A Simulation Study of Pipelining and Decoupling a Dynamic Instruction Scheduling Mechanism

Toshinori Sato
Toshiba Microelectronics Engineering Laboratory
580-1, Horikawa-Cho, Saiwai-Ku, Kawasaki 210-8520, Japan
toshinori.sato@toshiba.co.jp

Abstract

As instruction window size increases, it becomes difficult to maintain processor cycle time. Pipelining the instruction window is not a solution for maintaining the processor cycle time, since it is said that it affects processor performance seriously. However, the pipelining has not been evaluated quantitatively. Therefore, in this paper, we evaluate a dynamic instruction scheduling mechanism whose wakeup and select logic is pipelined. On the other hand, recent interests on data speculation demand to further increase the instruction window size in order to realize instruction reissue mechanism which deals with incorrect data speculations. For the purpose of reducing the instruction window size with maintaining the instruction reissue capability, we propose to decoupling the reissue mechanism from the scheduling mechanism. Using a cycle-by-cycle simulator, we have evaluated the pipelining and the decoupling of the instruction window.
Keywords: instruction level parallelism, data speculation, instruction reissue, dynamic instruction scheduling, instruction window design

1 Introduction

Modern microprocessors schedule instructions dynamically in order to exploit instruction level parallelism (ILP). After instructions are fetched and decoded, they are issued into instruction window where they remain until their data dependences are resolved, and then they are executed in out-of-order. Therefore, it is necessary to increase the instruction window size for improving the instruction scheduling capability. However, it is difficult to increase the size without any serious impact on processor performance, since the instruction window is one of the dominant deciding processor cycle time [8]. Considering these situations, there are several studies decentralizing the instruction window [7,9,13]. In this paper, we focus on the centralized instruction window.

Pipelining is one of the techniques realizing the high speed circuits. It is possible to increase the instruction window size with maintaining the processor cycle time if its wakeup and select logic is pipelined. However, it is said that the pipelining of the wakeup and select logic diminishes processor performance severely [8]. To the best of our knowledge, this performance degradation has not been evaluated quantitatively. Therefore, we evaluate how the pipelining affects processor performance by using a cycle-by-cycle simulator. One of our findings is that the pipelining of the wakeup and select logic indeed diminishes processor performance severely.

On the other hand, data speculation attracts computer architects recently. The data speculation is a new technique removing serialization on a program execution caused by data dependences based on data value prediction [4,6]. When a predicted value is correct, it becomes possible to execute the predicted instruction and its dependent instructions simultaneously, thereby more ILP is extracted. Otherwise, it is necessary to revert processor state to a safe point where the speculation is initiated. Instruction reissue is such a technique recovering the processor state when a misspeculation occurs. It invalidates instructions dependent upon the misspeculated instruction selectively and then reissues them in the instruction window. It is possible to realize the instruction reissue with a simple hardware structure [10].

However, the instruction reissue technique has a problem. In order to realize the instruction reissue, each instruction remains in the instruction window until it is not speculative. In other words, it cannot release its entry in the instruction window until it is committed. Thus, effective capacity of the instruction window is reduced, diminishing the capability of the instruction scheduling [3]. In order to maintain the capability, it is necessary to increase the instruction window size. As explained above, the processor performance is degraded when the wakeup and select logic is pipelined for increasing the instruction window size without any serious impact on the processor cycle time. We have proposed decoupled instruction window to attack the problem [11]. In this paper, we also evaluate the decoupled instruction window using the cycle-by-cycle simulator.

The organization of the rest of this paper is as follows. In Section 2 the evaluation methodology is presented. Section 3 shows how the pipelining of the wakeup and select logic affects processor performance. Section 4 describes the decoupled instruction window and evaluates its usefulness. In Section 5, previously proposed related works are surveyed. Finally, our conclusions are presented in Section 6.

2 Evaluation Methodology

In this section, we describe the evaluation methodology by explaining a processor model and benchmark programs.

2.1 Experimental model

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

The simulator models a realistic 8-way out-of-order execution superscalar processor based on register update unit (RUU) [12]. Each functional unit can execute any operations. The latency for execution is 1 cycle except in the case of multiplication (4 cycles) and division (12 cycles). A 4-port, non-blocking, 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 of an 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. No memory operation can execute that follows a store whose data address is unknown. A 128KB, 32B block, 2-way set-associative L1 instruction cache is used for instruction supply and also has the backup of the L2 cache which is shared with the L1 data cache. For control prediction, a 1K-entry 4-way set associative branch target buffer, a 4K-entry gshare-type branch predictor, and an 8-entry return address stack are used. The branch predictor is updated at instruction commit stage.

Value predictor used in this study is a 4096-entry direct-mapped stride predictor [4]. Figure 1 depicts the predictor. It is indexed by instruction addresses and each entry has a tag field (`tag`), a previous value field (`prev_value`), a stride field (`stride`), and a confidence field (`conf`). The tag field is used for distinguishing individual instructions from each other. The previous value field holds the last value generated by the instruction. The stride field keeps a difference of the last two values generated by the instruction. The predicted value is produced as the sum of the previous value and the stride. The confidence filed is a 2-bit up-down saturated counter and decides if the speculation using the predicted value should be initiated. When a prediction is correct, it is incremented. Otherwise, it is decremented. When its most significant bit is 1, the speculation is initiated using the predicted value.

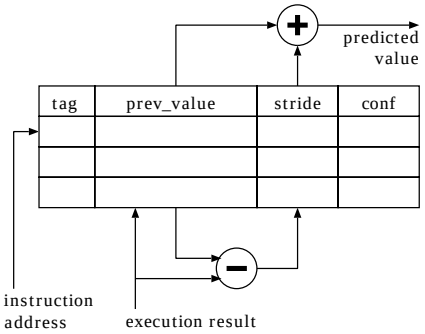


Figure 1: Stride value predictor

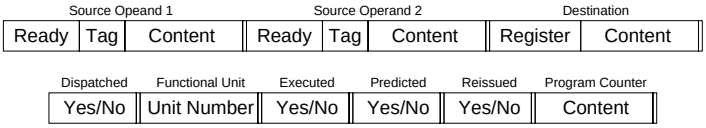


Figure 2: Extended register update unit

In order to recover the processor state when a misspeculation occurs, an instruction reissue mechanism is used. The instruction reissue re-executes only instructions dependent upon a misspeculated instruction. Those instructions are detected selectively and reissued in instruction window. We implemented an instruction reissue mechanism by extending the RUU [10]. Figure 2 shows the extended RUU. An entry of the RUU consists of two source operand fields, a destination field, a dispatched bit, a functional unit field, an executed bit, a predicted bit, a reissued bit, and a program counter field. If a source operand is not ready, the ready bit is reset to indicate that the source operand is not available and a tag for the operand is obtained. When the operand is ready, the content of the source register is held in the source operand field and the ready bit is set. The destination register number with renaming tag is held in the destination field, and an execution result is also held in the destination field when the execution completes. The dispatched bit indicates if the instruction is dispatched into a functional unit which is specified by the functional unit field. The executed bit is set when the execution finishes. The predicted bit indicates if the corresponding instruction predicts a data value. The destination field is used to keep the predicted value. Namely, the predicted bit indicates if the content in the destination field is the predicted value. When at least one of the executed and predicted bits is set, the following instructions which are dependent upon the instruction can obtain a source operand. The reissued bit indicates if the corresponding instruction is reissued. Lastly, the program counter field is used for correction of mispredictions and for precise interrupts. It is found that the reissue mechanism can be practically implemented without big hardware cost [10].

2.2 Workloads

The SPEC95 CINT 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 use the object files provided by University of Wisconsin Madison [2], except `132.ijpeg` which is 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

3 Simulation Results

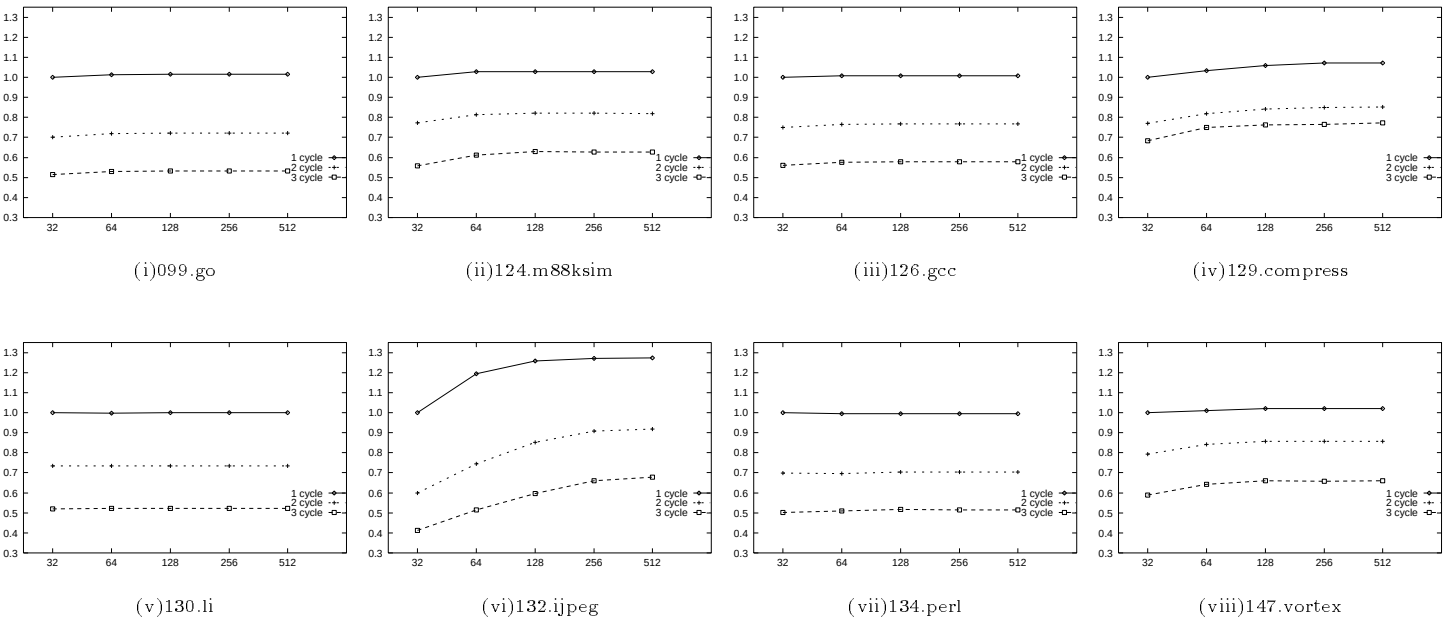
In this section, we present simulation results. We use committed instructions per cycle (IPC) as a metric for evaluating processor performance. First, we evaluate the effect of pipelining the wakeup and select logic of the RUU. And second, the impact of the data speculation on processor performance is shown.

3.1 Pipelined instruction window

In order to evaluate the effect of pipelining wakeup and select logic on processor performance, we simulate three RUUs. The first is the ordinary one whose wakeup and select logic is not pipelined. The second is the one whose wakeup and select logic is pipelined and its latency is two cycles. And the last is the pipelined RUU whose latency is three cycles.

Figure 3 shows simulation results. For each RUU, its size is varied between 32 and 512. The horizontal and vertical axes denote the RUU size and the processor performance respectively. Hereafter, we present the simulation result of each configuration by normalizing it by that of the 32-entry non-pipelined RUU for every program.

First, it can be easily observed that the pipelining of the wakeup and select logic diminishes processor performance severely. When the logic is pipelined by two cycles, the performance is decreased by approximately 30%. The decrease for the 3-cycle pipelined RUU is severer and is approximately 50%. This is because the throughput of the RUU is reduced in spite that the pipelining technique usually increases the throughput when it is utilized in functional units. This is explained as follows. Table 2 presents the utilization of the RUUs. The columns between 2 and 5 are for the non-pipelined RUU. The columns between 6 and 9 are for the 2-cycle pipelined RUU. And the remaining columns are for the 3-cycle pipelined RUU. For each group of four columns, the first two columns indicate the numbers of instructions remaining in the RUU. They include instructions dispatched to functional units. Note that the RUUs force each instruction to retain until the instruction has been committed. The remaining two columns



(i)099.go (ii)124.m88ksim (iii)126.gcc (iv)129.compress

(v)130.li (vi)132.ijpeg (vii)134.perl (viii)147.vortex

Figure 3: Effect of pipelining

show the numbers of instructions waiting for dispatch. Furthermore, for each group of two columns, the left column presents the average number and the right one presents the maximum number. From Table 2 it is found that the average number of instructions waiting for dispatch increases as the latency of the RUU increases. This means that every instruction remains longer in the pipelined RUUs than in the non-pipelined RUU and thus that the throughput of the RUU is reduced when it is pipelined. This explanation becomes clear if we look at the case of `129.compress` where performance difference between the RUUs is relatively smaller than the other programs. From Table 2 we can find that the average number changes little when the RUU is pipelined. Because the throughput is not reduced, the performance degradation in the case of `129.compress` is smaller than in the other programs.

Second, it is observed that the performance improvement except for `132.ijpeg` is modest when the RUU size increases. When the size is increased from 32 to 512, the improvement rates are up to 7.2%, 10.6%, and 13.1% for the non-pipelined, 2-cycle latency, and 3-cycle latency RUU models, respectively. This is due to the limit of instruction supply mechanism. Table 3 shows branch misprediction rate for each configuration. It is found that the misprediction rate is considerably large and is between 3% and 29%. Referring back to Table 2, it is observed that the average number of waiting instructions is not proportional to the increase in the RUU size. Since it is impossible to issue a lot of instructions enough to be scheduled efficiently, large RUU size has only small contribution to the processor performance. While the contribution is modest, it is important to note that the effect of the RUU size is larger for the pipelined RUUs than for the non-pipelined RUU. For example, the improvement rates of `132.ijpeg` are 27.2%, 53.2%, and 63.9% for the non-pipelined, 2-cycle latency, and 3-cycle latency RUU models, respectively, when the RUU size is increased from 32 to 512.

3.2 Data speculation

Figure 4 shows the characteristics of the stride value predictor used in this study. Each bar is divided into three parts. The bottom part (black) indicates the percentage of the instruction whose data value is correctly predicted. The middle part (white) indicates the percentage that is mispredicted. And the top part (gray) indicates the percentage that is not predicted. It is observed that 36.8% of dynamic instructions on average are correctly predicted. While this number of correctly predicted instructions is smaller than we expected, we use this stride value predictor since the study of value predictors is beyond the scope of this paper.

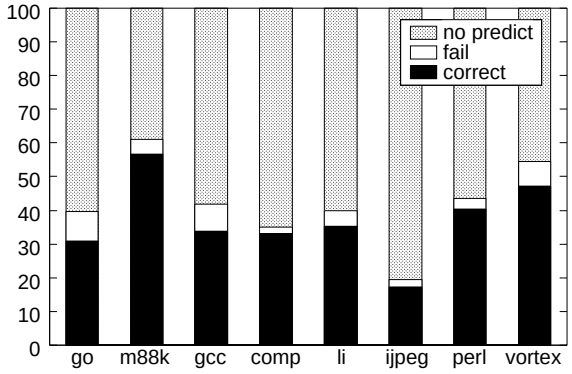


Figure 4: (%)Characteristics of value prediction

Figure 5 presents simulation results when the processor exploits data speculation using the value predictor. For each program, simulation results of the non-pipelined RUU models with (`1cycle+spec`) and without (`1cycle`) data speculation and the 2-cycle pipelined RUU models with (`2cycle+spec`) and without (`2cycle`) data speculation are shown.

First, data speculation contributes to processor performance for all cases. The improvement rate is between 3.4% and 8.6% for the non-pipelined RUU processor model and between 6.2% and 18.1% for the 2-cycle pipelined RUU processor model. As expected from Figure 4, the performance improvement is somewhat small.

Second, the contribution of data speculation on processor performance is larger for the pipelined RUU than for the non-pipelined one. This is because as follows. In the case of the processor with

Table 2: Instruction window utilization

size	1 cycle				2 cycle				3 cycle			
	total		waiting		total		waiting		total		waiting	
	avg	max	avg	max	avg	max	avg	max	avg	max	avg	max
099.go												
32	15.9	32	7.5	31	21.4	32	10.0	31	24.5	32	12.2	32
64	20.3	64	10.2	63	33.5	64	16.7	63	41.6	64	21.8	64
128	21.6	128	11.0	126	43.4	128	23.0	127	61.5	128	34.4	128
256	21.9	256	11.2	252	47.3	256	25.6	255	75.2	256	43.8	256
512	22.0	512	11.3	508	48.5	512	26.4	509	80.0	512	47.1	512
124.m88ksim												
32	16.6	32	7.2	31	23.6	32	10.1	31	26.4	32	12.7	32
64	18.1	64	8.1	63	32.4	64	14.9	63	43.5	64	22.9	64
128	18.3	128	8.3	121	36.8	128	17.0	127	58.2	128	32.4	128
256	18.6	256	8.6	249	37.5	256	17.6	252	61.9	256	35.4	255
512	19.1	512	9.0	505	37.9	512	18.0	503	62.2	512	35.7	509
126.gcc												
32	15.2	32	6.8	31	19.8	32	8.6	31	23.0	32	10.9	32
64	29.0	64	9.4	63	28.2	64	13.3	63	35.7	64	17.7	64
128	21.7	128	11.9	127	34.6	128	17.6	127	47.4	128	25.1	128
256	25.9	256	15.9	255	39.3	256	21.7	255	55.5	256	31.2	256
512	32.6	512	22.4	511	45.7	512	27.8	511	62.5	512	37.6	512
129.compress												
32	28.1	32	14.5	30	29.4	32	14.4	31	29.8	32	12.0	32
64	49.2	64	32.5	57	54.2	64	31.6	59	55.5	64	28.6	63
128	87.6	128	68.4	121	96.5	128	68.3	121	99.2	128	60.2	125
256	161.6	256	139.9	249	173.4	256	141.0	252	172.5	256	122.6	245
512	304.4	512	282.0	505	318.6	512	280.9	505	301.5	512	228.5	501
130.li												
32	16.0	32	6.9	30	21.6	32	9.6	31	24.3	32	12.2	32
64	17.8	64	8.2	57	28.4	64	13.5	62	36.0	64	19.0	64
128	17.9	128	8.2	121	30.3	128	14.8	122	44.3	128	23.8	127
256	17.9	256	8.2	249	30.7	256	15.0	246	47.2	256	25.4	252
512	18.0	512	8.3	505	30.9	512	15.3	500	50.8	512	26.9	501
132.ijpeg												
32	26.9	32	11.0	31	29.3	32	12.3	31	30.2	32	13.5	32
64	45.2	64	19.3	57	53.1	64	22.5	63	56.5	64	25.4	64
128	71.8	128	37.5	121	89.1	128	41.1	122	99.0	128	48.0	128
256	117.3	256	76.1	249	143.5	256	75.8	244	161.1	256	80.1	251
512	191.4	512	146.1	505	231.1	512	138.7	500	260.6	512	150.9	501
134.perl												
32	18.4	32	7.5	30	22.5	32	9.3	31	25.0	32	11.4	32
64	23.0	64	10.4	57	33.7	64	15.2	58	41.2	64	20.1	62
128	23.8	128	10.9	121	42.5	128	20.8	121	57.6	128	29.9	125
256	24.0	256	11.0	249	34.8	256	15.4	244	59.6	256	31.9	253
512	24.4	512	11.2	505	46.7	512	24.0	500	71.1	512	40.0	501
147.vortex												
32	18.6	32	7.9	30	22.6	32	8.4	31	25.2	32	10.4	32
64	23.6	64	11.6	57	33.2	64	13.6	61	39.8	64	17.4	63
128	27.8	128	15.0	121	40.3	128	18.6	123	51.6	128	25.2	126
256	30.8	256	17.9	249	44.1	256	21.9	251	58.0	256	30.4	254
512	34.4	512	21.5	505	47.7	512	25.4	507	63.2	512	35.3	510

the non-pipelined RUU, a data dependence length between an instruction whose outcome is correctly predicted and its dependent instructions is reduced by at most one. On the other hand, in the case of the processor with the 2-cycle pipelined RUU, the length can be reduced by two. In other words, there are more instructions which can be dispatched in the processor with the pipelined RUU than in the processor with the non-pipelined one. Thus, the former processor has more capability for scheduling instructions dynamically, when the data prediction is correct.

Third, there is a vast gap between the `1cycle` and the `2cycle +spec` models. This comparison is important since the data speculation technique favors large instruction window and thus the window may be pipelined in order to maintain the processor cycle time. Table 4 shows the RUU utilization when data dependence speculation is exploited. The layout and subject matter of Table 4 are the same as for Table 2. It is found that when data prediction is used, the average number of waiting instructions decreases except for `129.compress`. This means that the effective capacity of the instruction window is reduced even if it is compared with the RUU which also holds dispatched instructions. Therefore, Figure 5 points out that the data speculation technique can not contribute to processor performance if the RUU should be pipelined in order to implement the instruction reissue mechanism.

4 Decoupled Instruction Window

In order to increase the instruction window size with maintaining the cycle time of data speculative processors, we have proposed the decoupled instruction window [11]. It decouples the recovery mechanism for data speculation from dynamic instruction scheduling structure.

4.1 Mechanism overview

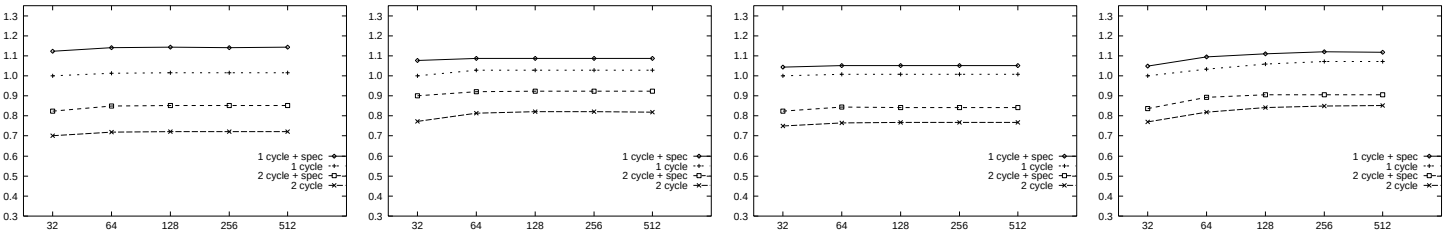
Figure 6 depicts a processor utilizing the decoupled instruction window. The decoupled window consists of a small instruction window for the instruction scheduling and a large instruction buffer for the instruction reissue. After an instruction is fetched and decoded, it enters both the scheduling window and the instruction buffer. When each instruction is dispatched to a functional unit, it leaves the scheduling window and releases its entry, but remains in the instruction buffer. When it is committed, it leaves the instruction buffer and releases its entry. In the case that either the window or the buffer is full, issuing instruction into the decoupled window stalls.

Figure 7 shows an entry of the scheduling window. The window is similar with a reservation station and its entry consists of two source operand fields and a destination field. If a source operand is not ready, the ready bit is reset to indicate that the source operand is not available and a tag for the operand is obtained. When the operand is ready, the content of the source register is held in the source operand field and the ready bit is set. The destination field holds a register number with renaming tag for dynamic scheduling. An entry of the instruction buffer is same as that of the extended RUU depicted in Figure 2.

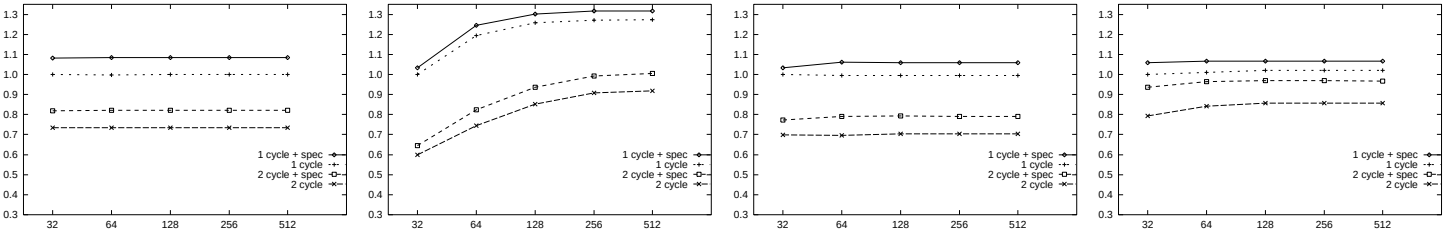
The small instruction window works for dynamic instruction scheduling. Thus, most of the times each instruction is dispatched from the small window. Since instructions are aggressively deallocated from the scheduling window when they are dispatched, the

Table 3: (%)Branch misprediction rate

	099.go			124.m88ksim			126.gcc			129.compress		
size	1 cycle	2 cycle	3 cycle	1 cycle	2 cycle	3 cycle	1 cycle	2 cycle	3 cycle	1 cycle	2 cycle	3 cycle
32	27.92	28.01	28.04	4.41	3.97	3.98	16.74	16.89	16.88	3.17	3.33	3.32
64	28.22	28.44	28.50	4.02	4.58	4.52	17.11	17.63	17.75	3.28	3.58	3.51
128	28.24	28.49	28.59	4.03	4.54	4.53	17.16	17.76	17.96	3.57	3.95	4.08
256	28.23	28.48	28.58	4.03	4.53	4.51	17.15	17.74	17.94	3.58	4.07	4.34
512	28.22	28.48	28.59	4.03	4.55	4.51	17.14	17.77	17.93	3.58	4.10	4.34
	130.li			132.ijpeg			134.perl			147.vortex		
size	1 cycle	2 cycle	3 cycle	1 cycle	2 cycle	3 cycle	1 cycle	2 cycle	3 cycle	1 cycle	2 cycle	3 cycle
32	6.62	6.42	6.26	9.39	9.50	9.48	4.06	3.78	4.05	6.77	6.67	6.60
64	6.70	6.35	7.45	9.87	10.09	10.18	4.12	4.41	4.70	7.27	7.63	7.78
128	6.70	6.44	7.63	9.94	10.16	10.31	4.25	4.64	4.73	7.27	7.80	7.84
256	6.69	6.42	7.61	9.94	10.19	10.27	4.27	4.56	4.79	7.29	7.92	8.09
512	6.69	6.41	7.62	9.94	10.21	10.30	4.24	4.58	4.78	7.27	7.92	8.05



(i)099.go (ii)124.m88ksim (iii)126.gcc (iv)129.compress



(v)130.li (vi)132.ijpeg (vii)134.perl (viii)147.vortex

Figure 5: Effect of data prediction

problem reducing its effective capacity is solved. In addition, its small size does not have serious impact on processor cycle time. However, it is impossible to reissue misspeculated instructions in the scheduling window. The large instruction window works as the backup for the small window and performs the instruction reissue. Each instruction remains in the buffer until it is committed. When a misspeculation occurs, reissued instructions are obtained from the instruction buffer. In order to aggressively speculate instructions, the instruction buffer should be very large. Since it is difficult to access such a large buffer in one cycle, the wakeup and select logic of the buffer is pipelined to maintain high processor cycle time. It is expected that the pipelining does not degrade processor performance, since the instruction buffer is active only when misspeculations are detected.

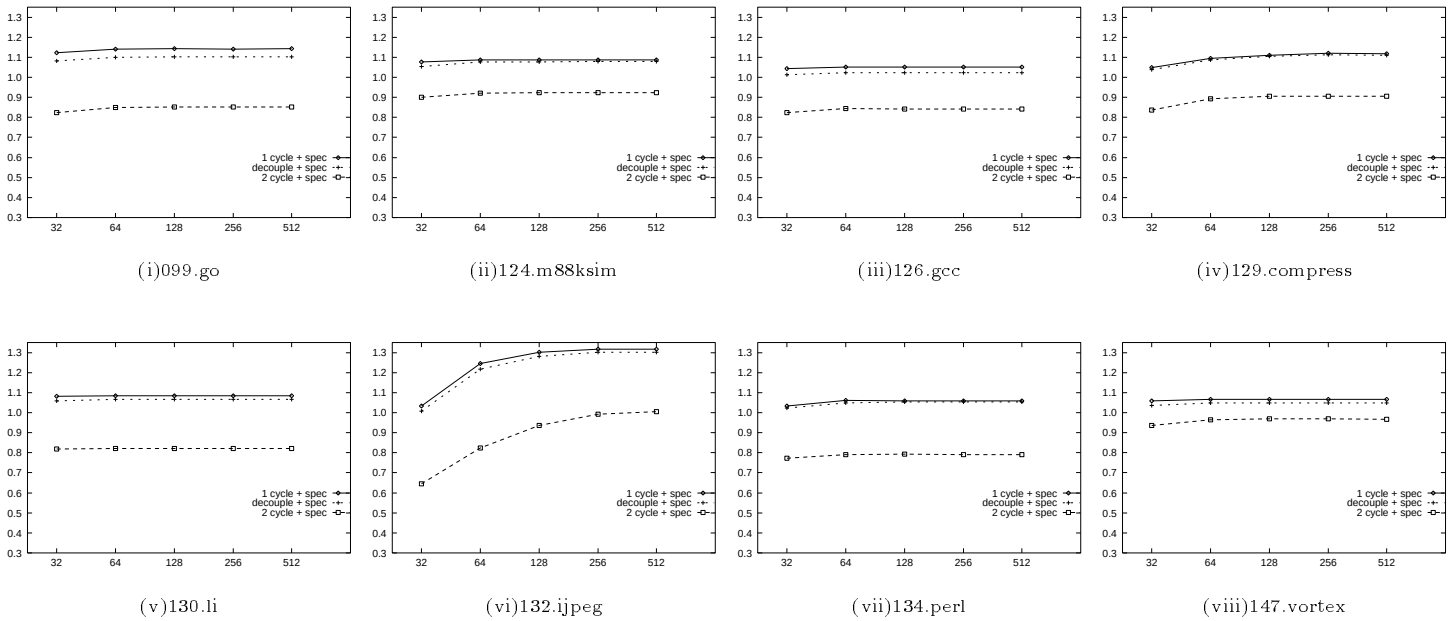
It is straightforwardly decided which structure dispatches an instruction to a functional unit. When an instruction is misspeculated, its dependent instructions which have already dispatched and thus which should be reissued are obtained from the instruction buffer only. They have already left the scheduling window. The dependent instructions which have not been dispatched remain in both the scheduling window and the buffer. However, the instructions are obtained only from the scheduling window, because the dispatched bits for the instructions are reset and thus they are not candidates for reissue.

One drawback of this scheme is that both the scheduling window and the instruction buffer will have the value of the inputs of the instructions. These replications will be eliminated if the decoupled instruction window is based on the MIPS R10000-style instruction window [15].

4.2 Evaluation

In this section, we show simulation results. First, we present performance impact of the decoupled instruction window. Second, we evaluate the effect of the scheduling window size. And last, the effect of the instruction buffer size is shown.

Figure 8 presents how the decoupling of the wakeup and select logic of the RUU affects processor performance. We vary the RUU size (i.e. the instruction buffer size) between 32 and 512. Since the average number of instructions waiting in the RUU is less than half of the RUU size as we have seen in Table 2, we decide that the size of the scheduling window is half of the instruction buffer. For each program, simulation results of the non-pipelined RUU (`1cycle+spec`), the 2-cycle pipelined RUU (`2cycle+spec`), and the decoupled instruction window (`decouple+spec`) processor models are shown. As can be easily observed, the performance of the decoupled instruction window processor model is comparable to that of the non-pipelined RUU processor model. Comparing Figure 8 with Figure 5, it is found that the data speculation becomes useful for processor performance with the help of the decoupled instruction window. From these evaluations, the decoupled instruction window is useful for processors utilizing the data spec-



(i)099.go (ii)124.m88ksim (iii)126.gcc (iv)129.compress

(v)130.li (vi)132.ijpeg (vii)134.perl (viii)147.vortex

Figure 8: Effect of decoupled instruction window

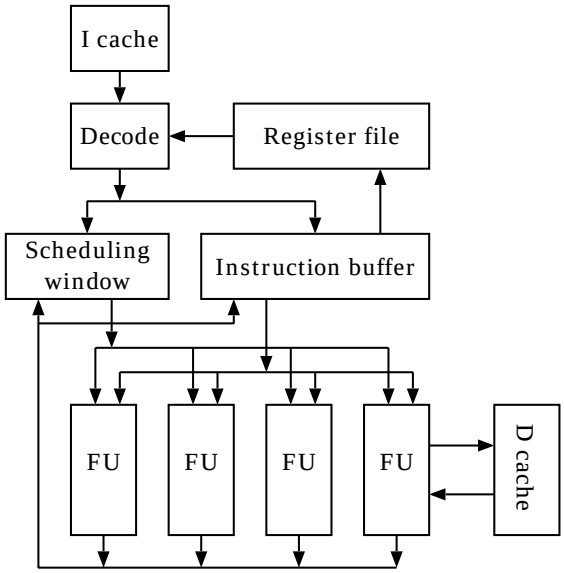


Figure 6: Processor diagram

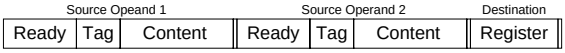


Figure 7: Instruction scheduling window

ulation technique.

Figure 9 shows the effect of the scheduling window size when the instruction buffer size is fixed. In the simulations, the buffer size is 512 and the window size is varied between 32 and 256. It is easily observed that the scheduling window size has little impact on the processor performance except for `132.ijpeg`. This is due to the limit of instruction supply mechanism as explained in Section 3.1. Thus, it is expected that the case of `132.ijpeg` would show the relationship between the scheduling window size and the processor performance if the ideal instruction supply mechanism were used.

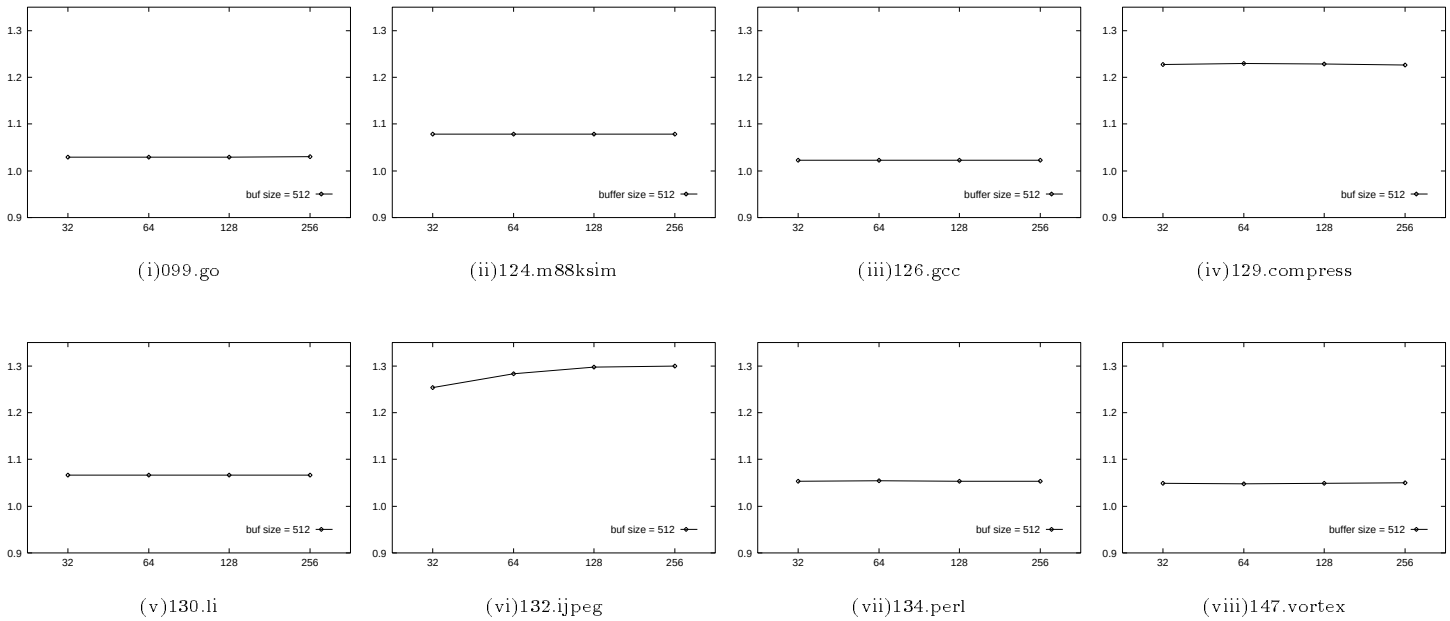
Figure 10 presents the effect of the instruction buffer size when the instruction scheduling window size is fixed. In the simulations, the window size is 32 and the buffer size is varied between 64 and 512. Due to the same reason of the limit of instruction supply mechanism, the impact of the instruction buffer size on processor performance is modest. However, the cases of `129.compress` and `132.ijpeg` show that the processor performance is improved as the buffer size increases.

5 Related Work

Data speculation [4,6] is a technique which executes instructions speculatively using predicted data values. Data dependences are speculatively resolved and thus ILP is increased.

The study of the relationship between the effect of data speculation and the instruction window size was first evaluated by Gabbay [4] and extended by Gonzalez et al. [5]. However, they assumed that processor resource except the instruction window are infinite and also assumed that instruction supply is ideal. In this paper, we restricted the processor resource and the instruction supply mechanism so that the processor model was practical.

When an incorrect data speculation occurs, it is necessary to recover processor state. There are two mechanisms for the recovery action. One is instruction squashing and the other is the instruction reissue. The instruction squashing flushes all instructions following a mispredicted instruction. It is easy to implement the instruction squashing since the same mechanism is already implemented for branch prediction. However, it is found that data speculation relying upon the instruction squashing sometimes diminishes processor performance due to large misprediction penalties [10,14]. On the other hand, the instruction reissue re-executes only instructions dependent upon a misspeculated instruction. Those instructions are detected selectively and reissued in instruction window. Lipasti et al. [6] proposed the concept of the instruction reissue but did not mention its implementation.



(i)099.go (ii)124.m88ksim (iii)126.gcc (iv)129.compress

(v)130.li (vi)132.ijpeg (vii)134.perl (viii)147.vortex

Figure 9: Effect of scheduling window size

Tyson et al. [14] evaluated the usefulness of the instruction reissue. They proposed a renaming-based load value predictor and found that the instruction reissue proposed in [6] can improve processor performance even for the applications whose performance is degraded when the instruction squashing is used. However, the practical implementation of the instruction reissue was not discussed. Rotenberg et al. [9] investigated an instruction reissue scheme on Trace Processor architecture and found it is useful for dataflow speculation. However, they did not evaluate it on superscalar processors which are currently in mainstream.

We proposed a practical implementation of the instruction reissue [10]. The RUU is extended to realize the instruction reissue. It is very suitable for the instruction reissue, since it forces each instruction to retain until the instruction has been committed. Our instruction reissue mechanism serially detects and reissues all instructions dependent upon a misspeculated instruction, and thus a lot of comparators working in parallel to detect the dependent instructions can be removed.

Akkary et al. [1] proposed two level instruction window structure. There are two instruction windows. One is small and its entry is released immediately when a instruction allocated to the entry is dispatched. The other is large and works as the backup of the small one when a misspeculation occurs. The backup process works just like cache refill process. Dependent instructions which should be reissued are injected to the small instruction window from the large one. Since most of the time the small instruction window is utilized, the negative impact caused by the large window on processor performance is reduced. They evaluated the two level instruction window only on a multithreading processor.

Recently, we proposed the decoupled instruction window and only preliminary evaluation results were presented [11]. In this paper, we have evaluated it extensively.

6 Concluding Remarks

In this paper, we have evaluated the effect of pipelining the wakeup and select logic of the instruction window and that of decoupling the instruction reissue mechanism from the instruction scheduling structure. From the cycle-by-cycle simulations, we have found the followings.

- The pipelining of the instruction window has serious impact on processor performance. The performance degradation rates for the 2-cycle and 3-cycle pipelined RUUs are approximately 30% and 50% respectively.
- If the RUU should be pipelined in order to implement the instruction reissue mechanism, the data speculation technique cannot contribute to processor performance.
- The decoupled instruction window is useful to pipeline the instruction reissue mechanism without severe impact on processor performance.

Currently, we are planing to evaluate the pipelining and the decoupling when the instruction supply mechanism is ideal. We expect that we will have interesting simulation results, since the instruction supply mechanism is one of the bottleneck of modern microprocessors and the bottleneck is removed.

Acknowledgment

The author is grateful to Dr. Mitsuo Saito, Dr. Haruyuki Tago and Dr. Shigeru Tanaka for their continuous encouragements. He also thanks the anonymous reviewers whose comments and suggestions helped to improve the quality of this paper.

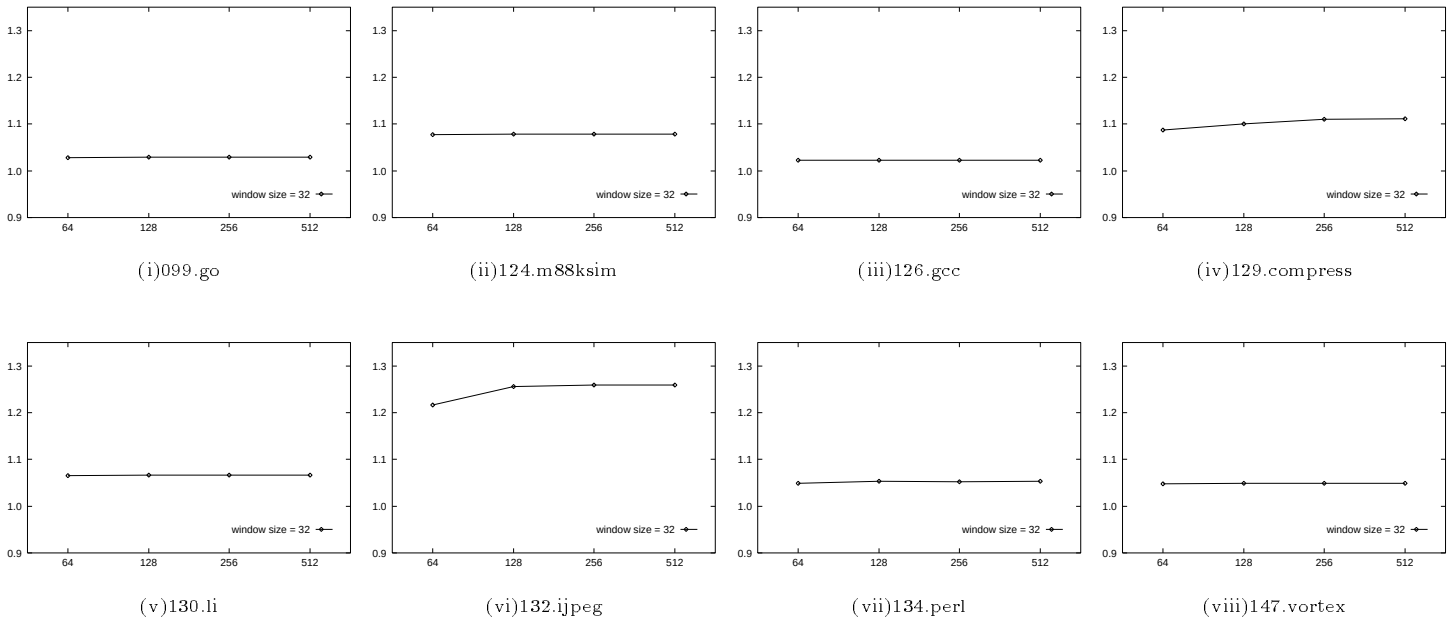
References

[1] H.Akkary, M.A.Driscoll "A dynamic multithreading processor", 31st Int'l Symp. on Microarchitecture, 1998.

[2] D.Burger, T.M.Austin, "The SimpleScalar tool set, version 2.0", ACM SIGARCH Computer Architecture News, 25(3), 1997.

[3] G.Z.Chrysos, J.S.Emer, "Memory dependence prediction using store sets", 25th Int'l Symp. on Computer Architecture, 1998.

[4] F.Gabbay, "Speculative execution based on value prediction", Technical Report #1080, Department of Electrical Engineering, Technion, 1996.



(i)099.go (ii)124.m88ksim (iii)126.gcc (iv)129.compress

(v)130.li (vi)132.ijpeg (vii)134.perl (viii)147.vortex

Figure 10: Effect of instruction buffer size

[5] J.Gonzalez, A.Gonzalez, "The potential of data value speculation to boost ILP", 12th Int'l Conf. on Supercomputing, 1998.

[6] M.H.Lipasti, C.B.Wilkerson, J.P.Shen, "Value locality and load value prediction", Int'l Conf. on Architectural Support for Programming Languages and Operation Systems VII, 1996.

[7] P.Marcuello, A.Gonzalez, "Speculative multithreaded processors", 12th Int'l Conf. on Supercomputing, 1998.

[8] S.Palacharla, N.P.Jouppi, J.E.Smith, "Complexity effective superscalar processors", 24th Int'l Symp. on Computer Architecture, 1997.

[9] E.Rotenberg, Q.Jacobson, Y.Sazeidas, J.Smith, "Trace processors", 30th Int'l Symp. on Microarchitecture, 1997.

[10] T.Sato, "Data dependence speculation using data address prediction and its enhancement with instruction reissue", Euromicro'98 Conf., Workshop on Digital System Design: Architectures, Methods and Tools, 1998.

[11] T.Sato, "Decoupling instruction reissue and scheduling mechanisms", IPS Japan SIG Notes 99-ARC-133-4, 1999 (in Japanese).

[12] G.S.Sohi, "Instruction issue logic for high-performance, interruptible, multiple functional unit, pipelined computers", IEEE Trans. Comput., 39(3) ,1990.

[13] G.S.Sohi, S.E.Breach, T.N.Vijaykumar, "Multiscalar processors", 22nd Int'l Symp. on Computer Architecture, 1995.

[14] G.Tyson, T.M.Austin, "Improving the accuracy and performance of memory communication through renaming", 30th Int'l Symp. on Microarchitecture, 1997.

[15] K.C.Yeager, "The MIPS R10000 superscalar microprocessor", IEEE Micro, April 1996.

Table 4: Instruction window utilization (with data speculation)

size	1 cycle				2 cycle			
	total		waiting		total		waiting	
	avg	max	avg	max	avg	max	avg	max
099.go								
32	15.8	32	6.4	31	21.1	32	8.5	31
64	20.1	64	8.7	63	32.6	64	14.4	63
128	21.5	128	9.6	126	42.0	128	20.2	127
256	21.8	256	9.8	252	46.1	256	22.8	255
512	22.2	512	10.0	508	47.4	512	23.6	509
124.m88ksim								
32	15.1	32	5.0	31	20.8	32	6.3	31
64	16.0	64	5.5	63	25.5	64	8.2	63
128	16.2	128	5.7	127	26.7	128	8.9	127
256	16.5	256	5.9	249	27.0	256	9.1	252
512	17.0	512	6.4	505	27.5	512	9.6	504
126.gcc								
32	14.9	32	5.9	31	19.3	32	7.1	31
64	18.6	64	8.3	63	27.2	64	11.2	63
128	21.6	128	11.1	127	33.0	128	15.3	127
256	26.1	256	15.4	255	37.9	256	19.7	255
512	33.2	512	22.5	511	44.5	512	26.1	511
129.compress								
32	28.0	32	14.9	25	29.3	32	14.0	29
64	49.4	64	34.1	57	54.2	64	34.6	58
128	88.5	128	72.0	121	96.6	128	73.7	121
256	164.8	256	146.7	249	175.1	256	150.0	249
512	313.4	512	292.6	505	326.8	512	300.1	505
130.li								
32	16.0	32	5.0	30	21.2	32	7.8	31
64	18.6	64	6.8	57	26.8	64	10.7	61
128	17.7	128	6.9	121	28.7	128	11.8	118
256	17.7	256	6.9	249	28.8	256	11.8	246
512	11.7	512	6.9	505	28.8	512	11.9	500
132.ijpeg								
32	26.5	32	9.8	31	29.1	32	10.9	30
64	44.8	64	17.8	57	52.6	64	19.9	63
128	71.8	128	26.5	121	89.5	128	37.5	121
256	118.7	256	75.7	249	147.4	256	73.0	244
512	198.3	512	149.5	505	243.9	512	129.0	500
134.perl								
32	17.7	32	6.6	29	21.7	32	7.7	31
64	21.6	64	8.9	57	31.8	64	12.5	58
128	22.0	128	9.1	121	38.4	128	16.4	122
256	22.2	256	9.2	249	40.6	256	18.0	250
512	22.5	512	9.4	505	42.1	512	18.8	500
147.vortex								
32	17.4	32	6.7	29	21.1	32	6.5	30
64	21.8	64	10.0	57	28.5	64	10.7	62
128	24.8	128	12.8	121	33.0	128	14.5	123
256	27.5	256	15.4	249	36.4	256	17.7	251
512	31.4	512	19.3	505	40.1	512	21.5	507