2025 IEEE International Symposium on High-Performance Computer Architecture (HPCA)

MASCOT: Predicting Memory Dependencies and Opportunities for Speculative Memory Bypassing

Karl H. Mose†, Sebastian S. Kim‡, Alberto Ros‡,
Timothy M. Jones†, Robert D. Mullins†
University of Cambridge†, University of Murcia‡
km781@cam.ac.uk, sebastiansumin.kim@um.es, aros@ditec.um.es,
timothy.jones@cl.cam.ac.uk, robert.mullins@cl.cam.ac.uk

***Abstract*—Memory-dependence prediction (MDP) increases instruction-level parallelism (ILP) by allowing load instructions to be issued even when addresses in the store queue are unknown. The predictor determines whether a load will alias with a prior store, delaying issue when a dependence is predicted. Speculative memory bypassing (SMB) further enhances ILP by short-circuiting a predicted dependence to forward the value written by a store to a load that is predicted to depend on it, without their addresses necessarily being known. This breaks data dependencies on the load and store addresses, allowing loads to obtain their values much earlier than they normally would.**

To obtain benefits, dependencies must be predicted with high accuracy. Furthermore, the benefits are skewed, with false negatives being more costly for performance than false positives for MDP, since the former requires squashing when the misprediction is identified, whereas the latter only delays the issue of independent loads. For SMB, on the other hand, false positives are very costly, as they require squashing, whereas false negatives have little impact in the presence of an accurate memory dependence predictor. Due to these differing requirements, the designs of predictors for these mechanisms have diverged.

In this paper, we propose MASCOT, a novel predictor capable of performing both MDP and SMB. MASCOT is inspired by the TAGE predictor, widely used in branch prediction. Although TAGE has proven effective as a universal predictor structure, we demonstrate how prior TAGE-based MDP or SMB predictors suffer from inaccuracy due to not learning patterns of non-dependence. By learning the context for dependencies as well as non-dependencies, MASCOT achieves sufficiently low false negatives and false positives to perform MDP and SMB, while at the same time uses less space than existing designs that only perform MDP or SMB.

Our simulation results show that for SPEC CPU 2017, MASCOT used for MDP alone yields an IPC gain of 0.4 % over the previous state-of-the-art predictor, on average, at the same size. When used for both MDP and SMB, it yields an increase in IPC of 1.9 % on average, with peak gains of 26 %. A compacted version of MASCOT, MASCOT-OPT, achieves similar numbers within 0.1 % while using just 10.1 KiB of space.

I. INTRODUCTION

High-performance out-of-order superscalar processors can increase instruction-level and memory-level parallelism (ILP and MLP) by speculatively executing load instructions early, before the addresses of all prior stores in program order are known. There are two opportunities to increase performance using this approach. In the first, loads that are predicted not to have a dependence on any older in-flight store instruction can be issued immediately, accessing the data cache. In the second, loads that are predicted to have a dependence with a specific prior store instruction can obtain their value directly from that store as soon as this value has been computed, irrespective of the addresses having been calculated. In both instances, loads can potentially obtain their value much earlier than otherwise, but must be squashed and re-executed if the speculative execution is found to be wrong.

In the past, researchers have taken advantage of these two opportunities, generally using separate, unrelated predictor schemes. Memory-dependence prediction (MDP) speculates on whether a load aliases a prior store [6], [11], [19], [28], [31]. It is important in this scenario to be liberal in making predictions—it is far more costly to predict a load to be non-dependent, issue it and its dependent instructions, and have to squash them all on a misprediction, than it is to hold the load up unnecessarily until all prior store instructions have their addresses known. Therefore memory-dependence predictors are historically skewed towards keeping the false-negative rate low, erring on the side of predicting a dependence when there is none, so as to keep the number of squashes low [6], [28], [31].

The complementary scheme is speculative memory bypassing (SMB) [16], [18], [26], [30], which predicts the exact older store on which a load depends (if there is one) and then forwards the value that would be stored to the load as soon as it is ready. Here, the load and its dependencies must be squashed if the predictor identifies the wrong store. Therefore, in this situation, predictors are skewed towards keeping the false-positive rate low, since predicting no dependence results in comparatively little overhead as long as the load is otherwise stalled correctly.

Due to this imbalance, prior work has considered predicting MDP or SMB opportunities separately. However, this leads to unnecessary overhead in predictor area and missed opportunities to improve ILP.

To address this, we propose MASCOT (Memory-dependence And Short-Circuit Optimising TAGE), a novel predictor for both MDP and SMB. MASCOT takes inspiration from TAGE, widely used for branch prediction [22]–[25], explored separately in the context of MDP [19] and SMB [18]. MASCOT records load-store dependencies and context-dependent non-dependencies, achieving state-of-the-art performance in terms of both false dependencies and missed dependencies. This

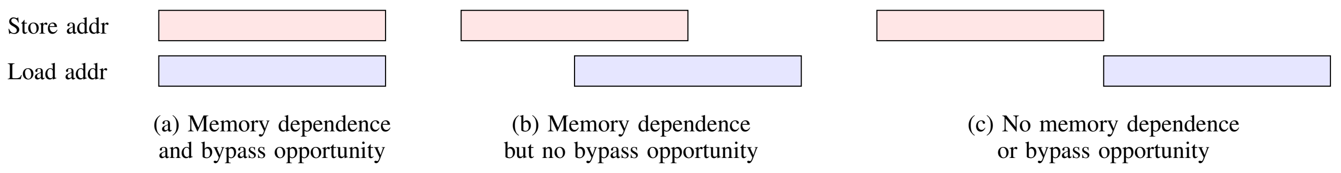

Store addr
Load addr
(a) Memory dependence and bypass opportunity
(b) Memory dependence but no bypass opportunity
(c) No memory dependence or bypass opportunity

Fig. 1. Examples of memory dependencies and bypass opportunities.

accuracy in both directions makes it uniquely suited for SMB. While SMB has been previously considered as not worth the effort given a good enough memory-dependence predictor [12], having a predictor that provides highly accurate predictions for both kinds of mis-speculation makes SMB much cheaper to implement, as it can solve both MDP and SMB using less space than other predictors that would traditionally solve just one of the two.

The key innovation in MASCOT is an optimised allocation policy that records context-dependent non-dependencies, alongside load-store dependencies. When a TAGE predictor used for branch prediction mispredicts, it allocates a new entry with the correct direction in the table with the next longest global history, so as to capture the additional context necessary to predict the branch the next time around. However, existing uses of TAGE for MDP only keep track of dependencies between loads and prior stores. They only allocate in the next table when a load's dependence is mispredicted *but it actually depends on a different prior store*. MASCOT differently allocates a new entry in the next table to record situations where the load does not depend on any store, using the global branch history as context to record this non-dependence. This drastically reduces the misprediction rate of false dependencies and allows the predictor to be more aggressive when predicting true dependencies.

Our simulation results show that MASCOT used solely as a memory-dependence predictor manages to beat the previous state-of-the-art predictor in PHAST [11] on the SPEC CPU2017 benchmarks, reducing MPKI by 70 % on average, and increasing IPC by 0.4 %. We further show that MASCOT can easily be extended to predict SMB opportunities, increasing IPC by a total of 1.9 % over PHAST. A size-optimised version of MASCOT, MASCOT-OPT, achieves a 1.8 % speedup over PHAST, while using just 10.1 KiB of space compared to 14.5 KiB for PHAST.

In summary, our contributions are:

- A TAGE-like predictor with an optimised allocation policy that learns context-sensitive non-dependencies (as well as dependencies) to achieve much higher accuracy than previous memory-dependence predictors.
- Unifying MDP and SMB into a single high-accurate predictor for false positives and false negatives to reduce memory overhead.
- A tuning strategy to reduce the area required by a TAGE-like predictor while maintaining high performance.

II. BACKGROUND

A. Memory-dependence prediction

Memory-dependence prediction (MDP) is an optimisation technique that allows the processor to speculate whether a load depends on an earlier in-flight store. When a predictor says that there is not a prior depending store, then the load can be issued as soon as its address has been calculated, rather than waiting for all prior stores to also obtain their addresses. If the predictor incorrectly predicts that a load has no dependencies on in-flight stores, the load may execute before its dependent store is ready, leading either to a spurious memory access or forward from an outdated store in the store queue, both scenarios requiring a squash. Alternatively, if the predictor says that a load is dependent on a store that it does not actually depend on then the load may stall unnecessarily. The penalty for the latter is much smaller (though not insignificant), and because of this, memory-dependence predictors have generally been designed to have many fewer false negatives (missed dependencies) than false positives (erroneous dependencies).

One of the earliest examples of MDP is Store Sets, introduced by Chrysos and Emer [6]. In this scheme, each load is allocated a store set, which contains the PCs of all stores that the load was previously dependent on. When a load enters the pipeline, it accesses its store set to look for dependent stores. If any of those are in flight, the load is delayed.

TAGE-MDP, first mentioned in a paper by Perais et al. [20], and most thoroughly explained by Kim and Ros [11], modifies the TAGE branch predictor to also predict memory dependencies. It is a relatively simple augmentation of TAGE, repurposing the 3-bit saturating counter to predict the store distance, and adding a single bit u to encode usefulness. If u is not 0, the entry can be used for predicting a memory dependence.

Recently, Kim and Ros developed the PHAST predictor [11]. PHAST employs a storage structure similar to TAGE, organising entries into tables with increasing context length. When making a prediction, all tables are looked up in parallel, using a hash of the load PC with increasing lengths of global history as the index and picking the entry from the table with the longest global history. PHAST entries have a 7-bit distance field, a 16-bit tag, a 4-bit usefulness counter and a 2-bit LRU field. It uses a unique allocation strategy, choosing to allocate to a given table based on the number of branches in between a given load-store pair. PHAST achieves state-of-the-art performance as a memory-dependence predictor.

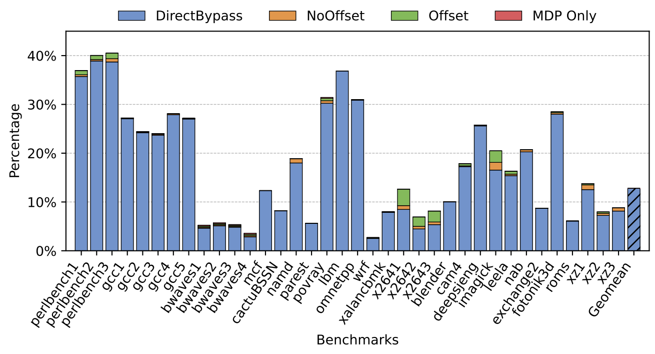


Fig. 2. Percentage of loads that have a dependence with a previous store.

B. Speculative memory bypassing

Speculative memory bypassing (SMB) is similar to MDP in that the processor speculates that a load depends on an earlier store. However, rather than stalling the load, SMB forwards the value that will be stored to memory directly to the predicted-dependent load without necessarily having computed load or store addresses [30].

Note that there is a subtle difference in the meaning of dependency here. In the context of MDP, a dependence means that the load may not execute before the store. For SMB, a dependence means that the store's source value provides the full value for the load. A partial load-store dependence, where a load might, for example, load half the value of a previous store and the remainder from memory, is a memory dependence but does not provide an opportunity for SMB.

Fig. 1 shows the three different cases that can occur when considering a load and a single prior store, where each box represents an address (red for a store and blue for a load), the length of the box represents the size of the access, and the placement of the boxes relative to one another represents their offsets in memory. A dependence arises when the memory accesses of the store and the load overlap (even with a single byte). Scenario (a) is the only situation with a dependence suitable for SMB, whereas there is still a dependence between the instructions in scenario (b), but it does not offer an opportunity for SMB as the load cannot be fully fed by the store.

1) Practicalities of SMB: Even with a dependence between a load and a prior store, there are multiple different scenarios in which forwarding can take place and logical operations that must be applied to a value to forward it correctly. The opportunities for SMB in our workloads are evaluated in fig. 2, using the experimental setup described in section V. Here, *DirectBypass* means that the prior store and load addresses and sizes are identical, so the load can directly bypass memory to get the stored value. *NoOffset* means that the load and store addresses are the same, but the store is larger than the load, so the value should be truncated during memory bypassing. *Offset* corresponds to a situation where the load can obtain the whole value through memory bypassing from a prior store, but its address starts at an offset from the store's address, which requires shifting and potentially truncating the value during memory bypassing. Finally, *MDP Only* refers to a memory dependence where the load would get only part of its data from the prior store and the rest from the memory system (or another, older in-flight store). Each bar shows a histogram of the dependencies, as a percentage of the number of loads executed.

As can be seen, although there are opportunities for SMB with a smaller load size and when the load is not aligned to the store, the overwhelming fraction of opportunities occur in the simple case where the load and store have the same size and are aligned. Note that this fraction varies by benchmark, with some, such as *perlbench* and *lbm*, having around 40 % of loads with SMB opportunities, whereas others, such as *bwaves* and *wrf*, having just around 5 % of loads with SMB opportunities.

2) Prior SMB schemes: There are many examples in the literature of SMB, with the two most notable examples described here.

In 2006, Sha et al. proposed NoSQ, which maps each dynamic load to its dependent store [26]. NoSQ proposes to remove the store queue and instead relies only on predictive bypassing. If the predictor does not predict a bypassing opportunity, the load is stalled until all prior stores have completed, meaning that NoSQ is effectively an SMB-only predictor. The NoSQ predictor is based on the GShare predictor [14], where one of its two predictor tables is XORed with a global-history vector, preferring the context-sensitive prediction when available. Each entry has a 6-bit store distance, a 3-bit offset used for misaligned addresses, 2-bits used for the store size, a 7-bit confidence counter, and a 22-bit tag.

NoSQ is one of the most sophisticated implementations of SMB that we are aware of, even covering cases such as partial-word bypassing, where an offset between a load and store address might require the microarchitecture to shift, sign extend or truncate the bypassed value.

Perais et al. provide a detailed implementation for physical-register sharing to enable optimisations such as SMB [17], [18]. Their IDist predictor is a TAGE-based predictor, which uses 2, 5, 11, 27 and 64 bits of global branch history combined with 16 bits of path history and the load PC. To minimise squashes, IDist only makes predictions when it is highly confident. Because of this, it is not suitable for memory-dependence prediction, and thus the authors implement it in conjunction with a 4 KiB store-sets predictor [6] for that purpose.

III. Challenges of using TAGE as a memory dependence predictor

TAGE has previously been used for both MDP and SMB separately, with notable examples being MDP-TAGE [19] and IDist [17]. Yet dealing with false dependencies creates a decision when designing memory-dependence predictors that does not occur in branch prediction, meaning that TAGE cannot be used directly for MDP to the best of its ability.

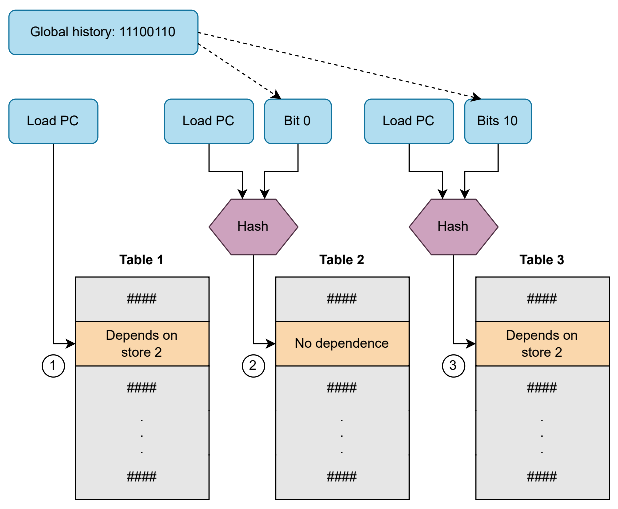


Fig. 3. MASCOT allocates non-dependence entries in higher-context entries when false dependencies are identified ① meaning a non-dependence can be predicted next time ②. Future dependencies can also create entries in higher-context tables ③.

If a memory-dependence predictor falsely predicts a non-dependent load to be dependent on a prior store, the designer is faced with a challenge. If the load is dependent on a different prior store, a new entry can be allocated as usual in TAGE, by creating that new entry in the table corresponding to the next-longest global history. However, if the load is not dependent on any prior store, the options are less obvious. It is likely that the context used in making the erroneous dependence prediction was inadequate (hence the misprediction), and one might wish to allocate a dependence in a higher-context table, but at the moment of misprediction, the longer history of the conflict is not available. Instead, the current global history has resulted in the load being non-dependent. To our knowledge, all TAGE-based memory-dependence predictors so far have opted to store only dependencies, and designers have opted to deal with this situation by decrementing the usefulness of the mispredicting entries. This leads to two issues: (1) learning a non-dependency can take a long time, especially for big counters; and (2) this does not strongly signal to the predictor that it has inadequate information.

A. Example

Consider, for example, a scenario where a load's dependency on a specific prior store distance is determined by the most recent branch in the global history, where if it is taken (70 % probability), the load is dependent on the store, but if it is not taken (30 % probability), it is non-dependent. Assume that the predictor has allocated an entry for this dependence in its first table, which is indexed using only the PC and no global history. The predictor now faces the following issues.

Prediction persistence. Given that the load is always either correctly predicted to be dependent, or otherwise fully non-dependent, there is no occasion where it is obvious to allocate a new entry.

Slow confidence adjustment. One approach might be to allocate a new entry with the same dependency if the current entry is found to have confidence 0. However, using a 3-bit counter initialised to the maximum value, it would take an expected 1,625 predictions[1] before the entry reaches confidence 0.

Eviction risk. Even with the scheme mentioned above, when the counter reaches zero, the entry also becomes vulnerable to eviction, reducing the likelihood of growing the prediction pattern to capture the branch-dependent behaviour.

Despite these issues, TAGE generally performs well if used for MDP or SMB alone. For MDP, this stems from the relatively low penalty from false dependencies in traditional memory-dependence prediction, since false dependencies lead only to loads being issued later than if the non-dependence had been identified. As long as the predictor allocates when a new load-store dependence is realised, false negatives are kept low. For SMB, predictors can filter prediction from any entries that do not have perfect confidence.

However, both of these approaches lead to sub-optimal performance and missed opportunities. We instead propose, upon a false dependency, to allocate a new entry within the predictor with a bigger context length that encodes a non-dependency.

Revisiting our example, and illustrated in fig. 3, the initial state is shown at point ①, where only the load's PC is used to access the first table that stores the dependent prior store. When the predictor falsely predicts a dependence, we will allocate a new entry in the next table, hashing the load PC and the branch history, which includes the not-taken bit from the most recent branch, to create the index, shown at ②. Now, when the branch is taken, the predictor will hit in the first table, correctly identifying a dependence with the prior store, but when the branch is not taken the predictor will hit in the next, correctly identifying no dependence. Later, if there are situations where this non-dependency entry also causes mispredictions, then a further entry can be created in a higher-context table ③.

B. Discussion

Prior MDP and SMB predictors have opted to only track dependencies between loads and stores. We liken this to branch target prediction, where predictors only track branch targets when branches are taken. However, because of this, branch target prediction is only effective in the presence of a branch direction predictor that tracks the patterns of branches both when they're taken and not taken, and uses this to predict branch directions. Similarly, a memory-dependence predictor that only tracks the patterns of load-store dependencies (and not non-dependencies) will struggle to accurately predict whether there exists a dependency or not.

[1]This value was obtained by creating Markov chains and calculating the predictions required for the counter to reach 0.

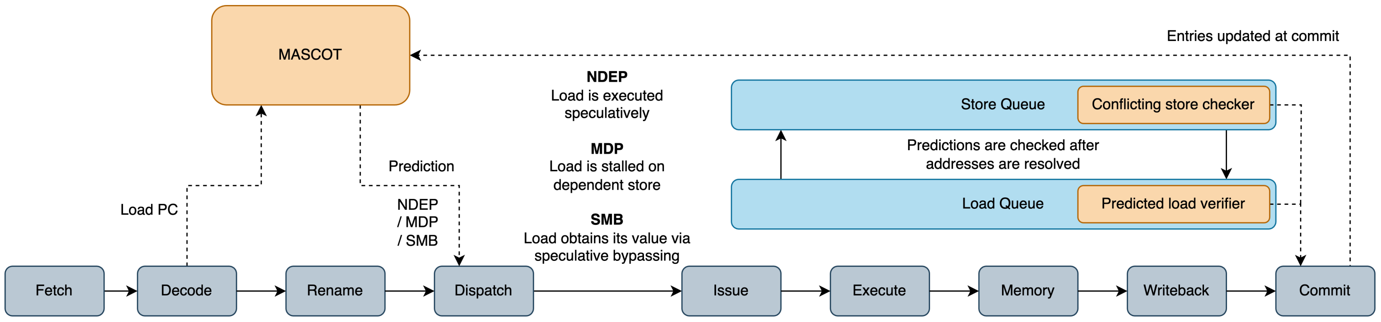

MASCOT
Entries updated at commit
NDEP
Load is executed speculatively
MDP
Load is stalled on dependent store
SMB
Load obtains its value via speculative bypassing
Store Queue
Conflicting store checker
Predictions are checked after addresses are resolved
Load Queue
Predicted load verifier
Load PC
Prediction
NDEP / MDP / SMB
Fetch
Decode
Rename
Dispatch
Issue
Execute
Memory
Writeback
Commit

Fig. 4. Pipeline with additions for MASCOT showing where predictions are made and used.

One solution would be to design a TAGE-like predictor that tracks dependencies as well as non-dependencies for predicting whether or not a load is dependent in conjunction with a traditional memory-dependence predictor for predicting the distance. However, if the memory-dependence predictor itself is based on TAGE, this is unnecessary, as the two predictors can be merged if we allow for entries to also predict non-dependencies.

As an example, the reason that ITTAGE and TAGE are kept separate in branch prediction is that TAGE entries are much smaller, and can be used to predict direct branches without consulting ITTAGE. In the analogy, all loads are indirect branches.

Allocating non-dependencies has the added benefit of more aggressively growing the dependent patterns as well. Where previous TAGE-style memory-dependence predictors are likely to keep an entry that even has a slightly higher accuracy than 50 % of being correct and otherwise non-dependent, in this scenario our predictor will continually allocate new entries. We argue that this is critical: traditional TAGE likely owes much of its performance to always allocating a new entry after a misprediction.

IV. MASCOT

We propose MASCOT, a predictor inspired by TAGE to tackle memory-dependence prediction as well as speculative memory bypassing. MASCOT is capable of tracking dependencies as well as non-dependencies, yielding high accuracy and low rates of both false positives and false negatives. We give an overview of MASCOT in section IV-A and describe its structure in section IV-B. Section IV-C then describe how MASCOT allocates entries and section IV-D provides details of how it tracks non-dependencies.

A. *Overview*

Fig. 4 shows a diagram of how MASCOT is incorporated into an out-of-order superscalar pipeline. When an instruction is decoded its instruction address is sent to MASCOT to make a prediction, which becomes available later once the instruction has been through the processor's frontend. MASCOT makes a three-way prediction: (1) no dependence on any prior store; (2) a dependence on a specified prior store, but no opportunity for speculative memory bypassing (MDP); (3) a dependence on a specified prior store and the value can be bypassed (SMB). These correspond to the left-hand side of fig. 5.

In case (1), the load can issue as soon as its address is known. In case (2), the load must wait for the specified prior store to obtain its address. At this point, the load issues, computes its address, and the dependence is checked. Finally, in case (3), the load can obtain its value through memory bypassing as soon as it has been calculated. In all cases, dependencies between load instructions and prior stores are identified as usual through snooping of the load and store queues. At commit, this dependence information is sent back to MASCOT to update its entries.

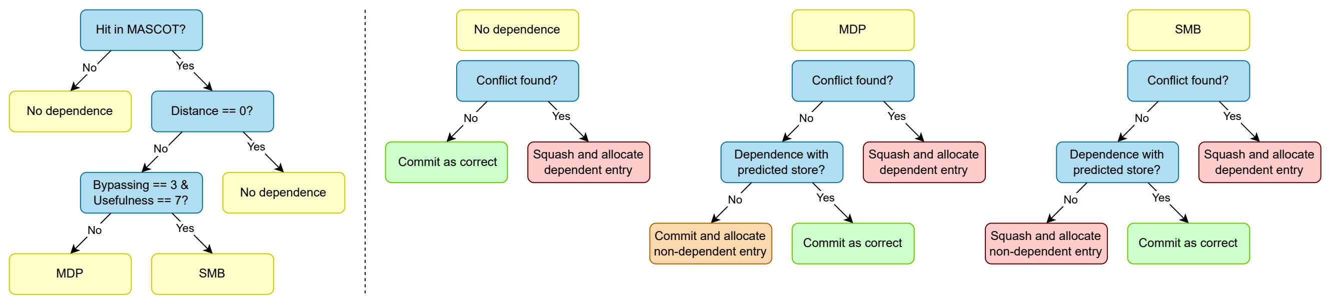
B. *Predictor structure*

Similar to TAGE, MASCOT has an array of tables that use increasing sizes of global history. Each table is 4-way associative, to tolerate some conflicts between entries with the same index. For each table, the index and tag are computed by folding the load PC and increasing lengths of the global branch and path history, as shown in fig. 3. The default configuration of MASCOT uses histories of [0, 2, 4, 8, 16, 32, 64, 128] branches respectively with 512 entries in each table. We use one bit to encode a taken or not taken branch, and for indirect branches we fold the target to 5 bits.

MASCOT makes a prediction for each load in the decode stage. The tables are searched in parallel, and the entry from the highest-context table that has a match is used for prediction. When the predictor finds no matching entry it will default to predicting a non-dependency (termed the base predictor).

Each entry (shown in fig. 6) has a 7-bit distance field, a tag field (default 16 bits) and two separate saturating counters: a 3-bit usefulness counter that indicates the confidence of the entry in predicting a memory dependence with the given distance, and a 2-bit bypass counter that indicates the confidence of the entry in predicting speculative memory bypassing. The sizes of counters and the global history lengths were selected via a grid-based sensitivity study. This means that the total predictor size (discarding logic) comes to 14 KiB.

A distance field of all 0s indicates that the entry is non-dependent (i.e., there is no dependence on any prior store), whereas any other value encodes an offset into the store queue. A value of 1, for example, means that the load depends on the



Hit in MASCOT?
No
Yes
No dependence
Distance == 0?
No
Yes
Bypassing == 3 & Usefulness == 7?
No dependence
No
Yes
MDP
SMB
No dependence
Conflict found?
No
Yes
Commit as correct
Squash and allocate dependent entry
MDP
Conflict found?
No
Yes
Dependence with predicted store?
Squash and allocate dependent entry
No
Yes
Commit and allocate non-dependent entry
Commit as correct
SMB
Conflict found?
No
Yes
Dependence with predicted store?
Squash and allocate dependent entry
No
Yes
Squash and allocate non-dependent entry
Commit as correct

Fig. 5. Decisions tree of predictions made by MASCOT (left-hand side) and the corresponding actions for each prediction once dependencies have been calculated (right-hand side). MASCOT predicts either no dependence, a dependence with a prior store (MDP) or a dependence with speculative memory bypassing (SMB). If a conflict is later found (where a conflict is defined as a dependence with a different store than the one predicted) then the load and its dependents must be squashed. For SMB, squashing must also occur if there was no dependence (since the load erroneously obtained its value through memory bypassing) and MASCOT allocates a non-dependency entry to avoid the false dependence the next time the load executes with the same context. For MDP, in the same situation there is no need to squash but an opportunity for performance improvement was lost.

Tag	Distance	Usefulness	Bypassing
16 bits	7 bits	3 bits	2 bits

Fig. 6. MASCOT entry structure consisting of tag and distance fields and two counters (28 bits total). Distance corresponds to the prior store that a load will depend on, or 0 to encode a non-dependency. Usefulness captures the confidence for MDP and the bypassing counter is confidence for SMB.

first store prior to the load in program order. Whenever the distance field is not zero, a memory dependence prediction is made regardless of the value of the usefulness field, whereas speculative memory bypassing is only predicted if both the usefulness and bypassing counters are saturated. Only entries with a usefulness counter at 0 may be evicted.

Entries are updated when a load is committed. When an entry correctly predicts a memory dependence the usefulness counter is incremented. Similarly, when there is a bypass opportunity correctly predicted, the bypass counter is incremented. When the entry predicts an incorrect memory dependence, the usefulness counter is decremented, and when the entry incorrectly predicts a bypassing opportunity, the bypass counter is reset to 0. This means that MASCOT is aggressive in predicting memory dependencies and conservative in taking advantage of SMB. Both are important to reduce the chances of mispredictions and their associated overheads from squashing.

C. Allocation

Like TAGE, MASCOT allocates entries on mispredictions, using lower-context tables first. Upon a misprediction from the base predictor (that always predicts a non-dependency), a dependent entry is allocated in the smallest-history table, which we can call N_0, with the distance set depending on the conflicting store. Dependent entries are always allocated with a usefulness of 6, and non-dependent entries (described more in section IV-D) are allocated with a usefulness of 2.

There are three types of misprediction that will lead to an allocation in a table with a longer history. First, when a load is predicted not to depend on any prior store but does have a dependence on one. Second, when a load is predicted to depend on a particular prior store but actually conflicts with a different one (where either the predicted store has the wrong address, or the conflict is with a younger store). Finally, when a load is predicted to depend on a prior store but no conflicts are detected. In the first two cases, the new entry contains the correct distance for the conflicting store. In the latter case, a non-dependency entry is allocated. These different scenarios can be seen in the right-hand side of fig. 5.

MASCOT's allocation strategy is tuned to be aggressive in its use of table entries. If a misprediction is made based on an entry found in table N_{i-1}, MASCOT will attempt to allocate to the next table ordered by length of global history, N_i. When allocating to a particular table, N_i, if the allocation fails (i.e., all entries in the relevant set have a usefulness greater than 0), then the predictor will attempt allocation in table N_{i+1}. Should allocation in table N_{i+1} fail, it will try in N_{i+2}, and so on. Given that allocation to N_{i+1} failed, regardless of whether an allocation was made to a bigger table or not, all four entries in table N_i have their usefulness counters decremented, but those in subsequent tables do not. This try-again style allocation policy means that MASCOT can make good use of its limited space, and the frequent usefulness decrements mean that stale entries do not live long.

We did not find any meaningful changes in performance from periodically decrementing all usefulness counters, despite this being a common optimisation across TAGE-like predictors. We suspect that this is a combination of the tables being 4-way associative as well as the above allocation policy. Every time an entry fails to allocate, the confidence of four entries is decremented, which likely helps to clear out the predictor.

D. Tracking non-dependencies

The power of MASCOT compared to other TAGE-like predictors lies in its ability to record instances of what we call conditional non-dependencies. While there are an effectively infinite number of non-dependencies to track (i.e., between every load and store where there is not a dependence), we only track non-dependencies for PCs where we would have otherwise predicted a dependence. Compared to prior memory-dependence predictors that generally rely on 'forgetting' predictions rather than learning these edge-cases, MASCOT can quickly adapt to program behaviour to learn false dependencies, rather than waiting for a counter to decrement sufficiently.

E. Speculative memory bypassing

SMB has been thoroughly explored in the literature [16], [18], [26], [30] with a variety of schemes to achieve bypassing. MASCOT operates independently of the specific SMB techniques employed, and can be implemented in any architecture that supports SMB in some form.

Given that MASCOT conservatively handles SMB, the bypassing counter is initially set to 1 when a new conflict is allocated, provided it is a potential bypassing scenario; otherwise, the counter is set to 0. In both cases, no bypassing is predicted until the counter saturates. If subsequent instances of this prediction continue to indicate bypassing opportunities, the counter increments by one, or resets to zero if not.

We discuss our implementation of SMB further in the section V, but MASCOT is designed for any microarchitecture that permits bypassing between stores and loads with no address offset (i.e., the addresses are the same), although the load size can be smaller than the store. As fig. 2 shows, these represent the vast majority of bypassing opportunities seen in our workloads. Nevertheless, MASCOT can be easily extended to support bypassing with offsets by incorporating a shifting field.

F. Tuning the predictor

Within MASCOT there are multiple parameters to tune to increase the performance or decrease the area of the predictor. Despite TAGE predictors being ubiquitous, there exists little literature for fine-tuning them. In section VI-D we focus on tuning the length of the tags and the sizes of the tables within MASCOT using the following method. We begin with a candidate predictor with the default table sizes (described in section IV-B) and run it on a set of benchmarks. For each entry in each table, we periodically compute its F1 score[2] and then sort all entries in each table by their F1 score. After this, the values are recorded and the F1 scores are reset. The recording from each period is averaged together to produce the final result.

The length of this period can be tuned: short intervals mean that only a few entries will have non-zero scores, while a long interval will mean that many entries per table might appear useful while, in reality, they were used in different time slices and could be allocated on top of one another.

Averaging across all benchmarks, we identify tables that should be larger (all entries have high F1 scores) and those that can be smaller (some entries have low F1 scores or are not useful at all). We show the results of this tuning on performance and area in section VI-D.

[2]The F1 score is defined as the harmonic mean of precision and recall. This measure balances entries with high accuracy but are seldom used and those with lower accuracy that are often used. One could opt for a weighted F-score if fine tuning for a specific architecture, where the weights can symbolise the respective gains/penalties of a given misprediction.

TABLE I
SYSTEM CONFIGURATION

4-core Golden Cove Processor [7]	
Front-end width	6-wide fetch and decode
Branch predictor	TAGE-SC-L [24]
Back-end width	12 execution ports and 8 commit width
ROB/IQ/LQ/SB	512/204/192/114 entries
Memory hierarchy	
L1I (private)	32KB, 8 ways [29], 4-cycle hit latency, pipelined, 64 MSHRs
L1D (private)	48KB 12 ways [29], 5-cycle hit latency, pipelined, 64 MSHRs
L1D prefetcher	IP-stride with a prefetch degree of 3
L2 (private)	1.25MB, 10 ways [29], 14-cycle hit latency, 64 MSHRs
L3 (shared)	3MB/bank (4 banks), 12 ways [29], 36-cycle hit latency, 64 MSHRs
Memory	4GB, 100-cycle access latency

This style of optimisation is well suited for a predictor like MASCOT, which uses a try-again style allocation policy. If one table is under-sized, it can allocate new entries to tables with longer histories. This means that we still get the benefits of shrinking the predictor when our table sizing works well for a given phase, and we pay less of a penalty in the worst case. For PHAST, because it is only able to allocate a given load-store pair to one specific table, this optimisation technique would likely lead to a bigger penalty.

V. METHODOLOGY

We evaluate MASCOT using a cycle-level in-house simulator that models in detail an out-of-order processor with an x86 instruction set architecture. The core is fed with an instruction flow (split into micro-operations at decode) generated by Sniper [4]. The memory hierarchy is modeled with Gems [13], using its embedded GARNET interconnect network model [1].

The simulated core resembles an Intel Golden Cove microarchitecture [21]. The main system parameters are summarised in table I. The pipeline has 3 ports for load execution and 2 ports for store execution. The 2-ported LQ and the 3-ported SB are searched associatively and in parallel with the L1D access, incurring the same latency as the L1D [8], but allowing 2 and 3 new searches, respectively, each cycle (pipelining). Stores are issued once both the address and the data registers are ready. Memory order violations are filtered when the load already has a forwarding store that is younger than the one triggering the violation. The simulator has been modified to support speculative memory bypassing.

We compare MASCOT against the state-of-the-art memory-dependence predictor, PHAST, with the configuration described by the authors [11], a combined MDP-SMB predictor based on NoSQ [26] and Store Sets [6]. The NoSQ predictor has a path-dependent table and a path-independent table. High-confidence predictions from the path-dependent table are allowed to perform SMB, while low-confidence predictions mark the load to wait only for the predicted store. Predictions from the path-independent table are never allow to perform SMB. If no prediction is found, the load is allowed to execute speculatively. Store Sets perform memory dependence

TABLE II
CONFIGURATION OF THE EVALUATED PREDICTORS

Predictor	Tables	Total entries	Fields per entry	Size (KB)
Store Sets	SSIT (direct mapped)	8K	1 valid bit 12 bit SSID	**18.5**
	LFST (direct mapped)	4K	1 valid bit 10 bit St ID	
NoSQ	2 (4 way)	4K	22 bit tag 7 bit counter 7 bit distance 2 bit lru	**19**
PHAST	8 (4 way)	4K	16 bit tag 4 bit counter 7 bit distance 2 bit lru	**14.5**
MASCOT	8 (4 way)	4K	16 bit tag 3 bit counter 7 bit distance 2 bit bypass	**14**

prediction by recording sets of dependent stores for each load, and enforcing serialisation within these sets.

MASCOT is simulated both as solely a memory-dependence predictor (the bypassing counter is ignored), and with memory-dependence prediction as well as speculative memory bypassing together. The main configuration of Store Sets, NoSQ, PHAST and MASCOT is shown in table II.

To simulate bypassing, instructions that depend on a bypassed load will not wait for its completion before issuing. Instead, they will issue as soon as the data source physical register of the store is available, from which they will obtain their values. Loads execute as soon as their (address) operands are ready. The value obtained by the load is then compared to the value of the store's physical register, and if they match, the speculative bypass is considered correct. If not, the load and all following instructions are squashed. An address check also occurs as soon as the addresses are available, to allow for earlier misprediction-detection if the addresses do not match.

The predictors are evaluated on the SPEC CPU 2017 rate suite [27]. Benchmarks were compiled with *-O3* optimisation, as well as *-mno-avx -fno-unsafe-math-optimisation -fno-tree-loop-vectorise*. Vectorisation has been disabled since our simulator does not model vector units. Nevertheless, MASCOT can be incorporated into a processor that has SIMD instructions by leveraging the memory conflict-detection mechanism, which should account for vector-vector, vector-scalar, and scalar-vector cases.

Benchmarks with multiple inputs are labeled with an increasing number as a suffix to their name. For each combination of application-input, we generated a set of SimPoint intervals of 100 M instructions following the guidelines of Gottschall et al. [9]. To investigate the impact that MDP and SMB might have in future architectures, we also evaluate MASCOT on a core resembling the Intel Lion Cove microarchitecture [5].

VI. RESULTS

We compare the per-benchmark performance of MASCOT to that of other existing MDP and SMB predictors, showing performance gains of 1.9 % over state-of-the-art. We then compare an MDP-only version of MASCOT to other existing MDP predictors, to show that even when only performing MDP, MASCOT is still state-of-the-art. Then, to demonstrate the performance and accuracy impact of recording non-dependencies, we compare MASCOT to a TAGE-like predictor that does not allocate non-dependencies. Finally, we show how we can optimise MASCOT for size to yield a more compact predictor that maintains most of MASCOT's performance while taking up only 10.1 KiB of space for its tables.

A. Performance

In Fig. 7, we compare the IPC of MASCOT to both NoSQ and the previous state-of-the-art memory-dependence predictor, PHAST. All results are normalised to a perfect MDP predictor that does not do bypassing. Fig. 8 compares the total number of mispredictions (false positives and false negatives) across all benchmarks for NoSQ, PHAST and MASCOT, as well as their distribution of false positives/negatives.

We observe that despite doing speculative memory bypassing, NoSQ generally performs worse than perfect MDP, which does no bypassing at all, likely owing to its relatively simple GShare-based predictor. PHAST, the current state-of-the-art memory-dependence predictor, generally falls within 93–99 % of perfect MDP, while MASCOT doing MDP as well as SMB overtakes it with a 1.9 % higher geometric mean.

As is evident from the figure, the benefits of bypassing with MASCOT are highly application specific. Some benchmarks, such as *perlbench2*, see more than a 17.8 % increase in IPC compared to a perfect MDP and 26.4 % compared to PHAST while others such as *exchange2* see barely any impact. Overall, comparing the geometric means, we observe that for IPC, MASCOT yields increases of 4.9 %, 1.9 % and 1.0 % over NoSQ, PHAST, and a theoretically perfect MDP respectively. Compared to a perfect MDP and SMB predictor (not shown), MASCOT is 1.0 % slower.

To investigate the performance increase in *perlbench2* further, we measured, for instructions that depend on one load or more, the average number of cycles spent in the issue stage waiting for dependencies to resolve. Enabling bypassing decreased this number from an average of 38.7 cycles down to 15.7, or a 60 % reduction. For *lbm*, another benchmark which similarly to *perlbench* has a high rate of bypass prediction, enabling bypassing only reduces this number by 1.9 %. This indicates that *perlbench* is particularly sensitive to load-values being available early.

In fig. 8 we observe that MASCOT reduces the total errors by 98 % and 85 % compared to NoSQ and PHAST respectively. Comparing the distribution of error types, MASCOT reduces speculative errors by 39 % and false dependencies by 91 % compared to PHAST. This shows the importance of allocating non-dependency entries to reduce the high rate of false dependencies.

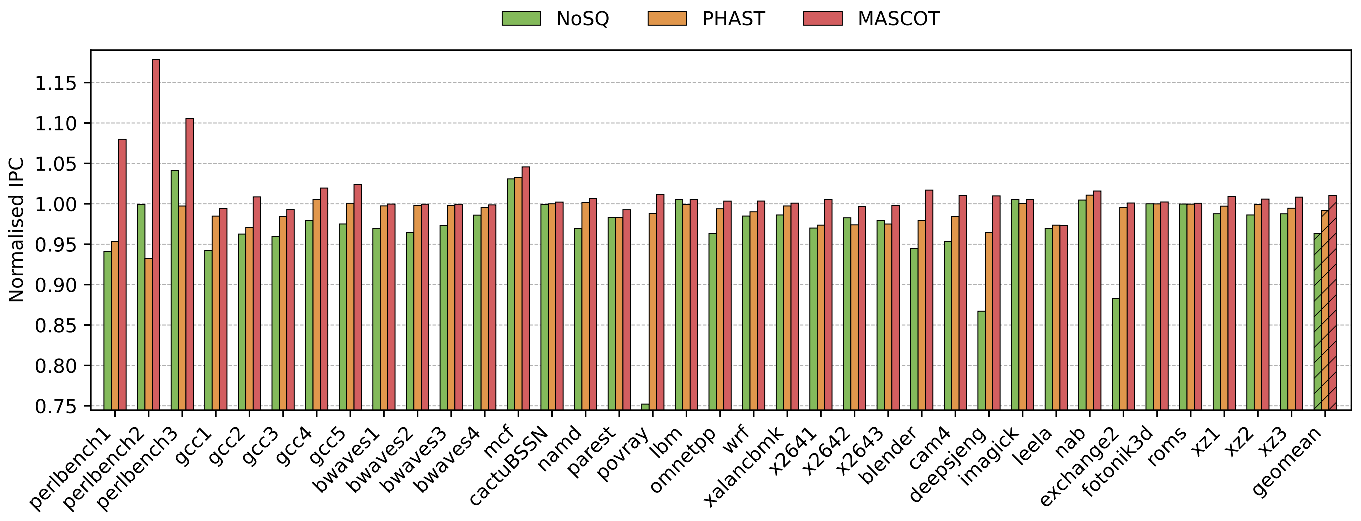


Fig. 7. IPC normalised to a perfect memory-dependence predictor (no SMB). We show NoSQ (a state-of-the-art SMB predictor) and PHAST (a state-of-the-art MDP predictor) alongside the full MASCOT implementation (MDP and SMB). MASCOT out-performs NoSQ by 4.9 %, PHAST by 1.9 % and perfect MDP by 1.0 %.

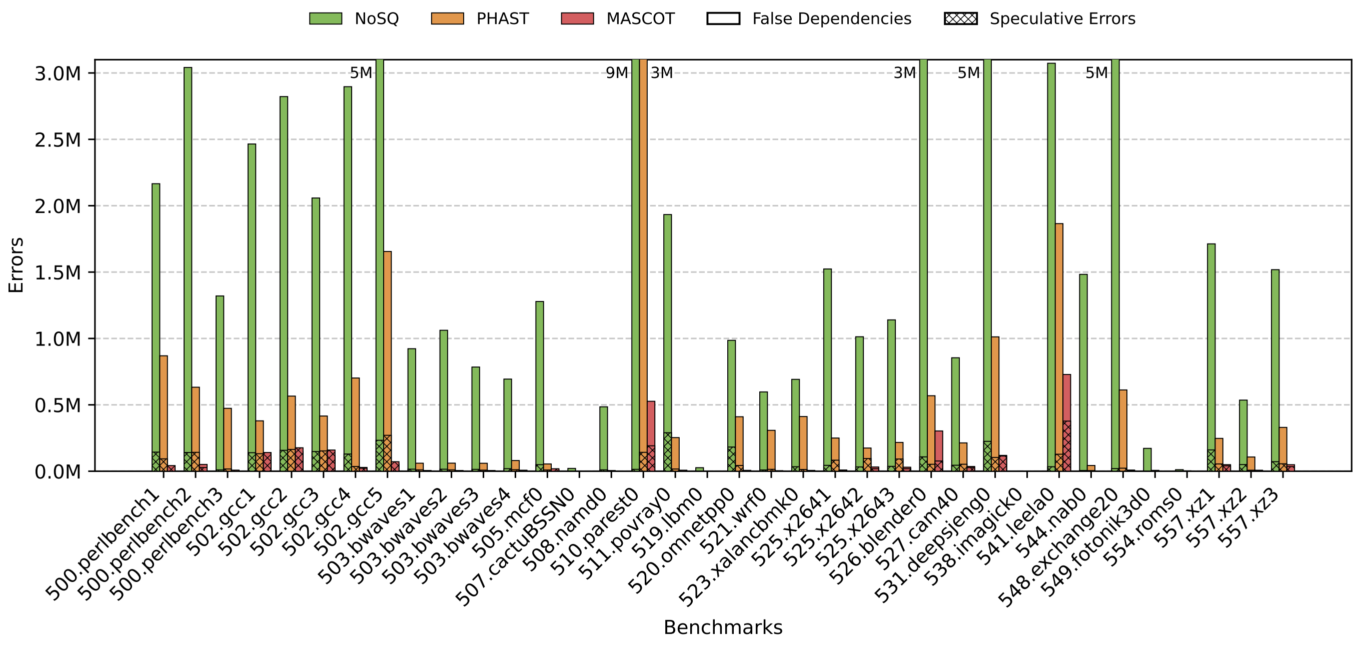


Fig. 8. Number of total mispredictions for each type of predictor, and the distribution of false dependencies and speculative errors.

In fig. 9, we compare the performance of Store Sets, PHAST and an MDP-only version of MASCOT, normalised to a perfect MDP predictor, to illustrate that even for memory-dependence prediction alone, MASCOT is state of the art. The MDP-only version of MASCOT achieves IPC improvements of 6.2 % over Store Sets and 0.4 % over PHAST. The performance of Store Sets is likely explained by the large size of the modelled CPU. As the ROB gets bigger, so does the number of possible conflicting load-store pairs. Store Sets, not using context-dependent prediction, is not well equipped to deal with this.

Note than in some benchmarks (namely *gcc4*, *gcc5*, *mcf* and *nab*), the MDP-only predictors perform better than perfect MDP. This is because there are some instances where an incorrect non-dependency prediction might lead to better performance. For example, if the load is issued at the same time as a dependent store, due to program characteristics, the store might always resolve before the load reaches the memory stage and then simply forward its data to the load. The perfect MDP predictor is inherently conservative, and in these instances, will stall the load by at least 1 cycle to be

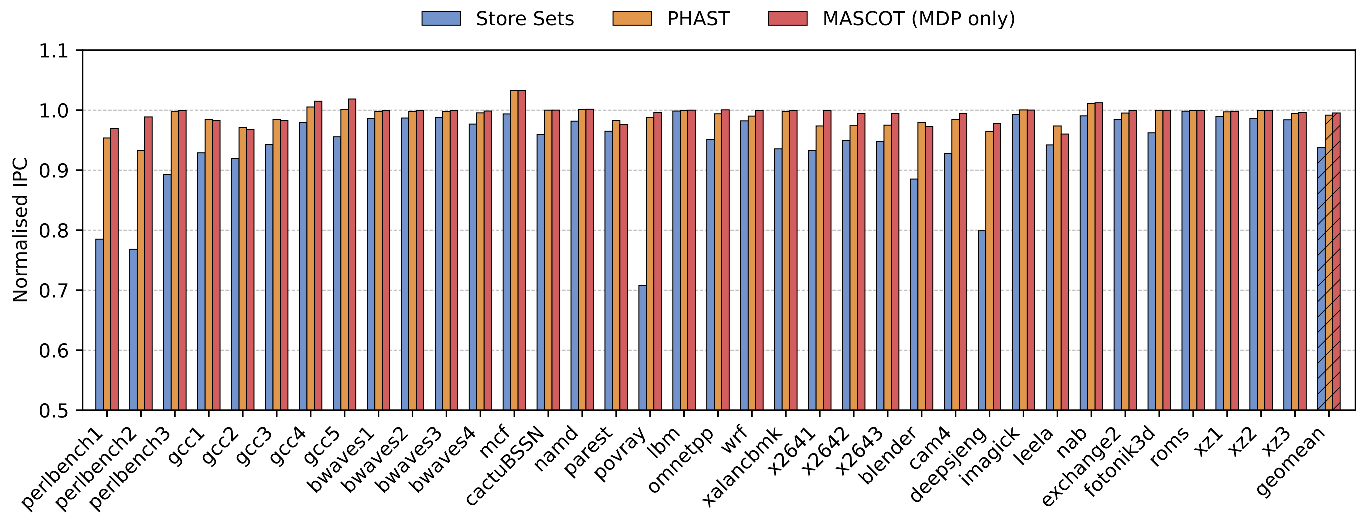


Fig. 9. IPC for MDP techniques only, normalised to a perfect memory-dependence predictor. We show Store Sets and PHAST alongside the MDP-only version of MASCOT. MASCOT out-performs Store Sets by 6.18 % and PHAST by 0.36 %.

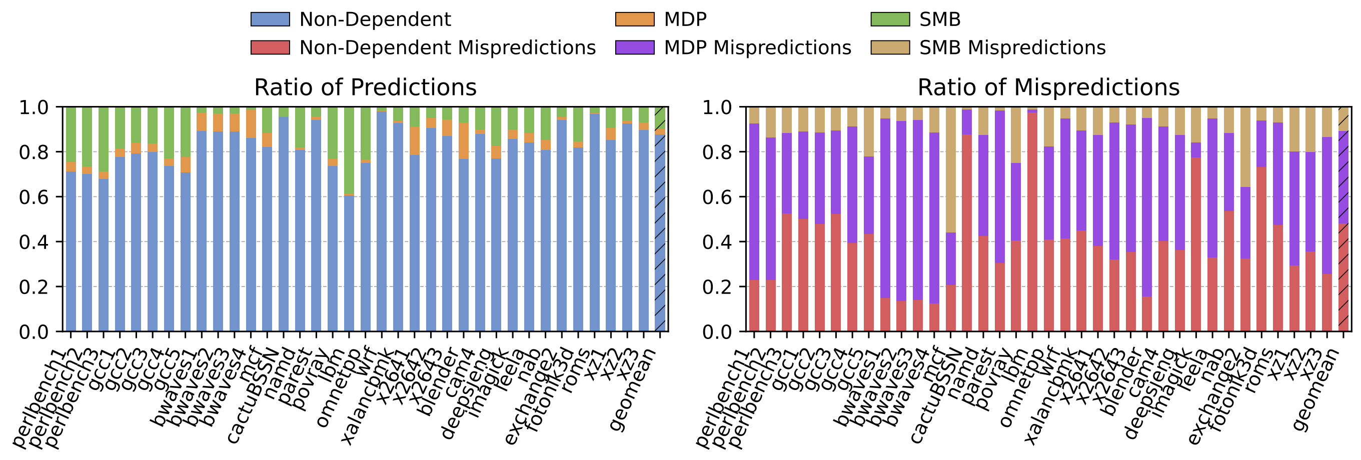


Fig. 10. Distribution of prediction and misprediction types for each benchmark with MASCOT.

certain that no conflict arises. Thus the perfect MDP predictor makes optimal predictions but this can lead to sub-optimal performance.

Fig. 10 shows the distribution of prediction and misprediction types for each benchmark. In the left-hand graph, over 80 % of all predictions are of no dependency, with the majority of the rest being for SMB. Considering the right-hand graph, we observe that the proportion of SMB mispredictions, which always cause a squash, is low. This is likely due to using high confidence requirements before making this type of prediction. *mcf* has a significantly higher proportion of SMB mispredictions compared to the rest of the benchmarks. Nonetheless, we observe that MASCOT still outperforms its MDP-only version because the total mispredictions for that benchmark are low, as shown in fig. 8.

B. Analysis

We argued earlier that much of MASCOT's increased accuracy comes from learning the context of non-dependencies. To test this claim, we constructed a similar TAGE-like predictor that is structurally similar to MASCOT but does not allocate non-dependent entries. Instead, on a false dependency, it will simply decrement the confidence of the predicting entry, similar to previous MDP and SMB implementation using TAGE.

Fig. 11 shows that the TAGE-like predictor without non-dependent entries functions worse as a memory-dependence predictor, and much worse as a memory-dependence and speculative memory bypassing predictor. The difference becomes especially clear in the number of false dependencies, where the TAGE-like predictor has more than 12× as many false dependencies than MASCOT. The reason for this is that when the predictor cannot allocate non-dependent entries, these will

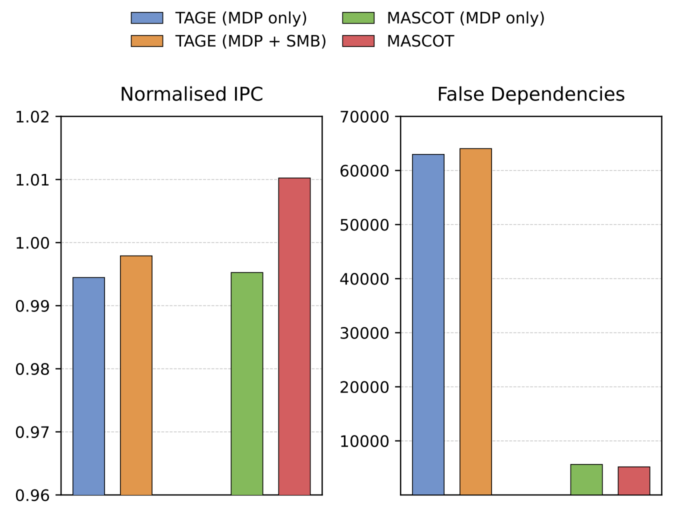


Fig. 11. Comparison of MASCOT to an equivalent TAGE-based predictor that does not allocate non-dependencies. IPC values are normalised to a perfect MDP. Allocating space for non-dependencies within the predictor's tables enables a significant reduction in false dependencies, yielding MASCOT's IPC gains.

instead reduce the confidence of dependent entries, which then become less likely to predict bypassing.

C. Future architectures

To assess the impact of MASCOT in future microarchitectures with larger core structures, we simulate on our existing configuration for Golden Cove, as well as one aimed at the more recent Lion Cove. We observe that the potential gains of SMB are raised. In Golden Cove, a perfect MDP+SMB predictor outperforms a perfect MDP-only predictor by 2.1 %, whereas for Lion Cove this ceiling is raised to 2.8 % as shown in fig. 12. Compared to a perfect MDP-only predictor, MASCOT achieves gains of 1.0 % and 1.3 % respectively, illustrating how MASCOT can yield additional gains for wider architectures.

D. Fine tuning

Section IV-F described a technique for tuning MASCOT so as to optimise its area whilst maintaining high performance. MASCOT's default configuration uses 14 KiB of space for its entries, with each table using 512 entries. We seek to optimise this by analysing entry usage. Following the methodology in section IV-F, we simulated MASCOT across all benchmarks, recording F1 scores every 1,000,000 cycles.

Fig. 13 shows the distribution of predictions made from each table in MASCOT, and fig. 14 shows the average ranking of entries by F1 score across all SPEC CPU 2017 benchmarks. We make two observations. First, table 1 could benefit from being larger, as its least important entry ranks similar to the top 40 entries in table 2. Second, to reduce space, tables 5–7 could be reduced to half their size, and table 8 could be reduced to one quarter.

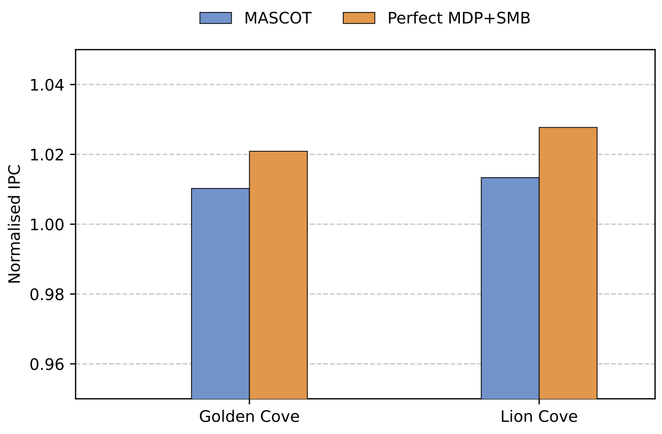


Fig. 12. IPC of MASCOT and a perfect MDP and SMB predictor in Golden Cove and Lion Cove. Both are normalised to a perfect MDP predictor for their respective architectures. The benefits of MASCOT and opportunities for performance gains increase as the core structures get larger.

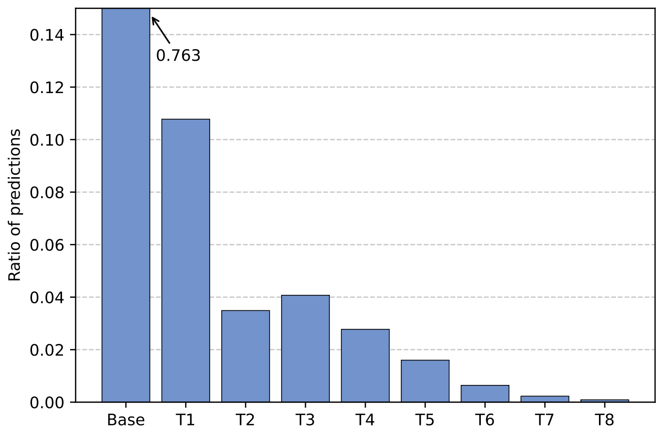


Fig. 13. Distribution of predictions made from each table. Base is the default non-dependence prediction given that no entry was found.

We modify the predictor to instead use table sizes of [1,024, 512, 512, 512, 256, 256, 256, 128], and tag sizes of [15, 16, 16, 16, 17, 17, 17, 18] respectively to keep the same likelihood of conflicts, obtaining a 16 % decrease in size. We term this new predictor MASCOT-OPT.

We simulate MASCOT-OPT, as well as additional configurations where the tag sizes are reduced by 2, 4 and 6 bits, to save further area. Fig. 15 shows the resulting changes in IPC. MASCOT-OPT leads to a loss of only 0.09 % in IPC. Reducing the tag size by 4 bits for MASCOT-OPT leads to an IPC decrease of just 0.13 % due to an increase of 17.4 % in mispredictions, yet saves 27.7 % area, requiring only 10.1 KiB.

VII. RELATED WORK

There are a number of works in the literature that study MDP and SMB using a variety of prediction structures. We consider here additional publications that were not described earlier in section II.

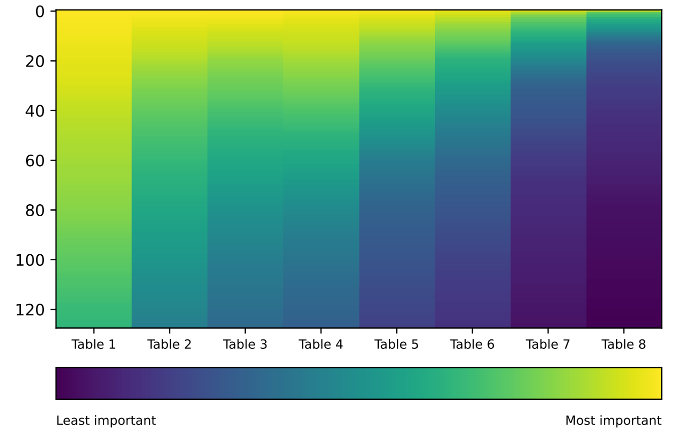


Fig. 14. F1 scores of entries ranked over each table. Table 1 contains the most important entries and could be larger. Conversely tables 5–8 could be reduced in size since their entries do not have high F1 scores.

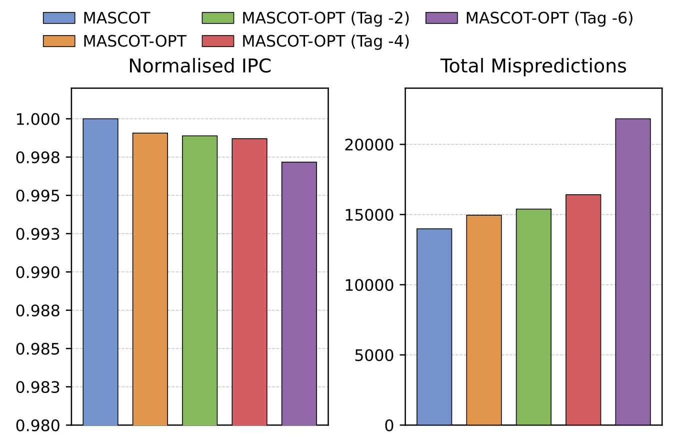


Fig. 15. After optimising table sizes and reducing tag sizes, MASCOT-OPT requires only 10.1 KiB for an IPC reduction of 0.13 %.

Önder and Jin [10] proposed a method to speculatively execute loads earlier in store-queue free architectures. Whereas NoSQ will always delay loads for whose dependencies it has poor confidence, DMDP proposes to predicate these loads. When a a load is hard to predict, DMDP will instead issue a micro-op to compare the addresses of the store and load and, based on this result, either obtain the value from the store or perform the load. This essentially achieves the benefits of non-speculative forwarding without the overhead of a load and store queue. The predictor used for DMDP is the same as NoSQ, and thus one can imagine that the gains would likely be greater with MASCOT.

Moshovos and Sohi showed how bypassing can also be applied to load-load pairs, rather than just load-store pairs [15]. Perais, Endo and Seznec proposed distance prediction as a generalization to SMB [17]. Distance prediction attempts to determine how many instructions separate a given instruction from the most recent instruction that provides the same result.

Alves et al. [2] proposed a unified store-queue/buffer/cache (S/QBC) to filter L1/TLB probes. With the help of a memory-dependence predictor based on store distance [31], they predict when the load will hit or miss in the S/QBC. Hits predicted correctly reduce energy consumption since the L1/TLB is not probed, while correctly predicted misses reduce latency by letting the load probe both the S/QBC and L1/TLB in parallel. The memory-dependence predictor used yields 93.6 % accuracy on average, with a worst case of 89.7 %, leaving potential for further improvement with MASCOT.

Bekerman et al. propose multiple different schemes for detecting load-store dependencies through register tracking [3]. One example of such a scheme is tracking stores to the stack pointer. The authors find, for example, that 25 % of all loads are stack loads and 95 % of them can be resolved in the front-end solely through register tracking.

VIII. CONCLUSION

In this work we have shown that current prediction mechanisms are tailored either for memory dependencies or for speculative memory bypassing. We have proposed MASCOT, a prediction mechanism based on TAGE that improves its allocation policy by learning non-dependencies. MASCOT's optimised allocation policy is able to considerably reduce the prediction of false dependencies while still keeping a low rate of missing dependencies, obtaining a reduction in mispredictions of 70 % on average with respect to the state-of-the-art predictor PHAST. Thanks to that characteristic, MASCOT obtains an IPC improvement of 0.4 % over PHAST. More importantly, because MASCOT has high accuracy both with regards to false dependencies and missing dependencies, it also functions as an accurate predictor for speculative memory bypassing. When applied to a processor supporting speculative memory bypassing, it is able to increase performance by 1.9 % on average compared to PHAST while using the same amount of space, or 1.8 % while being 30 % smaller in size at 10.1 KiB.

IX. ACKNOWLEDGEMENTS

This paper was made possible due to funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 819134), the Engineering and Physical Sciences Research Council (EPSRC) grant reference EP/W00576X/1 and from the MCIN/AEI/10.13039/501100011033/ and the "ERDF A way of making Europe", EU (grant PID2022-136315OB-I00). Karl Hvarregaard Mose is a PhD student and Trinity Hall Scholar funded by the Cambridge Commonwealth, European & International Trust. Sebastian S. Kim is a PhD student funded by the Fundación Séneca, Región of Murcia (21456/FPI/20). Additional data related to this publication is available in the repository at https://doi.org/10.17863/CAM.114977.

References

[1] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," in *ispass*, Apr. 2009, pp. 33–42.

[2] R. Alves, A. Ros, D. Black-Schaffer, and S. Kaxiras, "Filter caching for free: The untapped potential of the store buffer," in *2019 International Symposium on Computer Architecture (ISCA)*, Jun. 2019, pp. 436–448.

[3] M. Bekerman, A. Yoaz, F. Gabbay, S. Jourdan, M. Kalaev, and R. Ronen, "Early load address resolution via register tracking," *ACM SIGARCH Computer Architecture News*, vol. 28, no. 2, pp. 306–315, 2000.

[4] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulations," in *SC*, Nov 2011, pp. 52:1–52:12.

[5] Chips and Cheese. (2024, 6) Intel's lion cove architecture preview. [Online]. Available: https://chipsandcheese.com/2024/06/03/intels-lion-cove-architecture-preview/

[6] G. Z. Chrysos and J. S. Emer, "Memory dependence prediction using store sets," *ACM SIGARCH Computer Architecture News*, vol. 26, no. 3, pp. 142–153, 1998.

[7] clamchowder, "Popping the hood on golden cove," https://chipsandcheese.com/2021/12/02/popping-the-hood-on-golden-cove, Dec. 2021.

[8] A. Fog, "The microarchitecture of Intel, AMD and VIA CPUs: An optimization guide for assembly programmers and compiler makers," https://www.agner.org/optimize/microarchitecture.pdf, Technical University of Denmark, Nov. 2022.

[9] B. Gottschall, S. H. C. de Santana, and M. Jahre, "Balancing accuracy and evaluation overhead in simulation point selection," in *IEEE International Symposium on Workload Characterization, IISWC 2023, Ghent, Belgium, October 1-3, 2023*. IEEE, 2023, pp. 43–53.

[10] Z. Jin and S. Önder, "Dynamic memory dependence predication," in *2018 International Symposium on Computer Architecture (ISCA)*, Jun. 2018, pp. 235–246.

[11] S. S. Kim and A. Ros, "Effective context-sensitive memory dependence prediction," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024, pp. 515–527.

[12] G. H. Loh, R. Sami, and D. H. Friendly, "Memory bypassing: Not worth the effort," in *Proc. 1st Workshop on Duplicating, Deconstructing, and Debunking*. Citeseer, 2002, pp. 71–80.

[13] M. M. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *can*, vol. 33, no. 4, pp. 92–99, Sep. 2005.

[14] S. McFarling, "Combining branch predictors," Digital Western Research Laboratory, Technical report TN-36, Jun. 1993.

[15] A. Moshovos and G. S. Sohi, "Read-after-read memory dependence prediction," in *MICRO-32. Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture*. IEEE, 1999, pp. 177–185.

[16] A. Moshovos and G. S. Sohi, "Speculative memory cloaking and bypassing," *International Journal of Parallel Programming*, vol. 27, no. 6, pp. 427–456, 1999.

[17] A. Perais, F. A. Endo, and A. Seznec, "Register sharing for equality prediction," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[18] A. Perais and A. Seznec, "Cost effective physical register sharing," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2016, pp. 694–706.

[19] A. Perais and A. Seznec, "Storage-free memory dependency prediction," *IEEE Computer Architecture Letters*, vol. 16, no. 2, pp. 149–152, Jul. 2017.

[20] A. Perais and A. Seznec, "Cost effective speculation with the omnipredictor," in *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, 2018, pp. 1–13.

[21] E. Rotem, A. Yoaz, L. Rappoport, S. J. Robinson, J. Y. Mandelblat, A. Gihon, E. Weissmann, R. Chabukswar, V. Basin, R. Fenger, M. Gupta, and A. Yasin, "Intel Alder Lake CPU architectures," *ie3m*, vol. 42, no. 3, pp. 13–19, Mar. 2022.

[22] A. Seznec, "The L-TAGE branch predictor," *The Journal of Instruction-Level Parallelism*, vol. 9, pp. 1–13, May 2007.

[23] A. Seznec, "A new case for the TAGE branch predictor," in *2011 International Symposium on Microarchitecture (MICRO)*, Dec. 2011, pp. 117–127.

[24] A. Seznec, "TAGE-SC-L branch predictors again," in *5th JILP Workshop on Computer Architecture Competitions (JWAC-5): Championship Branch Prediction (CBP-5)*, Jun. 2016.

[25] A. Seznec and P. Michaud, "A case for (partially) tagged geometric history length branch prediction," *The Journal of Instruction-Level Parallelism*, vol. 8, 2006.

[26] T. Sha, M. M. Martin, and A. Roth, "NoSQ: Store-load communication without a store queue," in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*. IEEE, 2006, pp. 285–296.

[27] Standard Performance Evaluation Corporation, "SPEC CPU2017," 2017. [Online]. Available: http://www.spec.org/cpu2017

[28] S. Subramaniam and G. H. Loh, "Store vectors for scalable memory dependence prediction and scheduling," in *2006 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2006, pp. 65–76.

[29] A. Syed, "Intel 12th gen alder lake golden cove-gracemont cache configuration detailed," Jul. 2021, https://www.hardwaretimes.com/intel-12th-gen-alder-lake-golden-cove-gracemont-cache-configuration-detailed/.

[30] G. S. Tyson and T. M. Austin, "Improving the accuracy and performance of memory communication through renaming," in *Proceedings of 30th Annual International Symposium on Microarchitecture*. IEEE, 1997, pp. 218–227.

[31] A. Yoaz, M. Erez, R. Ronen, and S. Jourdan, "Speculation techniques for improving load related instruction scheduling," in *1999 International Symposium on Computer Architecture (ISCA)*, May 1999, pp. 42–53.