found that the contribution to branch prediction accuracy is also modest.

6. Related Work

Oberman and Flynn [14] developed a variable latency pipelined floating-point adder. In order to achieve maximum system performance, the latency of the floating-point adder becomes variable according to its source operands. This reduces the effective latency while the throughput of the adder is maintained. The effective latency is evaluated using the SPECfp92 benchmark suite, but its impact on processor performance is not evaluated.

Kondo's group [10] proposed the VLP structure for integer ALUs. Using properly two kinds of circuits according to the longest path of the circuits for each operation, the effective execution latency can be almost always made to be one cycle, while the processor cycle time reaches 1 GHz. Using the SPECint92 benchmark suite, the usefulness of VLP was confirmed on a platform using an in-order execution scalar processor, but it is not clear for dynamically scheduled superscalar processors.

Richardson proposed a result cache [16] which eliminates redundant calculations appearing repeatedly in floating-point units (FPUs). The result cache registers a floating-point operation and its result. When the same operation emerges, its execution results can be obtained without any calculation in the FPU. Since the latency of the FPU is generally very long, quick calculation using the table lookup will improve processor performance. Citron et al. [3] have evaluated the use of the result cache in multimedia applications.

Oberman et al. [15] have proposed an extension of the result cache called the reciprocal cache. This cache stores reciprocals (1/N) and converts divisions into multiplications. Ordinary floating point dividers, in which pipelined operation cannot be performed, have very long latency, while floating point multipliers that use pipelining have much shorter latency. Thus the reciprocal cache shortens the latency of division operations and consequently increases the throughput.

The reuse buffer proposed by Sodani and Sohi [21] caches all types of operation results, while the result cache holds only the results of floating-point operations. It is indexed by an instruction address, while the result cache is indexed by operand values, and thus it can supply execution results at earlier pipeline stages than the result cache. Therefore, the reuse buffer is useful even for operations whose latencies are 1. In other words, the effective latency can be less than 1.

Sazeides's group [20] evaluated collapsing of data dependences. Two operations are combined into one operation and thus its data dependence is reduced. In other words, the execution latency of a pair of operations is reduced. Collapsing increases the longest path of circuits and its application is limited to simple functions.

The Pentium 4 [7] calculates two ALU operations in one system clock cycle. This is made possible for ALUs by performing a less-than-16-bit operation in half as many clock cycles. That is, the upper and lower 16-bit operations in one ALU operation are executed in pipelined fashion. This strategy shares with VLP the observation that there are few more-than-16-bit operations in typical programs.

Value prediction [4, 11] also has the ability to reduce the effective execution latency. Using the history of each operation, its execution result is predicted before it is dispatched to a functional unit. Operations dependent on the predicted operation use the predicted value. It should be noted that value prediction is different from the techniques explained above in the sense that its execution is speculative. Thus, it is necessary to recover the processor state when a misspeculation occurs.

To the best of our knowledge, VLP has not been evaluated on ILP processors. In addition, combining VLP and the instruction reuse technique has not been considered. In this paper, we have evaluated the combination of these techniques on ILP processors.

7. Conclusion

Orand bypass logic is likely to be one of the critical structures for future microprocessors to achieve high clock speed. The delay of the bypass logic might cause the execution time budget to be reduced significantly, so that the execution stage is divided into several stages. In this paper, we have evaluated VLP integer ALUs as a means of dealing with this problem. We have also proposed to combine the result cache with the VLP structure and have evaluated the combination. The findings of this paper are as follows.

- VLP is useful for alleviating constraints on the bypass logic by reducing the effective execution latencies.
- The combination of VLP and the result cache makes a modest contribution to improvement of processor performance over the VLP technique alone.

One of the future areas of research dealing with VLP is exploiting speculative execution. We are currently investigating constructive timing violation (CTV) [18, 19]. In CVT, we propose to give up meeting timing constraints, but to tolerate them. We have already evaluated the use of CVT to improve processor performance [18] and energy efficiency [19], and have obtained encouraging results.

Acknowledgments. The authors are grateful to Dr. Yoshihisa Kondo at Toshiba Microelectronics Engineering Laboratory for his comments on earlier drafts. Parts of this work were performed while Toshinori Sato was with the laboratory. This work is supported in part by Grant #13558030 from the Japan Society for the Promotion of Science.

REFERENCES

1. Brooks D, Martonosi M. Dynamically exploiting narrow width operands to improve processor power and performance. 5th Int Symposium on High Performance Computer Architecture, 1999.
2. Burger D, Austin TM. The SimpleScalar tool set, version 2.0. ACM SIGARCH Computer Architecture News 1997;25(3).
3. Citron D, Feitelson D, Rudolph L. Accelerating multi-media processing by implementing memoing in multiplication and division units. Int Conference on Architectural Support for Programming Languages and Operation Systems VIII, 1998.
4. Gabbay F, Mendelson A. Using value prediction to increase the power of speculative execution hardware. ACM Trans Computer Syst 1998;16(3).
5. Gonzalez J, Gonzalez A. Speculative execution via address prediction and data prefetching. 11th Int Conference on Supercomputing, 1997.
6. Hara T, Ando H, Nakanishi C, Nakata M. Performance comparison of ILP machines with cycle time evaluation. 23rd Int Symposium on Computer Architecture, 1996.
7. Hinton G, Sager D, Upton M, Boggs D, Carmean D, Kyker A, Roussel P. The microarchitecture of the Pentium 4 processor. Intel Tech J, issue Q1, 2001.
8. Horowitz M, Ho R, Mai K. The future of wires. SRC Workshop, 1999.
9. Kessler RE, McLellan EJ, Webb DA. The Alpha 21264 microprocessor architecture. Int Conference on Computer Design, 1998.
10. Kondo Y, Ikumi N, Ueno K, Mori J, Hirano M. An early-completion-detecting ALU for a 1GHz 64b datapath. Int Solid State Circuit Conference, 1997.
11. Lipasti MH, Wilkerson CB, Shen JP. Value locality and load value prediction. Int Conference on Architectural Support for Programming Languages and Operation Systems VII, 1996.
12. Liu T, Lu S-L. Performance improvement with circuit-level speculation. 33rd Int Symposium on Microarchitecture, 2000.
13. McFarling S. Combining branch predictors. WRL Technical Note TN-36, Digital Western Research Laboratory, 1993.
14. Oberman SF, Flynn MJ. A variable latency pipelined floating-point adder. 2nd Int Euro-Par Conference, 1996.
15. Oberman SF, Flynn MJ. Reducing division latency with reciprocal caches. Reliable Computing 1996;2(2).
16. Richardson SE. Exploiting trivial and redundant computation. 11th Int Symposium on Computer Arithmetic, 1993.
17. Sato T, Arita I. Execution latency reduction via variable latency pipeline and instruction reuse. 7th Int Euro-Par Conference, 2001.
18. Sato T, Arita I. Give up meeting timing constraints, but tolerate violations. COOL Chips IV, 2001.
19. Sato T, Arita I. Constructive timing violation for improving energy efficiency. 2nd Workshop on Compilers and Operating Systems for Low Power, 2001.
20. Sazeides Y, Vassiliadis S, Smith JE. The performance potential of data dependence speculation & collapsing. 29th Int Symposium on Microarchitecture, 1996.
21. Sodani A, Sohi GS. Dynamic instruction reuse. 24th Int Symposium on Computer Architecture, 1997.
22. Sohi GS. Instruction issue logic for high-performance, interruptible, multiple functional unit, pipelined computers. IEEE Trans Computers 1990;39(3).

AUTHORS (from left to right)

Toshinori Sato is an associate professor in the Department of Artificial Intelligence at Kyushu Institute of Technology. He holds B.E., M.E., and Ph.D. degrees in electronic engineering from Kyoto University. From 1991 to 1999 he was with Toshiba Corporation, where he worked on the design of several embedded processors such as EmotionEngine for PlayStation2. His research interests include high-performance, energy-efficient, and complexity-effective microarchitectures and design methodologies for VLSI. He is a member of IPSJ, ACM, and IEEE-CS.

Itsujiro Arita received his Ph.D. degree from Kyushu University. He was with Kyushu University from 1965 to 1984, and currently is a professor in the Department of Artificial Intelligence at Kyushu Institute of Technology. His research interests include computer architectures, parallel processing, operating systems, and computer networks. He is a member of IPSJ and JSSST.