Direct Load: Dependence-Linked Dataflow Resolution of Load Address and Cache Coordinate

Byung-Kwon Chung[1], Jinsuo Zhang[2], Jih-Kwon Peir[2], Shih-Chang Lai[3], Konrad Lai[4]

[1]USUN02-203, Sun Microsystems, byung-kwon.chung@sun.com
[2]Department of CISE, University of Florida, {jizhang, peir}@cise.ufl.edu
[3]Department of ECE, Oregon State University, laish@ece.orst.edu
[4]Microprocessor Research Lab, Intel Corp., konrad.lai@intel.com

Abstract

An increasing cache latency in future processors incurs profound performance impacts in spite of advanced out-of-order execution techniques. In this paper, we describe an early address resolution mechanism that accurately resolves both regular and irregular load addresses. The basic idea is to build dynamic dependence links from the instruction that updates the base register to the consumer load instructions. Once a new base address is available, it triggers calculations of the new load addresses for dependent loads. Furthermore, the exact cache location of the requested data is predicted based on the newly resolved load address. As a result, this direct load can access the data cache directly to achieve a zero-cycle load latency. Performance evaluation using SPEC integer programs shows that the dynamic dependence links can be established accurately. Combined with a stride-based predictor, the proposed early address resolution achieves about 97% average accuracy with less than 1% misprediction. Based on a modified SimpleScalar model, the proposed method can potentially improve the IPC by about 18%.

1 Introduction

Memory load latency presents a performance bottleneck in modern processors [19, 23, 22, 15, 13], even when the data is present in the first-level cache. As the increasing in cache size and clock frequency continues in next-generation processors, it is estimated that first-level cache accesses may consume two to five cycles [3]. This increasing cache load latency will further disrupt pipeline execution and will impact performance in spite of advanced out-of-order execution techniques [3, 4].

One way to circumvent cache latency hazards is to predict the load address at the onset of pipeline execution so that a cache access can start speculatively without waiting for the true address [10, 12, 7, 3]. Current address prediction schemes, although shortening the load-to-use latency, suffer from two major deficiencies. First, existing address prediction methods exploit either regular or irregular but repeated address patterns. It is, however, difficult to predict a significant portion (over 30% [3]) of load addresses that do not fall into these two categories. Second, given aggressive multiple-issue microarchitectures, the time required to predict the load address and the lengthy cache access path may still induce stalls to the dependent instructions, even with a correct predicted address. An alternative way is to predict the load value [16]. However, the lack of close correlation between the instruction address and the value of the load makes the prediction accuracy low.

In this paper, we describe a new cache-latency hiding mechanism based on accurate early resolution of load addresses and the exact cache location of the requested data. It is observed that the inaccuracy of predicting a load address often comes from an update of its base register since the last execution instance of the load. When the update does not follow regular strides or some repeated pattern, it is difficult to obtain an accurate prediction. We cope with this problem by building dynamic dependence links from an instruction that produces a new base register value to the consumer load instructions. This newly-updated base address triggers calculations of consumer load addresses in a *dataflow* fashion. Therefore, we call the proposed method a *dependence-linked* address solver.

To further speed up cache accesses, the existing cache-way predictor [17, 6] is used to predict the location of the newly-resolved load address. Such a location, referred to as a *cache coordinate*, is defined by the cache set *index*, the *way* within the set, and the target byte/word within the line. Given the cache coordinate, cache access can be shortened by fetching the

target byte/word out of the data array directly. As a result, this *direct load* method has the potential to achieve a zero-cycle load latency and allow dependent instructions to be fetched/issued in the same cycle.

Performance evaluations based on a modified SimpleScalar [5] using SPEC95 and SPEC2000 integer programs show significant improvements in both address resolution accuracy and Instruction-Per-Cycle (IPC) rate for the proposed method. Combined with a stride-based predictor, the early address resolution can achieve 96.8% average accuracy with merely a 0.76% misprediction rate. This method can potentially improve the IPC by about 18.1% and 9.2% respectively compared with the model without early address resolution or with stride/context hybrid predictors.

The remainder of the paper is organized as follows. The motivation and related work are described in the next section. An architectural design of the proposed dependence-linked address resolution, an integrated cache coordinate predictor, and a few related correctness and performance issues are discussed in Section 3. In Section 4, an example pipeline microarchitecture with link/stride address resolutions is given. This is followed by performance evaluations of both the accuracy and the IPC improvement in Section 5. Finally, Section 6 concludes the paper.

2 Background and Related Work

In order to understand the performance impact of the first-level cache (L_1) access delay, we simulate a set of SPEC2000 integer benchmarks on SimpleScalar pipeline models. We vary (L_1) access latencies from 1 to 5 cycles with various combinations of second-level cache (L_2) and memory latencies. We also try different cache and TLB sizes, reorder buffer (RUU) and load-store queue (LSQ) depth, etc. In summary, each cycle reduction of the L_1 access delay improves the IPC by about 5–10%. Techniques to hide the cache latency are the main focus of this paper.

Load address predictions at the onset of pipeline execution have been considered to reduce the impact of increasing cache latencies. *Stride-based* predictors [10, 7, 12] provide accurate predictions if the operand address generated by a static load increments/decrements with a constant stride. *Context-based* predictors [21, 3] match recent address history or context with the previous address histories to capture irregular but repeated address patterns. A hybrid predictor, which integrates stride-based and context-based predictors, has been considered for load value prediction [16, 24] and load address prediction [3]. The latter study shows that the hybrid scheme can predict 67% of all loads with a very low miss prediction rate. The remaining 33% of loads, however, are still not captured [3].

A dependence-linked prefetching technique has been considered for handling load-load dependencies in special pointer-chasing problems [20, 1]. Similar techniques are also considered for branch resolutions [11]. Recently, it is observed that the addresses for certain types of memory loads, such as stack accesses, have regular increment/decrement patterns [4]. By tracking the registers used for this type of loads, register updates can be computed at the decode stage for early resolving of the dependent load address. There have been several works to achieve fast cache accesses [14, 2, 8, 18]. Their impact in hiding long cache delays on deep-pipelined microarchitecture is rather limited.

Our proposed dependence-linked address resolution has several advantages over existing load latency hiding schemes. First and foremost, the dependence-linked address solver is not limited by any load types, address patterns, or special base register updates. An accurate resolution can be made once the correct dependence link has been built. Second, a base register update can trigger address calculations immediately after the new value is produced. The established update-load dependence links initiate load address calculations in a dataflow fashion. In contrast, in order to capture the latest base address, the correlated context-based predictor is unable to resolve the address as early. Third, the accurate prior resolution of load address and cache coordinate can advance a load several cycles further ahead than other fast cache access techniques.

3 Dependence-Linked Address Solver

In Figure 1, we show how the dependence-linked method can accurately resolve addresses of a load that are difficult to predict. The function *mrglist* taken from *go* in SPEC95 and its corresponding MIPS assembler code are illustrated. At the beginning, *mrglist()* checks whether the two merge lists are empty. Line *(7)* of the assembler code loads the beginning address of the second list using base register *$5* which was set by callers of *mrglist()*. The load value is required immediately by the next two branches. Predicting the load address at line *(7)* by either a stride-based or context-based scheme will fail because the linked list *list2* is a dynamically allocated entity. The dependence-linked address solver, on the other hand, can forward the newly-updated *$5* from the caller for calculating the load address early and accurately.

In order to establish update-load dependence links, two new data structures are created as shown in Figure 2. The *Register Update Table (RUT)* remembers for each register the instruction address (PC) where the register is most recently updated. This is done before

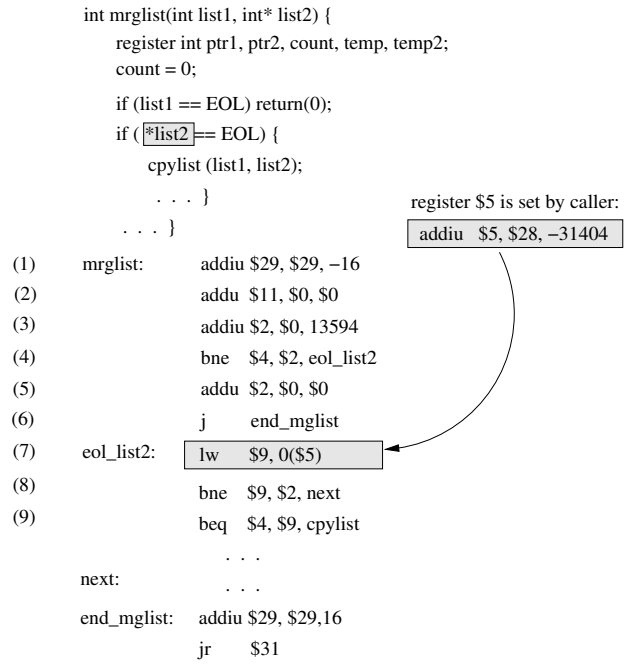

Figure 1: A program segment from *Go* shows an accurate dependence-linked address resolution.

register renaming. Thus, the size of the RUT is same as that of the register file. The *Update Link Table (ULT)* records dependence links that connect the most recently executed instructions, which participate in register updates, to load instructions where the updated registers are used as base/index registers.

The procedure of building dynamic dependence links is also illustrated in Figure 2. When an instruction that involves a register update is decoded, the current instruction address is recorded in the RUT. For instance, PC_i is entered in the RUT's location corresponding to *$5* after *addiu* is decoded. When a load instruction is decoded, the base register dependence is built and recorded in the ULT if such a link does not already exist. The source of the link (PC_i) can be fetched from the RUT using the base register ID (*$5*). The destination of the link is the instruction address (PC_j) of the load. For simplicity, we consider a common case of only a base register used in memory instructions.

Besides the RUT and the ULT, an *Address Resolution Table (ART)* is established to remember the newly-updated base register value for dependent loads. An early load address resolution based on dependence links involves several steps. First, correct dependence links must be built as described previously. Second, whenever an instruction is fetched, the ULT is looked up simultaneously. The dependence links found in the ULT will be used to forward the updated value to dependent loads in the ART. Finally, when the load instruction is fetched, the base and the offset can be obtained from the table to form the load address. A load using the early-resolved address can proceed without waiting for the true address and is referred to as a *Predicted Load (P-load)*. The correctness of a P-load will be verified against its regular load.

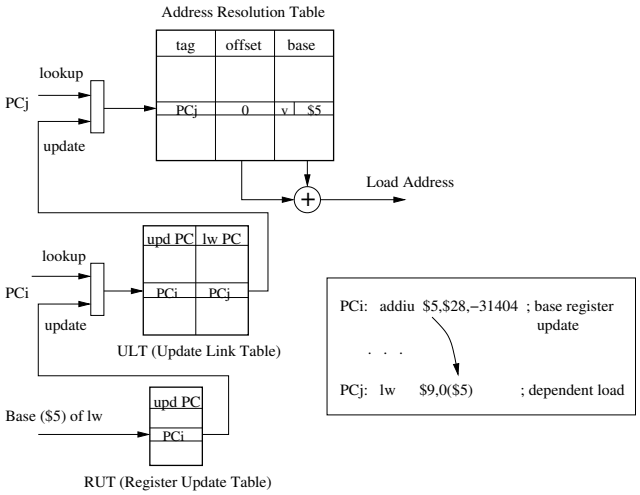

Figure 2: The dependence-linked address resolution.

3.1 Fast Path for Predicted Load

The execution of a P-load involves two phases. First, based on established dependence links, an early resolution of load address is initiated by the instruction that updates the base register. Second, when a load is fetched, the P-load can be issued using the early-resolved address from the ART to access the cache. Therefore, a P-load can start only when the load is actually fetched. This conservative approach can prevent any P-load from issuing by an incorrect dependence links. It is conceivable that these two phases are overlapping. Under a tight dependence distance, the dependent load may reach the fetch stage before the new base address becomes available. The advantage of a P-load may disappear because both the regular and its P-load are waiting for the new base address.

A *fast path* using cache *way* prediction is considered to gain additional speed for the P-load [17, 6]. Instead of looking up the tag array, the cache *way* for the P-load can be obtained through a way history table. The predicted way along with proper index and offset bits forms the *cache coordinate* for direct accessing the cache data array. The correctness of the predicted way is verified after the way is obtained from the regular load. Besides faster speed, this fast path avoids tag array accesses for the P-load. Furthermore, a common P-load

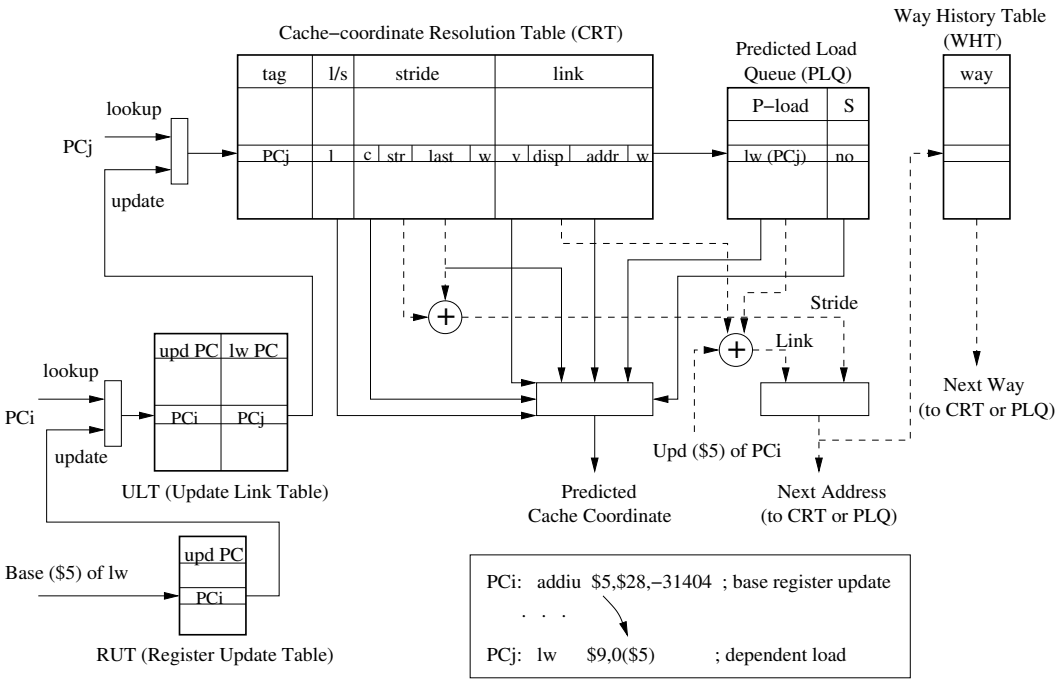

Figure 3: The integrated link/stride cache-coordinate resolution method.

recovery mechanism can be established to handle both mis-predicted addresses and ways.

Figure 3 illustrates an example design with an integrated cache-coordinate resolution method. *Solid arrows* indicate paths to establish update-load links and to access the predicted cache coordinate, while *dashed arrows* show an early resolution of a load address and its associated way. The predicted cache coordinate is saved and accessed from a *Cache-coordinate Resolution Table (CRT)* that is expended from the ART. A *Way History Table (WHT)* records the past way information for way predictions. Finally, a *Predicted Load Queue (PLQ)* is for handling the early address resolution and the execution of the P-load. Detailed functions of these tables will be described in subsequent sections.

3.2 Confidence and Stride Predictor

The accuracy of the dependence-linked address resolution depends primarily on correct base addresses. There are two reasons for an incorrect base address. First, a correct link may be missing. As a result, stale information may be obtained from the CRT. Second, a base register update has not completed when the update instruction is close to the dependent load. To achieve high accuracy, a P-load may be delayed until the new base address becomes available.

A *confidence* mechanism is instituted to handle these two cases. Two status bits, v are associated with each entry in the CRT to provide three prediction states. The state is *valid* if a newly predicted cache coordinate has been saved in the CRT. The state is *invalid* if no new base address has produced since the last instance of the load. Lastly, the state is *wait* if a new base address is currently being produced. To improve accuracy, a P-load is issued only with a *valid* cache coordinate. In case of a *wait* state, the P-load will be delayed until the new cache coordinate is predicted. No P-load is created when the state is *invalid*.

Recall that a *stride-based* predictor can resolve accurately the next load address for constant increment/decrement address patterns. Such an effective stride-based predictor can complement the dependence-linked address solver for reducing the impact of missing links and short dependence distances. Three selection strategies are considered. The *link-first* strategy prefers the link-based solver unless the cache coordinate in the CRT is invalid. The *stride-first* strategy selects the stride-based predictor as long as it reveals high confidence using simple saturation counters. The third *hybrid* selection strategy borrowed from [3] institutes a separate accuracy counter for selecting the stride or the link resolution when both are confident and valid. Similar to the stride/context predictor [3], both stride and link-based predictions are performed for each load and the prediction table is updated accordingly.

As shown in Figure 3, the *tag* in the CRT is for matching a load instruction. For achieving fast CRT

look-up, tag may be omitted using a direct-mapped design. The *l/s* represents a 2-bit up-down counter for the hybrid selection strategy. This counter is incremented or decremented when either link or stride resolution is correct. A fixed value can be assigned for other static selection mechanisms. The *stride* predictor maintains four values, a confidence counter *c*, a stride value *str*, the *last* (predicted) address, and the predicted way *w*. The *link* resolution also has four values, prediction status bits *v*, a displacement *disp*, the early-resolved address *addr*, and the predicted way *w*. According to the confidence mechanism, a proper predictor can be selected based on the values of the *l/s*, *c* and *v* bits. A truth table with respect to these bit values for selecting the predictor is given in Table 1.

l/s	c	v	pred	l/s	c	v	pred
link	lo	inv	no	strd	lo	inv	no
link	lo	valid	link	strd	lo	valid	link
link	lo	wait	link	strd	lo	wait	link
link	hi	inv	strd	strd	hi	inv	strd
link	hi	valid	link	strd	hi	valid	strd
link	hi	wait	link	strd	hi	wait	strd

Table 1: Selection of link/stride cache coordinate resolution and prediction.

3.3 Predicted Load Queue

A *Predicted Load Queue (PLQ)* is established to handle both early address resolutions and executions of the P-loads (Figure 3). A P-load can be in three states (*S*) in the PLQ: *no-trigger, ready-trigger*, and *issued*. Let's first consider a simple case when two execution phases of a P-load are not overlapping. On a ULT match, a P-load is inserted into the PLQ in the *no-trigger* state waiting for the address resolution and cache coordinate prediction. The CRT entry for the load is also changed to *wait*. A correct data dependence is built by renaming the base register of the load to the RUU location of the base register update. The corresponding CRT entry is updated and marked it *valid* once the predicted cache coordinate is available. The P-load is then removed from the PLQ. The P-load will enter the PLQ again in the *issued* state when the load is fetched and P-load is initiated for execution. This time the CRT entry is reset to *invalid*. The function and the content of the PLQ is similar to that of the LSQ for handling memory dependence and out-of-order execution. Note that the stride/context predictor also requires a queue structure for handling early execution of a load with a predicted address.

Under a tight dependence distance, the dependent load may reach the fetch stage before the new base address becomes available. A bypassing mechanism is designed to allow the P-load to be renamed and issued directly from the PLQ. Again, a P-load is first inserted in the PLQ in the *no-trigger* state by the base register update. The CRT entry is set to *wait* to delay the execution of any P-load. When the load is fetched, the corresponding P-load in the PLQ is upgraded to the *ready-trigger* state. Meanwhile, the CRT entry is reset to *invalid*. When the predicted cache coordinate is ready, the *ready-trigger* P-load in the PLQ will be upgraded to the *issued* state in the PLQ for an early execution. It is conceivable that multiple instances of a dependent P-load are initiated by multiple instances of the same base update. In this case, each instance of the P-load will be renamed to a different PLQ entry to receive the cache coordinate produced by the corresponding instance of the update. Note that the predicted cache coordinate is saved in the CRT and marked it *valid* only when the P-load in the PLQ is *no-trigger*. Therefore, only the latest cache coordinate is saved in multiple instance cases.

Assume that the dependence link from *addiu* to *lw* has established in Figure 3. A P-load (*lw*) is inserted in the PLQ in the *no-trigger* state when the *addiu* is fetched. The corresponding entry in the CRT for the *lw* is set to *wait*. When the *addiu* reaches the writeback stage before the fetch of the *lw*, the predicted cache coordinate is saved in the location for the *lw* in the CRT and becomes *valid*. The *lw* is then removed from the PLQ. The P-load may re-enter the PLQ in the *issued* state when the *lw* is fetched later. The CRT state becomes *invalid* afterwards. If the *lw* is fetched before the writeback of the *addiu*, the *lw* in the PLQ is upgraded to *ready-trigger* and the *lw* in the CRT is reset to *invalid*. This allow the P-load *lw* to be triggered directly from the PLQ.

Due to dynamic branch behavior, it is possible that the *addiu* is executed twice before the *lw* is encountered. Depending on the dynamic execution distance between the two *addiu*, the first useless early resolution of a *lw* can either be overwritten in the CRT or be nullified in the PLQ by the second and useful address resolution. On the other hand, it is also possible that the *lw* is executed twice before an *addiu*. In this case, the first *lw* will reset the corresponding CRT state to *invalid*, thus prevent any P-load for the second *lw*.

A P-load may fetch an incorrect data even with a correct cache coordinate if the P-load bypasses an early store that writes to the same memory location in the program order. After a P-load is issued for execution in the PLQ with the predicted cache coordinate, the P-load can be scheduled to access the cache right

Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13
Normal execution:													
lw	If1	If2	Dp1	Dp2	Sch	Agn	Ca1	Ca2	Ca3	Wb	Cm		
sub	If1	If2	Dp1	Dp2	-	-	-	-	-	Sch	Exe	Wb	Cm
With P-load using predicted cache-coordinate:													
lw	If1	If2	Dp1	Dp2	Sch	Agn	Ca1	Ca2	**Cm**				
P-lw	CRT	Sch	Ca1	Ca2	**Wb**								
sub	If1	If2	Dp1	Dp2	Sch	Exe	Wb	-	Cm				

Figure 4: Pipeline execution without/with the P-load (*P-lw*); *sub* follows and depends on *lw*

away. However, such a P-load must be cancelled later if a memory dependence is found with respect to early stores. One intuitive approach is to detect memory dependences by the regular load. However, the P-load can run many cycles ahead before the detection of such a violation. Existing memory dependence prediction techniques can help to stop issuing the P-load if a memory dependence violation is predicted [9]. We consider an alternative solution to handle memory dependences. The P-load always obeys memory dependences with respect to any early stores in the LSQ. In other words, the P-load will not be issued until the memory dependence is resolved. Whenever a store address is generated, any P-load in the PLQ that targets the same memory location is canceled if the corresponding regular load has not entered the LSQ. In other words, any P-load that could potentially run ahead of the store on which the load depends will be canceled.

3.4 Cache Way and Miss Predictor

Way predictions are carried out through the *Way History Table (WHT)*. Certain low order bits of cache line addresses are used to index into the WHT. The WHT is updated on a cache miss for both the replaced and the requested lines. The replaced line is marked as a *miss* except when another line in the same set is indexing to the same WHT entry. The new way of the requested line is updated in the WHT. For fast way prediction, a simple direct-mapped design without address tags is considered. In a direct-mapped design, multiple cache lines with the same low-order address (index) bits share the same entry. In general, large WHTs that minimize conflicts can provide high prediction accuracy. Marked replacement in large WHTs may provide a good coverage of cache misses.

A prefetch using the predicted load address can be initiated when a cache miss is predicted. Highly accurate way predictions also suggest to verify the way first before accessing the data array for the regular load. The data array is accessed twice by the regular and its P-load only when the prediction is incorrect.

4 Integrated Microarchitecture

We consider an example design with 8-cycle integer pipeline that is extended from the basic SimpleScalar pipeline [5]. We assume instruction and data cache accesses take two *If1, If2* and three *Ca1, Ca2, Ca3* cycles respectively to reflect the increasing cache delay. In addition, a separate address generation cycle *Agn* is required. We further assume instruction decode, rename and dispatch take two cycles *Dp1, Dp2* plus one cycle *Sch* to schedule instructions to the execution unit or the data cache. These additional cycles reflect complexity in designing the front-end pipeline [19, 15, 13]. The rest of the stages including execution *Exe*, writeback *Wb* and commit *Cm* remain unchanged. The integer pipeline is extended to eight cycles, while loads take eleven cycles. Figure 4 illustrates pipeline executions of two adjacent instructions, a load *lw* followed by a subtraction *sub*. Both instructions start at cycle 1. The *sub* consumes the data loaded by the *lw*. Due to this data dependence, the *sub* is delayed from issuing for five cycles until the writeback (cycle 10) of the *lw*.

4.1 Execution of P-Load

The P-load based on early address resolution can save a few cycles as shown in the lower half of Figure 4. At the first cycle, the predicted cache coordinate is fetched from the CRT. The P-load, denoted by *P-lw*, is inserted into the PLQ in the next cycle and is scheduled to access the data array directly. Since the entire tag matching path can be skipped we assume a data array access takes two cycles. The *P-lw* is completed and removed from the PLQ after storing the result and other related information, such as the predicted address and way, into the LSQ entry reserved for the regular load at the fifth cycle. In case that the normal load is delayed at the dispatch stage, the result is saved in the PLQ

Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
6-cycle dependence distance:																	
add	If1	If2	Dp1	Dp2	Sch	Exe	Wb	Cm									
lw							If1	If2	Dp1	Dp2	Sch	Agn	Ca1	Ca2	**Cm**	(*Wb*)	(*Cm*)
P-lw							CRT	Sch	Ca1	Ca2	**Wb**						
0-cycle dependence distance:																	
add	If1	If2	Dp1	Dp2	Sch	Exe	Wb	Cm									
lw	If1	If2	Dp1	Dp2	-	-	Sch	Agn	Ca1	Ca2	Ca3	**Cm**	(*Cm*)				
P-lw	CRT	-	-	-	-	-	PLQ	Sch	Ca1	Ca2	**Wb**						

Figure 5: Execution examples with 6-cycle and 0-cycle dependence distance from *add* to *lw*

temporarily. Early updates of the reorder buffer may further wake up dependent instructions speculatively. Therefore, the *sub* can be issued without any stall.

The *P-lw* can be canceled for two reasons. First, a predicted load address/cache coordinate is incorrect. Second, a memory dependence violation occurs. For a regular load, the load address that is available after the *Agn* cycle is used to verify against the early-resolved address of the *P-lw*. There is a slim chance that the incorrect cache coordinate is due to the way prediction. This can be detected after the second cache access cycle of the regular load. With regard to memory dependence violations, any early *P-lw* in the PLQ that targets the same memory location of the current store is canceled if the corresponding regular load enters the LSQ later than the store. When a misprediction or a memory dependence violation is detected, the *P-lw* along with its early-triggered dependent instructions must be canceled and restarted using the correct load value.

The *lw* is allowed to commit early according to the program order after correctness of the *P-lw* is verified. Any memory dependence violation or an incorrect *P-lw* address can be captured at the address generation of the regular load (cycle 6). However, the *lw* will not be committed until the predicted *way* is also verified (cycle 9). After the verification, the regular load can be merged with the *P-lw* by simply changing the state of the *lw* to normal. Similarly, the state of any early-triggered dependent instructions by the *P-lw* is also changed to normal to continue fast execution paths.

4.2 Variable Update-Load Distances

We show examples with 0 and 6 cycle distances in Figure 5, in which we assume the *lw* uses the value produced by the *add* as the base address. When the dependence distance is 6 cycles or longer, the *P-lw* starts in parallel with the regular load *lw* at cycle 7. The new predicted cache coordinate available at cycle 7 is forwarded in time for the *P-lw* in parallel with the CRT update. The *P-lw* updates the LSQ at cycle 11 instead of cycle 16 as would be done by the regular load. Consequentially, instructions that depending on the load value can potentially start 5 cycles earlier. After the correctness of the *P-lw* is verified, the *lw* can be committed at cycle 15. Without the *P-lw*, the writeback and commit stages would be done at cycle 16 and 17.

When the *add* and the *lw* are fetched at the same cycle, i. e. a 0-cycle distance, the *P-lw* is forced to wait until the new base address has computed (cycle 6 in the example). The regular load is able to schedule one cycle earlier after writeback of the *add*. However, the *P-lw* can still win by one cycle using the predicted cache coordinate. Thus, for dependence distances ranging from 0 to 6 cycles, the P-load can be advanced from 1 to 5 cycles ahead of the regular load in this baseline microarchitecture. Advanced techniques such as *register tracking* [4], or triggering critical register updates early can further advance the P-load with short dependence distances. More discussions in this direction can be considered as future work.

5 Performance Evaluation

Performance evaluations of load address resolution methods are carried out on SimpleScalar-based simulators. We will first compare the accuracy of these methods using functional simulation models. In functional simulations, verifications of load address and predicted way as well as updates of various tables occur when a load is fetched and executed. This functional accuracy is independent of pipeline implementations and can be used to display the potential of the proposed methods. In addition, for comparison purposes, functional accuracy is more along the lines of early published results. Recall that a modified SimpleScalar model with an 8-cycle, out-of-order integer pipeline and 11-cycle memory loads has been discussed. The IPC improvement on the modified model will also be given. Some simulation

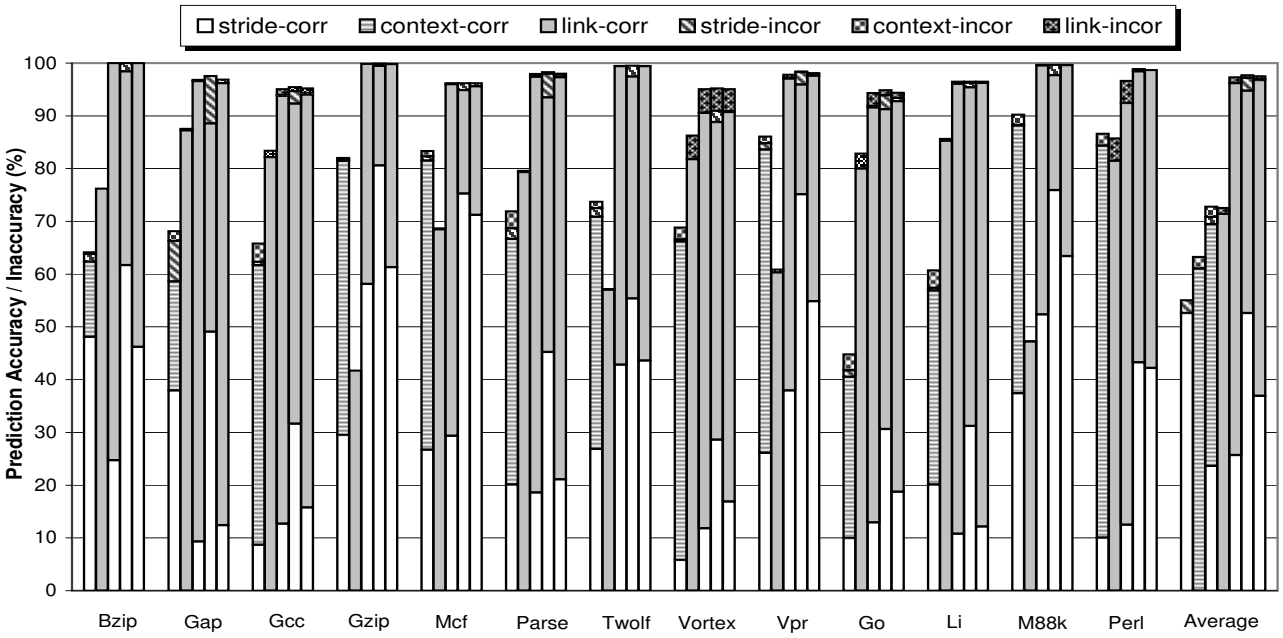

Figure 6: Accuracy of five address resolution methods: *hybrid stride/context, link-only, link-stride, stride-link, dynamic link/stride*, illustrated from left to right for each workload; on the average column, two more methods: *stride-only, context-only* are added on the left; the accuracy of each method broken into six categories (from bottom to top): *stride-correct, context-correct, link-correct, stride-incorrect, context-incorrect*, and *link-incorrect*; the space above each bar represents no-prediction.

Parameters	Size
Issue width	8
Functional units (+/*)	8/2
Memory port	4
Bimodal branch predictor	8K entry, BTB
RUU/LSQ/PLQ size	128/64/32
L_1 (I/D)/L_2 caches	32KB, 4-way/1MB, 4-way
L_1 (I/D)/L_2/memory delay	2/3/10/100
Stride/Context predictor:	total: 78KB
Load buffer (LB)	4K entry (51KB)
Link table (LT)	4K entry (27KB)
Link/Stride predictor:	total: 88KB
CRT	4K entry (61KB)
ULT/RUT	4K entry (24KB)
WHT	8K entry (3KB)

Table 2: Simulation parameters.

parameters are summarized in Table 2.

Several integer programs, *Bzip, Gap, Gcc, Gzip, Mcf, Parse, Twolf, Vortex, Vpr* from SPEC2000, and *Go, Li, M88k, Perl* from SPEC95, are used. The SPEC2000 binaries are generated by ssbig-na-sstrix-gcc/g++, version 2.7.2.3 compiler with option: -funroll-loops -O2. For each workload, we skip the first 200 million instructions, then collect simulation statistics from the next 200 million instructions.

5.1 Accuracy of Address Resolution

We first compare the accuracy of seven early address resolution methods. The first three: *stride-only, context-only* and *hybrid stride/context* [3] are existing methods. The next four are all link-based: *link-only, link-stride, stride-link*, and *dynamic link/stride*, where the last three are link and stride combined with the respective *link-first, stride-first*, and *hybrid* selection. The same prediction table size is used for all the methods. Both the CRT in the link-based scheme and the load buffer in the context-based scheme have 4K entries with 2-way set-associativity. Both the ULT and the link table have 4K entries with 16-way set-associativity. As shown in Table 2, the link/stride method requires a slightly bigger CRT for saving the predicted address and way comparing with that in the stride/context method. Other *stride-only*, *context-only*, and *link-only* methods use the same table size with less hardware requirement. Note that for better accuracy, context-based schemes predict base addresses [3].

Address Resolution Accuracy: The results in Figure 6 show superior address resolution accuracy us-

ing the dependence-linked address solver. Comparing with the *hybrid stride/context* scheme, the *link-only* address solver already shows a slight edge with 71.4% average accuracy against 69.4% in hybrid, and 1.1% inaccuracy against 3.3% in hybrid. Combined with the link and the stride predictors, the accuracy improves dramatically for all three selection mechanisms. The dynamic selection has the best results. It achieves 96.8% average accuracy with merely 0.76% misprediction rate. *Vortex* is the only exception among the workloads. It has 90.8% correct resolution with 4.3% incorrect rate. *Vortex* shows the highest number of dependent loads for each base register update, which causes conflict misses in the ULT.

Such high accuracies reveal three important facts. First, dynamic update-load dependence links can be built quickly and accurately, and the working set of the dynamic dependence history can be held in moderate-size tables. Second, the stride predictor complements the dependence-linked address solver almost perfectly. About 37.0% out of 96.8% of accurate resolutions are contributed by the stride predictor. Most of them belong to no-prediction cases in the *link-only* scheme due to a lack of new base address. Third, the dynamic selection of link and stride mechanisms is more accurate than the static selection mechanisms.

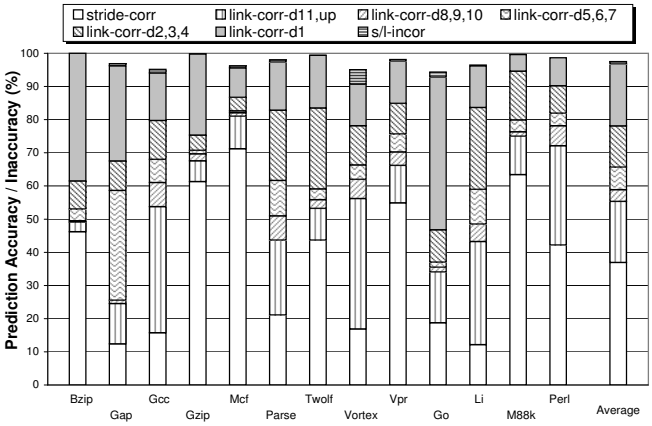

Figure 7: Dependence distance distribution.

Dependence Distance: The performance impact of the link-based address solver is constrained by the distance between a register update and its dependent loads. In Figure 7, the distance distribution for accurate link-based address resolutions is plotted. Although about 30% of accurate link-based resolutions have the update immediately preceding the load with distance of 1, there are also equal amount of distance 11 and longer for those accurate link-based resolutions. Since average IPCs for the selected workloads is quite low ranging from 1 to 2, dependence distance of 8 or longer may take the full advantage of the direct-load method. Together with a stride predictor, a total of 60% of predictions can advance the loads by 5 cycles. This update-load distance distribution varies among the workloads. *Bzip, Gap, Gzip,* and *Go* have relatively short update-load distances, while *Gcc, Vortex, Li,* and *Perl* have much longer distances.

Table Size and Topology: Prediction table sizes and topology likely influence the prediction accuracy. In Figure 8, average results of a sensitivity study for the 13 workloads are plotted. We varies both CRT and ULT sizes from .5K to 8K entries. We also simulate four set-associativities, 1, 4, 8, and 16 for the ULT with a fixed set-associativity of 2 for the CRT. As expected, larger tables with higher set-associativities provide better resolution accuracy. The results also indicate high set-associativity of the ULT is essential since each register update may trigger multiple loads and thus requires multiple links in the same set. From these results, we pick 4K entries for CRT/ULT with 16-way ULT for detailed pipeline simulations.

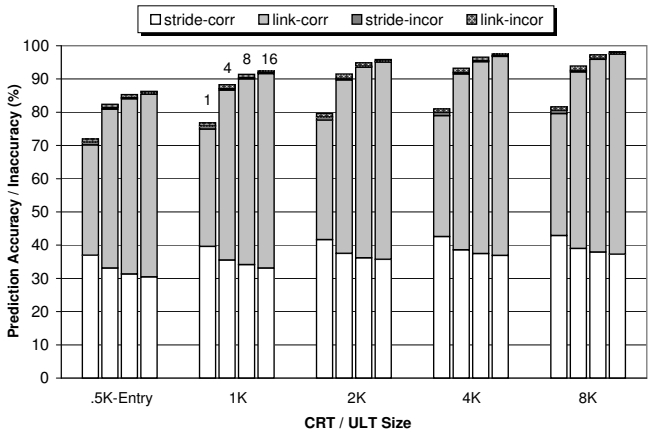

Figure 8: Sensitivity of address resolution accuracy with respect to CRT/ULT size and ULT set-associativity.

Multiple Dependent Loads and Updates: We measure two parameters with respect to the number of dependent loads for each dependence-linked base register update, and the number of base register updates received for each issued P-load. The results are shown in respective left and right bars with annotated averages in Figure 9. A majority of linked base register updates (66%) have a single dependent load, while a majority of P-load (87%) receives only a single base update as indicated by *1-load/upd,upd/load.* However, the average dependent loads are ranging from 2.12 to 2.40 for various CRT/ULT sizes indicating that multiple links in the ULT for each base update and sufficient ports to access the CRT are necessary. These multiple dependent

loads, such as stack accesses, are normal in applications. The average updates received per issued P-load, on the other hand, are ranging from 1.32 to 1.38. This indicates over 30% of early resolutions are useless due to multiple execution paths. However, since a P-load can be issued only when the load is fetched, these extra updates will not produce any P-load.

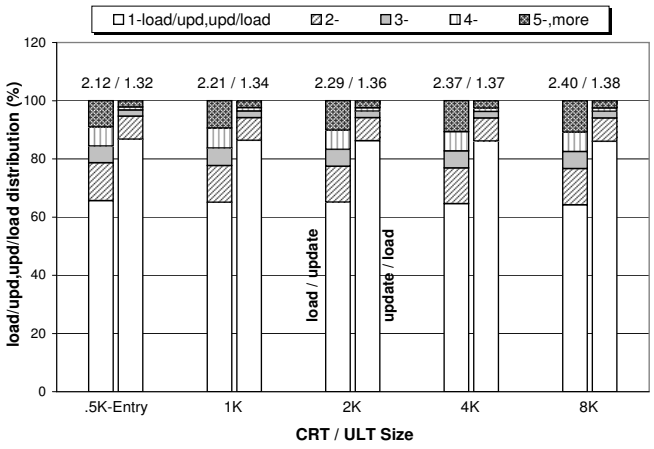

Figure 9: Average number of dependent loads per update, and average number of updates per load.

Fast CRT Design: The time required to access the CRT for a predicted cache coordinate is critical in the proposed method. We simulate a direct-mapped CRT with/without tag comparison when accessing the cache coordinate. As shown in Table 3, the results of direct-mapped CRTs are comparable to that of a 2-way CRT. Without tag comparison, inaccuracy increases slightly. Overall, the early resolution accuracy is rather insensitive with respect to the set-associativity of the CRT.

CRT	correct	str-cor	link-cor	str-no	link-no
1w-notag	95.26	36.16	59.10	0.41	0.78
1w-tag	95.26	36.16	59.10	0.25	0.45
2w	96.77	36.92	59.85	0.26	0.50

Table 3: Accuracy of link-based resolution using direct-mapped CRT with/without tag comparison.

Confidence Mechanism: Next, we evaluate the early resolution accuracy with respect to the number of base register updates received for each P-load. Table 4 shows the average result of 13 workloads. Note that in order to know the accuracy of 0-update, we ignore the *valid* bit in this set of simulations. As a result, there are tiny difference in comparison with other results because of the change in updating the selection counter. It is observed that 0-update is only account for 1.70% out of 63.12% total link-based resolutions, however, the incorrect rate is measured at .72% out of the total 1.29%. Although the absolute value is small, it represents 56% of the total inaccuracy. We observed similar results in the *link-only* scheme, where the percentage of 0-update is much higher. The results support our confidence mechanism to trade high-probability of misprediction with no-prediction when a 0-update is encountered.

	total	0-upd	1-upd	2-upd	3-upd	4&up
corr	61.83	0.98	51.83	4.84	1.54	2.64
incor	1.29	0.72	0.38	0.04	0.05	0.10
incor%	2.09	42.43	0.72	0.72	3.10	3.67

Table 4: Accuracy with respect to number of updates.

Way and Miss Prediction Accuracy: Finally, the average accuracy of cache way and miss prediction using the WHT is given in Table 5. We consider the prediction accuracy only when the early-resolved load address is correct (96.77%). WHTs that were used in the simulation can hold the way and miss information for 2 to 16 times of the number of cache lines in the L_1 data cache. We also show cache hit/miss ratios when addresses are correct. All WHTs are direct-mapped without tags. With an 8-time WHT, correct way and miss predictions with respect to the actual cache hits/misses are reaching 99.9% and 94.8% respectively as shown in the parentheses. This high miss prediction rate is due to records the recent evicted lines. The high accuracies support fast way and miss predictions without requiring early cache tag array accesses. Such high accuracies also suggest to verify the way prediction first for the regular load to avoid accessing the data array twice.

WHT	way-corr	way-no	miss-corr	miss-no
2x	93.90 (99.6)	0.96	1.80 (72.3)	0.11
4x	94.14 (99.9)	0.42	2.15 (86.4)	0.06
8x	94.21 (99.9)	0.18	2.36 (94.8)	0.02
16x	94.26 (100.)	0.06	2.44 (98.0)	0.01
cache	94.28 (hit)	0	2.49 (miss)	0

Table 5: Accuracy of cache way and miss prediction, measured with correct address prediction (96.77%).

5.2 Integrated Pipeline Performance

We now compare the IPC improvement of the dynamic link/stride address resolution and the hybrid stride/context predictor over the base model without address resolution. As shown in Figure 10, the aver-

age IPC increases by 18.1% and 9.5% respectively. In these simulations, we assume there are two read ports and two write ports for both the CRT and the ULT. We also assume a perfect scheduler for updating the CRT with precise recovery upon a mis-prediction. In detailed pipeline simulation, we have experienced two major difficulties in improving IPCs.

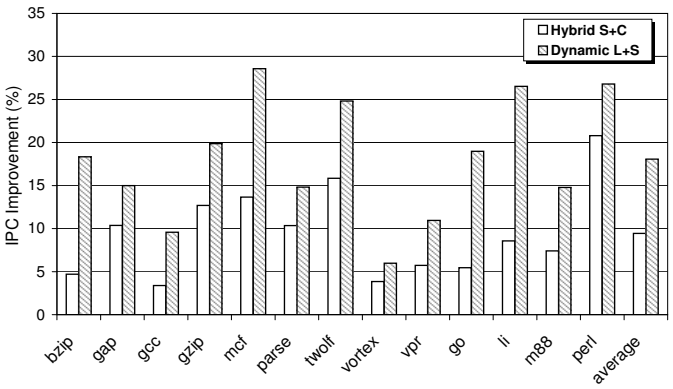

Figure 10: IPC comparison of base, hybrid and direct-load methods.

Prediction Accuracy: In the pipeline model, the prediction table is not updated until the true load address is generated. This delay of updating the CRT may reduce the accuracy of subsequent predictions. For example, in a tight-loop of using the stride predictor, a wrong address can be produced based on the *old* current address. In addition, address resolutions and updates of RUT, ULT and CRT could potentially happen on a mis-predicted path. These incorrect updates can affect the following address resolutions in the correct execution path. We did not attempt to make adjustment on these issues as we observe the inaccuracy of address prediction increases to about 3-4% in pipeline simulation. Similarly, although the dependence links can be built at the decode cycle of a load, the next register update may already be fetched and misses the correct link in the ULT as the no-prediction rate increases from 2.4% to nearly 5%.

Memory dependence violation: Memory dependence violations present another major problem. We found out near 10% of the correctly resolved addresses are canceled due to memory dependence violations. Detailed analysis shows that such short store-load memory dependences occur fairly often in SimpleScalar assembly codes. For instance, because of dynamic execution paths, a register may need to be free at the end of a basic block by storing the content back to memory. However, the stored data is often required at the beginning of another basic block. When those two blocks are adjacent at runtime, a tight memory dependence is formed and a P-load can run ahead of the store. Such a high degree of cancellation may incur heavy penalty because the P-load could potentially run far ahead before the store address is resolved. To alleviate this penalty, a P-load is held in the PLQ after accessing the data array when any store has entered the LSQ between the P-load and the regular load. After memory dependence is resolved by the regular load, the P-load updates the LSQ with the data. In this case, the P-load still has a three-cycle advantage over the regular load.

The above problems exist in other address prediction methods. The average useful address resolution is dropped to about 82%, which is still significantly higher than the useful prediction rate (60%) of the hybrid stride/context predictor in the pipeline execution mode. There are several directions to improve the pipeline performance. First, multiple stride predictions using a single current address may improve accuracy in a tight-loop situation [3]. Second, proper recovery mechanisms may be used to correct various table updates from wrong execution paths. Third, an accurate memory dependence predictor may be added to allow those P-load without memory dependence violations to be executed earlier.

6 Conclusion

We have shown a highly-accurate dependence-linked address resolution method for hiding the increasing cache latency in modern microprocessors. This dependence-linked address solver can handle those load addresses that are inherently irregular and difficult to predict. We demonstrated that dynamic dependence links can be established accurately between the instruction where a base address is produced to the load instruction. Given correct links, a new base address can trigger address calculation and cache coordinate resolution in a dataflow fashion to achieve 0-cycle latency for a significant portion of loads. The proposed method is especially appealing for building large first-level caches without compromising fast clock rate.

Performance evaluation based on a functional model using SPEC integer programs suggested that the dependence-linked address resolution mechanism can achieve 96.8% accuracy with as little as 0.76% misprediction rate. Preliminary studies also show that the proposed method has a potential to improve the execution time by about 18% on a modified SimpleScalar model. When an advanced stride/context hybrid predictor is used, the link-based method can still improve the overall performance by about 9%.

Acknowledgment:

The authors would like to thank Jean-Loup Baer and

John Shen for their feedback on this paper. Anonymous referees provide very helpful comments. This work is supported in part by NSF grants MIP-9624498, EIA-0073473 and by Intel research donations.

References

[1] P. Ahuja, J. Emer, A. Klauser, and S. Mukherjee, "Performance Potential of Effective Address Prediction of Load Instructions," *Prod. of 2001 Workshop on Memory Performance Issues*, July 2001, (12 pages).

[2] T. Austin and G. Sohi, "Zero-cycle loads: microarchitecture support for reducing load latency", *Proc. of 28th annual international symposium on Microarchitecture* , Ann Arbor, MI, Dec. 1995, pp. 82–92.

[3] M. Bekerman, S. Jourdan, R. Ronen, G. Kirshenboim, L. Rappoport, A. Yoaz, and U. Weiser, "Correlated Load-Address Predictors," *Proc. of 26th Annual Int'l Symp. on Computer Architecture*, Atlanta, GA, May 1999, pp. 54–63.

[4] M. Bekerman, A. Yoaz, F. Gabbay, S. Jourdan, M. Kalaev, and R. Ronen, "Early Load Address Resolution Via Register Tracking," *Proc. of 27th Annual Int'l Symp. on Computer Architecture*, Vancouver, Canada, June 2000, pp. 306–315.

[5] D. Burger and T. Austin, "The SimpleScalar Tool Set, Version 2.0", Technical Report #1342, CS Department, Univ. of Wisconsin-Madison, June 1997.

[6] B. Calder and D. Grunwald, "Next Cache Line and Set Prediction," *Proc. of 22nd Int'l Symp. on Computer Architecture*, S. Margherita Ligure, Italy, June 1995, pp. 287–296.

[7] C. Chen and A. Wu, "Microarchitecture Support for Improving the Performance of Load Target Prediction," *Proc. of 30th annual international symposium on Microarchitecture*, Triangle Park, NC, Dec. 1997, pp. 228–234.

[8] B. Cheng, D. Connors, and W. Hwu, "Compiler-Directed Early Load-Address Generation," *Proc. of 31st annual international symposium on Microarchitecture* , Dallas, TX, Dec. 1998, pp. 138–147.

[9] G. Chrysos and J. Emer, "Memory Dependence Prediction using Store Sets", *Proc. of 25th Int'l Symp. on Computer Architecture*, Barcelona, Spain, June 1998, pp. 142–153.

[10] R. Eickemeyer and S. Vassiliadis, "A Load-Instruction Unit For Pipelined Processors," *IBM Journal of Research and Development*, Vol. 37(4), pp. 547–564, July 1993.

[11] A. Farcy, O. Temam, R. Espasa, and T. Juan, "Dataflow Analysis of Branch Mispredictions and Its Application to Early Resolution of Branch Outcomes," *Proc. of 31st annual international symposium on Microarchitecture* , Dallas, TX, Dec. 1998, pp. 59–68.

[12] J. Gonzalez and A. Gonzalez, "Speculative Execution via Address Prediction and Data Prefetching," *ACM 1997 Int'l Conf. on Supercomputing*, Vienna, Austria, Aug. 1997, pp. 196–203.

[13] T. Horel and G. Lauterbach, "UltraSPARC-III: Designing Third-Generation 64-Bit Performance", *IEEE Micro*, May/June 1999, pp. 73–85.

[14] K. Hua, A, Hunt, L. Liu, J-K. Peir, D. Pruett, and J. Temple, "Early Resolution of Address Translation in Cache Design," *Proc. of 1990 Int'l Conf. on Computer Design*, Boston, MA, Sep. 1990, pp. 408–412.

[15] R. Kessler, "The Alpha 21264 Microprocessor," *IEEE Micro*, Vol. 19(2), March/April 1999, pp. 24–36.

[16] M. Lipasti, C. Wilkerson and J. Shen, "Value Locality and Load Value Prediction", *Proc. of the 7th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, Boston, MA, Oct. 1996, pp. 138–147.

[17] L. Liu, "History Table for Set Prediction for Accessing a Set-Associate Cache," U.S. Patent 5,418,922, May 1995.

[18] W. Lynch, G. Lauterbach and J. Chamdani, "Low Load Latency through Sum-Addressed Memory (SAM)," *Proc. of 25th Annual Int'l Symp. on Computer Architecture*, Barcelona, Spain, June 1998, pp. 369–379.

[19] D. Papworth, "Tuning the Pentium Pro Microarchitecture," *IEEE Micro*, Vol. 16(2), April 1996, pp. 8–15.

[20] A. Roth, A. Moshovos, and G. Sohi, "Dependence Based Prefetching for Linked Data Structures," *Proc. of the 8th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, San Jose, CA, Oct. 1998, pp. 115–126.

[21] Y. Sazeides and J. Smith, "The Predictability of Data Values," *Proc. of 30th annual international symposium on Microarchitecture*, Triangle Park, NC, 1997, pp. 248–258.

[22] T. Slegel, et al., "IBM's S/390 G5 Microprocessor Design," *IEEE Micro*, Vol. 19(2), March/April 1999, pp. 12–23.

[23] P. Song, "IBM's Power3 to Replace P2SC," *Microprocessor Report*, Vol. 11(15), Nov. 1997, pp. 1–11.

[24] K. Wang and M. Franklin, "Highly Accurate Data Values Prediction using Hybrid Predictors," *Proc. of 30th annual international symposium on Microarchitecture*, Triangle Park, NC, Dec. 1997, pp. 281–290.