Performance Improvement with Circuit-Level Speculation

Tong Liu and Shih-Lien Lu
Intel Corporation
tl999@yahoo.com; shih-lien.l.lu@intel.com

Abstract

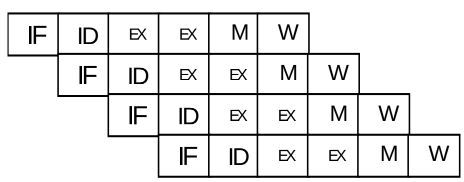
Current superscalar microprocessors' performance depends on its frequency and the number of useful instructions that can be processed per cycle (IPC). In this paper we propose a method called approximation to reduce the logic delay of a pipe-stage. The basic idea of approximation is to implement the logic function partially instead of fully. Most of the time the partial implementation gives the correct result as if the function is implemented fully but with fewer gates delay allowing a higher pipeline frequency. We apply this method on three logic blocks. Simulation results show that this method provides some performance improvement for a wide-issue superscalar if these stages are finely pipelined.

1. Introduction

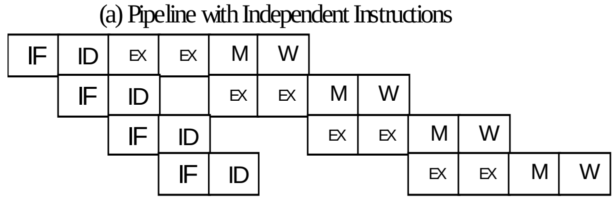
The performance of microprocessor has been accelerating rapidly in recent years. This gain has been achieved through two fronts. On one front, microarchitecture innovations have been able to take advantage of the increase number of devices to process more useful instructions per cycle (IPC). Superscalar is the predominant scheme used. A superscalar processor issues multiple instructions and execution them with multiple identical function unit. It employs dynamic scheduling techniques and executes instructions out of the original program order. The main goal is to exploit as much instruction level parallelism as possible in the program. On the other front, the miniaturization of devices improves layout density and makes the circuits run faster since electrons and holes need only to travel shorter distance. Clever circuit techniques have also been invented to further speed up the logic. Together with finer pipestages, modern microprocessor has accelerated its frequency greatly in recent years.

However, it is believed more complexity is necessary to continue the exploitation of parallelism. This complexity increase tends to cause more circuit delay in the critical path of the pipeline, thus limiting the frequency to go up further. The current approach is to allow logic structures with long delays to spread over multiple pipe-stages resulting in structures that complete the computation in single pipe-stage previously to take more than one cycle. However, finer pipelined machine leads to longer pipeline latency and imposes higher penalty due to branch miss-prediction and miss-speculation. Moreover, other instructions that depend on the results of these multi-staged functional blocks will have to wait until they finish in order to move forward in the pipeline. Figure 1 illustrates the effect of executing consecutive dependent instructions. Therefore, these long delay structures may become the performance bottleneck of microprocessor as clock frequency continues to rise in the future. Thus, one of the essential challenges in achieving performance in future microprocessors is the ability to increase IPC without compromising the ever-increasing clock frequency.

Much work had been devoted to finding methods to increase IPC. One possible approach is to increase the width of the superscalar processor [1-6]. Another approach considered by many researchers is multi-threading [7-12]. Both methods tend to increase the size of the structures used internally such as instruction window and re-order-buffer. Larger size means longer delay and may affect the growth in clock frequency. Work done by Cotofana and Vassiliadis [13] identified the delay complexity of issue logic in a superscalar processor to be a function of issue



(a) Pipeline with Independent Instructions



(b) Pipeline with Dependent Instructions

Figure 1. Example of Dependent and Independent Instructions Pipeline Execution

width. Work by Palacharla et. al. [14, 15] concluded that possible clock limiting structures in a superscalar processor include, register rename logic and issue logic. Also as the machine data and address width increases (currently moving from 32 to 64 bits), we believe adder may also become a bottleneck limiting the increase in frequency because many groups reporting the design of high performance microprocessors include their add circuits in their papers [16-18]. This suggests that adder may limit the frequency of microprocessor if we want to have finer pipeline stages in the future.

In this paper, we propose to use circuit level “prediction” to “speculate” the output of critical logic blocks. The approach calls for a simpler and faster circuit implementation to approximate the original complex function. We termed this technique *approximation*. Approximation circuit should be designed so that it produces the correct result most of the time. Since it is not 100% correct it does require a way to verify the correctness of the approximation. A duplicated logic block, which implements the true function and samples the output at the original worst case delay is used for verification. Results from the approximation and verification blocks are compared to determine if the approximated result used to advance the pipeline is correct or not. When the comparison result is negative we kill the instruction and use the correct result to continue. The recovery mechanism is similar to what is reported in [19]. A modified SimpleScalar [20] tool set in section is used to compare performance.

2. Background and Baseline Design

The logic structures we have considered are adder, issue logic and register rename logic. Adder circuit delay is not related to issue width. However address calculation done by integer adder is the key operation for instruction fetch, branch prediction and data supply from memory [21]. Moreover, we are observing a trend in the growth of datapath width. Currently we are in transition from 32 bits to 64 bits. Designing very fast large adder has been a constant research topic [22, 23]. The latter two are key structures used to exploit ILP in a wide-issue superscalar microprocessor and generally considered as single cycle function logic that are proved to be difficult to pipeline inside. We called these structures cycle limiter. In order to compare performance improvement, a baseline microarchitecture is needed. There are different ways to implement an out-of-order superscalar. Our baseline uses a centralized issue window, which combines the reorder buffer and instruction window together, and can provide precise interrupt [1, 14, 24].

2.1. Adder

Many instructions contain add. Load, store and branch use adder for address calculation. Arithmetic instructions use adder for add, subtract, multiply and divide. There are many different kinds of adders. Due to performance requirement, most of the current high performance processors employ one of the known parallel adders [25]. These parallel adders, such as Carry Look Ahead (CLA), Brent-Kung Adder (BKA), Kogge-Stone Adder (KSA) and Carry Select Adder (CSA), all have comparable asymptotic performance [26] proportional to log (N), where N is the number of bits of the adder. The cost complexity of parallel adders approaches N^2 when fan-in and fan-out of gates used are fixed.

2.2. Register Rename Logic

Register renaming eliminates storage conflicts (anti- and output dependencies). When an instruction is decoded, its destination register is assigned to a physical register (renamed). Usually the number of physical registers is greater than the number of architectural or logical registers. When a later instruction refers to a previously renamed destination register (with its logical binding), it must be able to traverse the renaming and obtains the value stored inside the corresponding physical register or just the tag of the physical register if the value has not yet been produced. Thus, the register rename logic is used to translate logical register designators into physical register designators and is accomplished by accessing a mapping table with the logical register designator as the index. From [14, 15], there are two different implementations: RAM and CAM. In the RAM scheme, the number of entries (i.e., rows) in the mapping table is equal to the number of logical registers and is independent of the number of physical registers. However the mapping table’s entry length (i.e., columns) of the RAM scheme depends on the number of checkpoints needs to be stored. As we issue more instructions per cycle we need to predict over nested branches that will increase the width of the mapping table. The CAM scheme, on the other hand, has fixed table width but requires a larger number of entries. We use the CAM structure in our baseline machine. A block diagram of the renaming logic is shown in Figure 2 (in this figure the horizon entries are rows). It consists of a set of physical registers, a mapping table and a priority encoding logic block. The number of entries in the mapping table is equal to the number of physical registers. When a decoded instruction enters into the rename logic, its destination register is assigned a new entry in the physical register and the corresponding physical register is stored with the logical register binding. The same decoded instruction’s source registers binding will be used to lookup the mapping table associatively. Since it is possible that a logical register can match multiple physical registers due to earlier instructions specify the same destination registers, the result from this associative lookup is channeled into the priority encoding logic. The priority encoder converts the multiple ones into a single active line to be used to access the physical register. The critical path

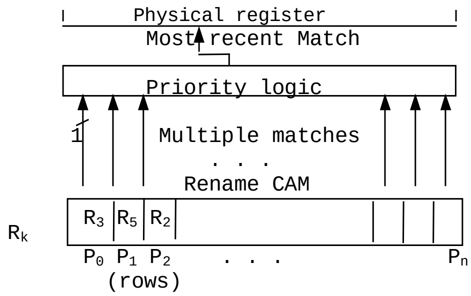


Figure 2. Rename CAM and priority logic, R is logical register, and P is physical register

of register rename using this scheme is the time for mapping table lookup and the priority encoding logic when multiple matches are found. In the worst case, when the matched entry is at the head of the mapping table, N-bit adder-like ripple structure will be formed through the entire priority encoder. A carry look ahead structure (parallel-prefix) can be used and the delay will be in the order of log (N), where N is number of physical registers.

2.3. Instruction Issue Logic

The issue logic contains three different parts, and all of them are speed critical [13, 14, 15]. When an instruction is finished from the functional unit, it writes result back to its destination register. Status of its dependent instructions will be updated by broadcasting the tag associated with the result register to all the instructions in the issue window. If there is a match that particular operand is marked ready. If all operands of an instruction are marked ready, it is ready to be issued. If multiple dependent instructions are ready to issue, there may be contentions on issue bandwidth and functional unit. A selection logic is needed to arbitrate which ready instruction to be issued first. There are different kinds of selection policy, and oldest-first policy, which grant instruction occurs earliest in program order first, is one of most popular policies. In a superscalar machine, since out-of-order issued instructions usually retire in-order, this policy is necessary because issuing old instruction first can resolve dependency quicker and committing earlier instruction first can leave space in the instruction window for newly decoded instructions. When a ready instruction is granted to issue, writeback data of the instruction it depends on will be bypassed from output of the corresponding functional unit to the source register. The delay of wakeup-selection-bypass logic increases with increasing issue window size. The selection logic will start to check the request of instructions from earliest to latest in program order, which is the order of RUU [28] from head to tail. In the worst case, when the only request is from tail of RUU, an adder like ripple carry will be formed through all entries of RUU. A carry look ahead structure can be used to make this process parallel and the delay is the order of log (N), where N is the window size. For wakeup and bypass logic, the RC delay dominates the circuit speed. Circuit simulation shows that RC delay is more sensitive to window size than logic gate level [14]. For the multiple issue case, the delay analysis will be similar.

3. Circuit Level Speculation

Previous study [29] shows that for random input data, the average carry length of a CLA is only 1/3 of its data length. Moreover, other works have shown that there is redundancy exits in programs [30-32], i.e., many instructions perform the same computation with similar input data pattern repeatedly. This could be used for adder output speculation. For example, in address calculation, one of the input to the adder is static. Moreover the other operand is usually incrementing with a regular stride. Therefore the actual adder delay is much shorter than the worst case maximum delay. We use the approximation technique described in the introduction section by generating a part of the whole carry chain. As for the register renaming logic, we believe that the renaming will mostly happen among instructions close to each other, so we employ the approximation method described previously and use a simpler priority encoding logic. For issue logic, we only select among a small group of instructions close to the head of instruction queue to issue. Due to this selection strategy, the wakeup and bypass logic can be prioritized to work on the corresponding instructions closer to head of instruction queue first, and work on the rest of instructions later. Because of the approximation techniques, the total pipestages of machine are shorter, the dependency chain will be resolved faster, and results in higher IPC. As other prediction methods, circuit level value prediction is not 100% accurate. If the prediction is wrong, the false speculated instruction has to be re-issued and re-executed. This will cause more resource contention, and dependency chain will be resolved even slower than the baseline structure. If the prediction accuracy goes down to a certain point, the speculatively architecture will perform worse than the baseline architecture. So we can only work on the logic structure whose behavior is highly predictable. If the prediction accuracy is high enough to overcome the replay penalty of false speculation, the performance improvement is expected. Also the wrongly speculated instruction output will trigger its dependent instructions to start execution and produce more false result. These false results will trigger their own dependent instructions to execute, and cause a chain reaction resulting in large overhead and overall performance loss. Therefore it is important to stop the write-back of the speculative instructions as soon as the false prediction is detected. We describe the details of our design and analysis used in the following sections.

3.1. Adder

The critical path of an adder is its full carry chain. For an N-bit adder, we denote the individual bits of the two input operands as a_i, b_i and intermediate carries as c_i (i=0, …, N-1). Each intermediate carry signal - c_i depends on all its previous input bits. i.e.,

$$c_i = f(a_{i-1}, b_{i-1}, a_{i-2}, b_{i-2}, \ldots, a_0, b_0)$$

Thus, in order to generate the correct final result, we must consider all input bits (look ahead all inputs) to obtain the final carry out. However in real programs, inputs to the adder are not completely random and the effective carry chain is much shorter for most cases. Our approximated design considers only the previous k inputs (lookahead k-bits) instead of all previous input bits for the current carry bit. i.e.,

$$c_i = f(a_{i-1}, b_{i-1}, a_{i-2}, b_{i-2}, \ldots, a_{i-k}, b_{i-k}) \text{ where } 0 < k < i+1 \text{ and } a_j, b_j = 0 \text{ if } j<0$$

If we choose $k = \sqrt{N}$, our new approximation adder only need half of the original delay ($\log\sqrt{N}$ = ½ log N). The prediction rate of an N-bit adder with k bits carry chain is:

$$P(N, k) = \left(1 - \frac{1}{2^{k+2}}\right)^{N-k-1}$$

For example, a 64-bit approximation adder with 8-bit (8 $=\sqrt{64}$) look-ahead gives correct result 95% of the time assuming random input data.

3.2. Rename Logic

As mentioned previously, the critical path of the register rename logic is the delay of the associative lookup and the priority logic when multiple matches are found. By experimenting with benchmarks, we found that dependent instructions may have spatial locality. In other words, they are most likely to be close to each other. Thus, we propose to use a smaller CAM to implement the mapping table. The CAM table basically contains a portion of the whole map. When a new instruction enters the rename logic, its destination binding is assigned a new physical binding. The mapping table is updated if the table is not full. Otherwise the oldest one is dropped to leave room for the newly renamed destination binding. At the same time the source bindings are used to lookup the partial CAM. If there is no physical mapping found in the small CAM but the mapping does exist in the full CAM, A mis-speculation occurs. Since the number of inputs to the priority encoder is equal to the number of entry in the smaller CAM, the delay for the rename logic is also smaller. In order to double the speed, we propose to use a much smaller CAM table containing only the latest $\sqrt{N}$ number of instruction's register mapping table in it, where N is the window size. Because of the locality property of register dependency, we hope to get most of the reading operation from the rename logic correctly. Beside the faster (approximation) renaming logic, we still keep a regular full CAM and the associated full length priority encoder. It will be used to recover the mis-speculation and provide the correct renaming result in the next cycle.

3.3. Issue Logic

We use the same idea as rename logic by targeting the issue logic on the earliest $\sqrt{N}$ entries (N = window size), so that the issue logic only needs to consider waking up, selecting and bypassing data to instructions within $\sqrt{N}$ entries to the head of RUU. Since the wakeup and bypass delay are RC dominated, and RC delay is more sensitive to the window size, we will have more than twice speed up in these two logics. So the total speculative issue logic delay will be less than half of the issue logic in baseline microarchitecture if only $\sqrt{N}$ entries are considered. There is no replayed needed for the approximated issue logic since there is no false result generated.

4. Implementation and Recovery

4.1. Implementation Cost

Our new microarchitecture uses the speculative adder, rename and issue logic as described previously. A pair of duplicated normal adders and rename logic is also included in the machine being sampled at a slower frequency. Since the slower verification logic is running half speed as the speculative logic in the main data path, two identical ones are needed to interleave the input data so as to catch up with the fast frequency. The size of the above mentioned circuit-level speculation logic for rename and issue is smaller than the original logic used in the baseline machine, since the speculative window size is scaled down (in our case the size is the square root of the original size). For an N-bit adder with k-bit carry look-ahead, a total of N k-bit adders are needed. When k is large, the new design may have a significantly larger area. Fortunately, from our benchmark experiment, 4 bits of carry look-ahead can achieve an average of 85% prediction rate for 64 bits adder (random inputs give only 40% accuracy), this is due to the redundancy in program data. Each pieces of small carry chain only has local wire routings, so the device size can be smaller and layout can be rather compact. Thus, in general, our duplicated hardware used to speculate is smaller in size than the original hardware. This is different from DIVA processor proposed by Austin [33], which requires an almost identical hardware as checker. Both approaches speculate on circuit timing and both can avoid metastability.

4.2. Recovery

After the verification logic finished, the result is compared with corresponding "speculative result". If they match, no other action is required. Otherwise instructions, which generate a false result, will be issued again and write back with the correct result from verification logic. We assume that it takes an extra cycle for the slow (original) logic to finish and verify the speculative result. Also, as soon as the false speculation is known, the writeback of the speculative instruction is stopped so that it won't trigger the next dependent instructions. For issue speculation, there won't be any false result generated, so no replay is needed.

The issue mechanism in the superscalar microarchiteture is event triggered. This means an instruction will check the readiness of all of the source registers and decide to send a request to issue only when new data is written to any of the source registers. This can happen in two cases:

I. In rename stage, if all source register data are available, either in physical register it matched with, or direct from architectural register file, then the instruction is ready to issue immediately.
II. In writeback stage, when an instruction finishes execute and writeback data, its dependent instructions will be waked up, instructions with all source data available are ready to issue.

We now discuss the detail on how the newly proposed microarchitecture handles speculation and replay. In our design, RUU has the same content as baseline microarchitecture except every entry has flags showing the bogus speculation, one per each source register. We call it value prediction flag (VPF). Initially all VPFs are reset. The VPF of a register will be set when the verification logic finds out that the speculation done on the corresponding instruction before is wrong, or that register is written back by an earlier instruction whose VPF has been set. The VPF will be cleared when the corresponding register is written back by an earlier instruction whose VPF is cleared. VPF will gate the writeback of the instructions so that they won't contaminate its dependents. Because it takes one extra pipestage for the verification logic to figure out the result of the speculation, VPF will be updated one cycle later than the speculation stage. If an instruction's writeback stage is immediately following it speculation stage, it will trigger its dependent instruction to issue because VPF hasn't been set yet. However, after the dependent instruction issues, its VPF will be assigned and its writeback will be stopped if false speculation happens. Since updating VPF for the dependent instructions can be done in parallel with their executions, it won't degrade the performance. We didn't use speculative adder for branch instruction. The reason is that branch will be resolved in the next cycle immediately after the adder calculates the address, and before VPF of the branch instruction is assigned. The false speculation of adder will cause spurious branch mispredictions. In other words, a correctly predicted branch will be considered mispredicted because the adder that is used to calculate target address and to verify the branch prediction is wrong. The penalty of recovering from spurious branch mispredictions will be higher than the benefits we get from the value prediction of add. For rename speculation, because it happens at front end of the machine pipeline, the VPF of the false speculated instruction would be set before the branch resolved. So no spurious branch miss-predictions will happen.

5. Simulation Result

Table 1. Common parameters of base simulator

Fetch width	4 inst. per cycle
Instruction cache	16K byte, Direct mapped, 32 byte line, 6 cycle miss latency
Branch Predictor	Bimodel, 2048 BTB entries with 2 bit saturating counter
Issue mech.	Out-of-order issue, commit at 4 operations per cycles, load may execute when all prior store addresses are known
FU	2 load/store, 4 fp adders, 1 fp MUL/DIV
FU latency (total/issue)	load/store 1/1, int ALU 1/1, int MUL 3/1, int DIV 29/19, fp adder 2/1, fp MUL 4/1, fp DIV 12/12, fp SQRT 24/24
Data cache	16K byte, 4 way set associate, 32 byte line, 6 cycle miss latency

Table 2. Parameters of four cases of base simulator

	Issue width	RUU, LSQ	ALU	MUL	Speculation window	carry chain
I4R64	4	64, 64	4	1	8	4
I8R64	8	64, 64	8	2	8	4
I4R32	4	32, 32	4	1	4	4
I4R16	4	16, 16	4	1	4	4

We use SimpleScalar tool set to compare the performance of our speculative microarchitecture with the baseline machine. Assume both models run with the same frequency. In the baseline machine, in order to keep up the frequency the cycle limiter logic blocks all take 2 cycles. While in the new speculative machine with approximation circuits, these same logic blocks take only 1 cycle. However the speculative machine will need to replay when the result is incorrectly generated and incur miss speculation (replay) penalty. Independent simulation experiment is performed for each of the above mentioned cycle limit logic - rename logic, issue logic and adder, with the assumption that only one of them is the main performance limiter. We run eight integer benchmarks from the spec95 suite, using the reference input database. First, we set the RUU window size = 64, issue width = 4, integer adder number = 4, integer multiplier number = 1, and run 2 billion instructions for each benchmark. Then by shuffling the parameters: window size of 16, 32, issue width of 8, integer adder number of 8 and integer multiplier number of 2, we run each benchmark for 500 million instructions. These parameters are listed in Table 1 and 2. The speedup results are summarized in Figure 3-5. The speedup is basically the ratio of IPC with baseline machine normalized to one. Bars labeled HM in all figures are the harmonic mean over all the benchmarks simulated.

From these diagrams, we can see that circuit-level speculation method described does improve the overall performance of the new microarchitecture. Also from simulation result, the average prediction rates for speculative adder, rename logic and issue logic are 88%, 80% and 36% respectively. For adder speculation, the performance improvement is less than the other two speculations. This is because addition completes close to the back end of the machine, it is more likely to pollute the dependent instructions by false writeback and cause more penalties. By reducing window size, the adder speculation performance relative to the baseline machine increased. This reason is smaller number of independent instructions is available in a smaller issue window. So the speculation is more important and efficient to resolve dependencies. On the other hand, increasing issue width and number of function units degrades the relative performance, since wider issue width, larger window size and more functional units potentially cause larger instruction level parallelism, and the mis-speculation penalty will overcome the performance gain by resolving dependency chain. However, for rename and issue speculation, the speculative window size will change to match the baseline window size so that to achieve the circuit speedup of twice fast. This will compromise the relationship between relative performance and window size, issue width and functional unit. For case I8R64, which means wide issue, large window and more functional unit, the relative performance of ijpeg degrades a lot in issue and add speculation. The predication accuracy of issue speculation means the percentage of ready instructions in speculation window over the total ready instructions. It is as low as 24% for ijpeg, causing huge waste of execution bandwidth. Since ijpeg is a computational intensive program, it is full of independent data processing instructions, which means there are fewer dependencies than other benchmarks. This explains the low performance

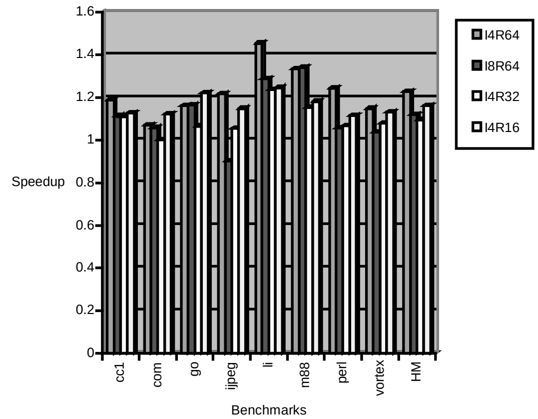


Figure 3. Speedup by speculative issue logic

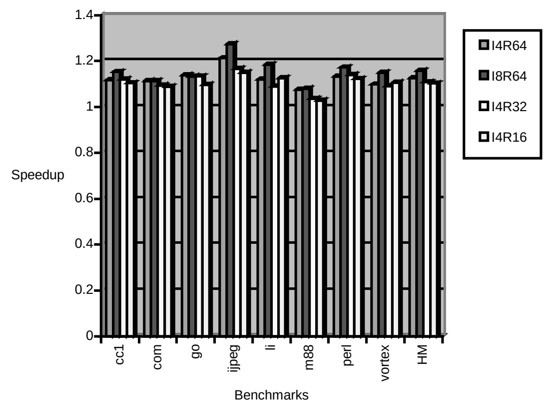


Figure 4. Speedup by speculative rename logic

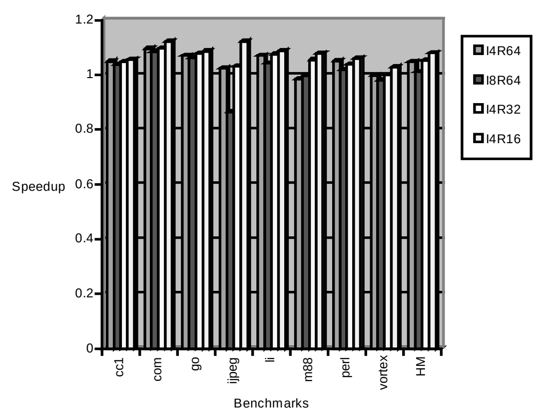


Figure 5. Speedup by approximation adder

gain with issue and adder circuit-level speculation.

6. Conclusion

In this paper, we first try to identify some possible cycle limiters in a superscalar microprocessor, namely adder, rename logic and issue logic and analyze their speed path. Then we propose a circuit level speculation method – approximation to speedup these critical logic blocks. For adder, carry chain is generated by a subset of the input data. For rename and issue logic, we only target on a subset of instructions in the issue window. For adder and rename logic, the corresponding verification logic must be duplicated to detect the correctness of value prediction. In case of false speculation, the instruction will be replayed. Our simulation of SPEC95 benchmarks with different window size, issue width and number of function units shows performance improvement for this newly proposed microarchitecture over the baseline machine. Our conclusion is that circuit level speculation method is a potential way to speedup some cycle limiting logic structures and achieve better performance in wide issue superscalar microprocessor. Approximation method works better on program with more dependencies than that with high ILP originally. The extra hardware cost both for duplicated logic blocks and verification logic is somewhat limited.

References

[1] James E. Smith, and Gurindar S. Sohi, “The Microarchitecture of Superscalar Processors,” in Proc. of the IEEE, Vol.: 83 12, Dec. 1995, pp. 1609 –1624.

[2] P. Michaud, A. Seznec, and S. Jourdan, “Exploring instruction-fetch bandwidth requirement in wide-issue superscalar processors,” in Proc. of the Int. Conf. on Parallel Architectures and Compilation Techniques, 1999, pp. 2 –10.

[3] S. Dutta, and M. Franklin, “Control flow prediction schemes for wide-issue superscalar processors,” IEEE Transactions on Parallel and Distributed Systems, Vol.: 10 4, April 1999, pp. 346 –359.

[4] Sangyeun Cho; Pen-Chung Yew; Gyungho Lee, “Decoupling local variable accesses in a wide-issue superscalar processor,” in Proc. of the 26th Int. Symp. on Comp. Arch., 1999, pp. 100 –110.

[5] J. Farrell and T. C. Fischer, “Issue Logic for a 600-MHz Out-of-Order Execution Microprocessor,” IEEE JSSC, Vol. 33, No. 5, May 1998, pp. 707-712.

[6] S. J Patel, D. H. Friendly and Y. N. Patt, ”Evaluation of design options for the trace cache fetch mechanism,” IEEE Transactions on Computers, Vol.: 48 2, Feb. 1999, pp.193 - 204

[7] D. M. Tullsen, S. J. Eggers, and H. M. Levy, “Simultaneous multithreading: Maximizing on-chip parallelism,” in Proc. of 22nd Ann. Int. Symp. Comp. Arch., 1995, pp. 392 –403.

[8] C. B. Zilles, J. S. Emer and G. S. Sohi, “The use of multithreading for exception handling,” in Proc. of 32nd Ann. Int. Symp. on Microarchitecture, 1999, pp. 219 –229.

[9] P. Marcuello, J. Tubella, and A. Gonzalez, “Value prediction for speculative multithreaded architectures,” in Proc. of 32nd Ann. Int. Symp. on Microarchitecture, 1999, pp. 230 –236.

[10] S. Wallace, D. M. Tullsen and B. Calder, “Instruction recycling on a multiple-path processor,” in Proc. of Fifth Int. Symp. On High-Performance Comp. Arch., 1999, pp. 44 –53.

[11] J. -M. Parcerisa, and A. Gonzalez, “The synergy of multithreading and access/execute decoupling,” in Proc. of Fifth Int. Symp. On High-Performance Comp. Arch., 1999, pp. 59 –63.

[12] H. Akkary, and M. A. Driscoll, “A dynamic multithreading processor,” in Proc. of 31st Ann. Int. Symp. on Microarchitecture, 1998, pp. 226 –236.

[13] S. Cotofana, and S. Vassiliadis, “On the Design Complexity of the Issue Logic of Superscalar Machines,” in Proc. of the 24th Euromicro Conf., 1998, pp. 277 –284.

[14] Subbarao Palacharla, Norman P. Jouppi, J. E. Smith, “Complexity-Effective Superscalar Processors,” in Proc. of the 24th Int. Symp. on Comp. Arch., June 1997.

[15] Subbarao Palacharla, Norman P. Jouppi, J. E. Smith, “Quantifying the Complexity of Superscalar Processors,” Technical Report CS-TR-96-1328, University of Wisconsin-Madison, November 1996.

[16] R. Bechade et. al., “A 32b 66 MHz 1.8 W microprocessor,” in Digest of Technical Papers of the 41st IEEE Int. Solid-State Circuits Conf., 1994, pp. 208 –209.

[17] D. Dobberpuhl et. al., “A 200 MHz 64 b dual-issue CMOS microprocessor,” in Digest of Technical Papers of the 39th IEEE Int. Solid-State Circuits Conf., 1992, pp. 106 -107, 256.

[18] H. Sanchez et. al., “A 200 MHz 2.5 V 4 W superscalar RISC microprocessor,” in Digest of Technical Papers of the 43rd IEEE Int. Solid-State Circuits Conf., 1996, pp. 218 - 219, 448.

[20] M. H. Lipasti, and J. P.Shen, “Exceeding the dataflow limit via value prediction,” in Proc. of the 29th Ann. IEEE/ACM Int. Symp. on Microarchitecture, 1996, pp. 226 –237.

[21] D.C. Burger and T.M. Austin, “The SimpleScalar Tool Set, Version 2.0,” University of Wisconsin Computer Science Technical Report #1342, June 1997.

[22] Y. Shintani et. al., “A Performance and Cost Analysis of Aplying Superscalar method to Mainframe Computers,” IEEE Trans. On Computers, Vol. 44, No. 7, July 1995, pp. 891-902

[23] Wei Hwang; Gristede, G.; Sanda, P.; Wang, S.Y.; Heidel, D.F, “Implementation of a Self-resetting CMOS 64-bit Parallel Adder with Enhanced Testability,” IEEE JSSC, Vol.: 34 8, Aug. 1999, pp. 1108 –1117.

[24] L.A. Lev et. al., “A 64-b microprocessor with multimedia support ,” IEEE JSSC, Vol.: 30 11 , Nov. 1995 , pp. 1227 - 1238.

[25] Mike Johnson, Superscalar Microprocessor Design. Prentice Hall Series in Innovative Technology. 1991.

[26] C. Nagendra, M.J. Irwin, and R.M. Owens, “Area-time-power tradeoffs in parallel adders,” Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on Vol.: 43 10 , Oct. 1996 , pp. 689 –702.

[27] T. Lynch,and E. Swartzlander, “The redundant cell adder,” in Proc..of the 10th IEEE Symp. on Computer Arithmetic, 1991, pp. 165 –170.

[28] G. Sohi, “Instruction Issue Logic for High Performance, Interruptible, Multiple Functional Unit, Pipelined Computers,” IEEE T. on Computers, Vol. 39, No. 3, March 1990, pp.349-359.

[29] R. Ramachandran and S. L. Lu, "Carry Logic," Wiley Encyclopedia of Electrical and Electronics Engineering, Edited by John G. Webster, 1999.

[30] Avinash Sodani and Gurindar S. Sohi, “Dynamic Instruction Reuse,” Proc. of the 24th Int. Symp. on Comp. Arch., June, 1997.

[31] Avinash Sodani and Gurindar S. Sohi, “An Empirical Analysis of Instruction Repetition,” in Proc. of 8th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII), Oct 1998.

[32] Avinash Sodani and Gurindar S. Sohi, “Understanding the Differences between Value Prediction and Instruction Reuse,” in Proc. of 31st Int. Symp. on Microarchitecture, Nov-Dec 1998.

[33] T. M. Austin, “DIVA: a reliable substrate for deep submicron microarchitecture design,” in Proc. of the 32nd Ann. Int. Symp. on Microarchitecture, 1999, pp. 196-207.