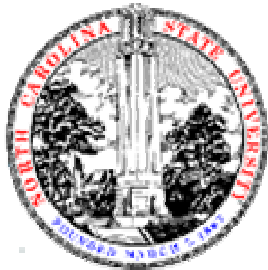


Enhancing Memory Level Parallelism via Recovery-Free Value Prediction

Huiyang Zhou, Thomas M. Conte



TINKER Research Group
Department of Electrical & Computer Engineering
North Carolina State University

Motivation

- **Memory wall problem**
 - Contemporary processor design makes it worse
 - Hide long memory latencies (cache misses)
 - Overlap multiple cache misses (MLP)

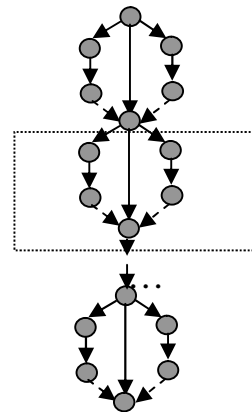
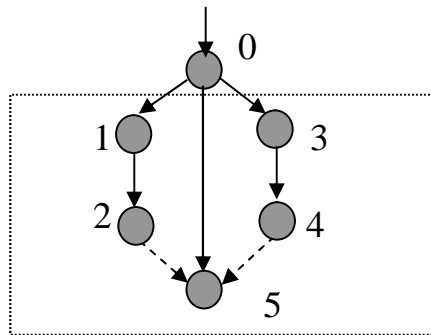
Memory Dependence Chain

A code example from the benchmark *mcf*

```

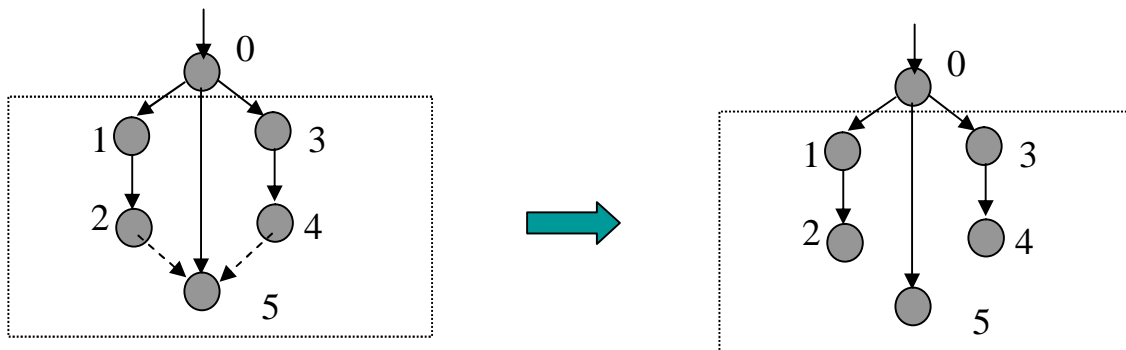
while( node )
{
  if( node->orientation == UP )
    node->potential = node->basic_arc->cost + node->pred->potential;
  else
    {
      node->potential = node->pred->potential - node->basic_arc->cost;
      checksum++;
    }
  tmp = node;
  node = node->child;
}

```



Breaking Memory Dependencies

- Alias dependence: aggressive memory disambiguation.



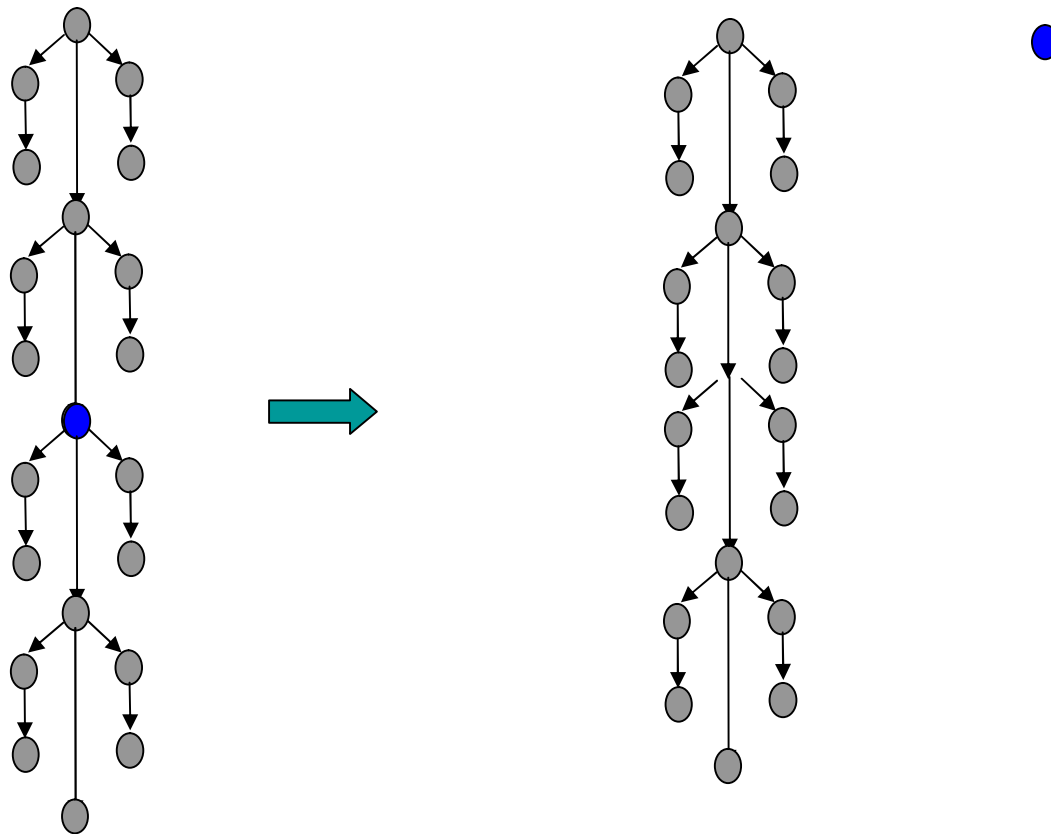
```

while( node )
{
  if( node->orientation == UP )
    node->potential = node->basic_arc->cost + node->pred->potential;
  else /* == DOWN */
    ...
  node = node->child;
}

```

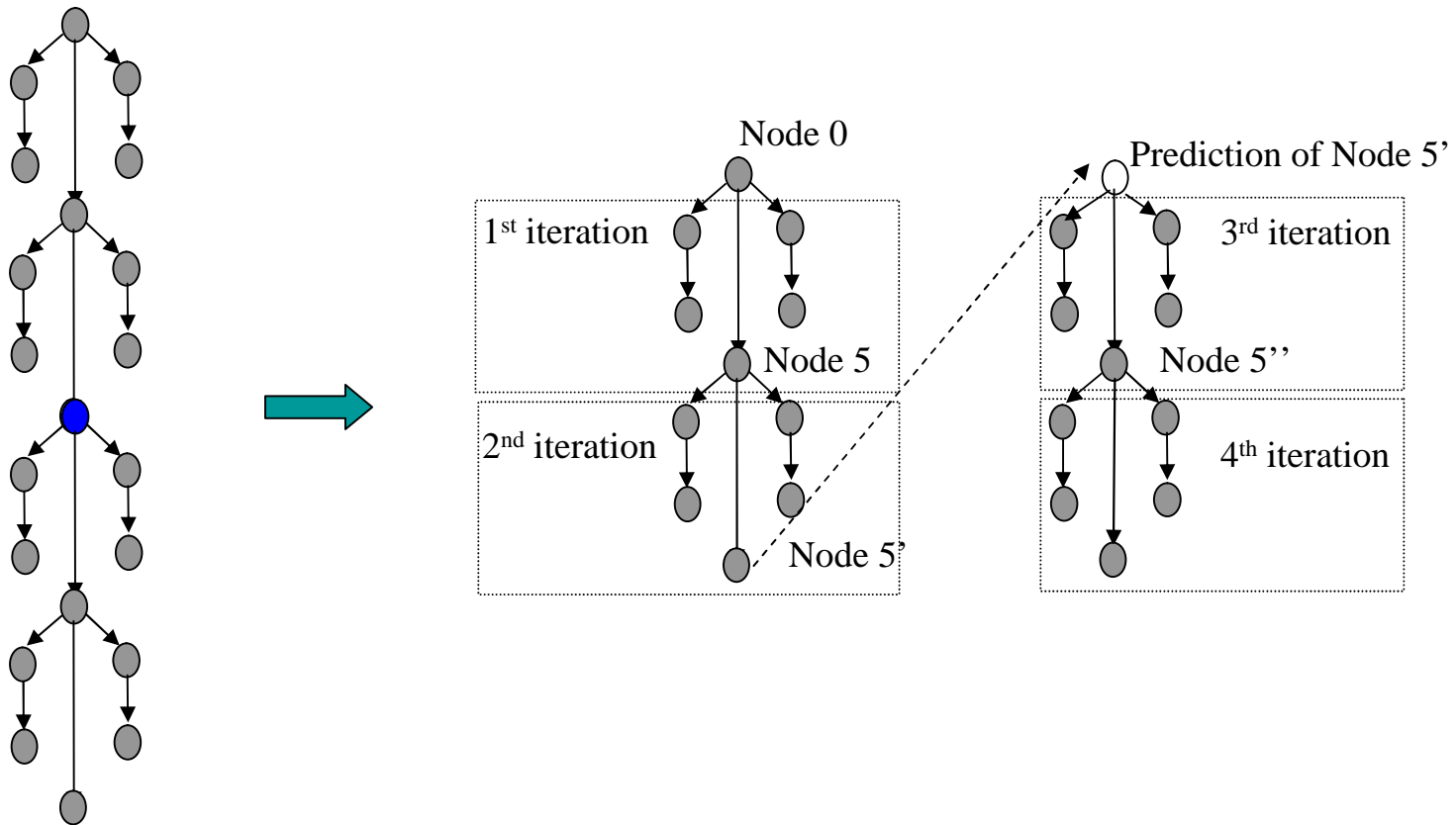
Breaking Memory Dependencies

- True data dependence: prefetching based on load address prediction



Breaking Memory Dependencies

- True data dependence: value prediction.



Performance Modeling of Value Prediction and Memory Prefetching

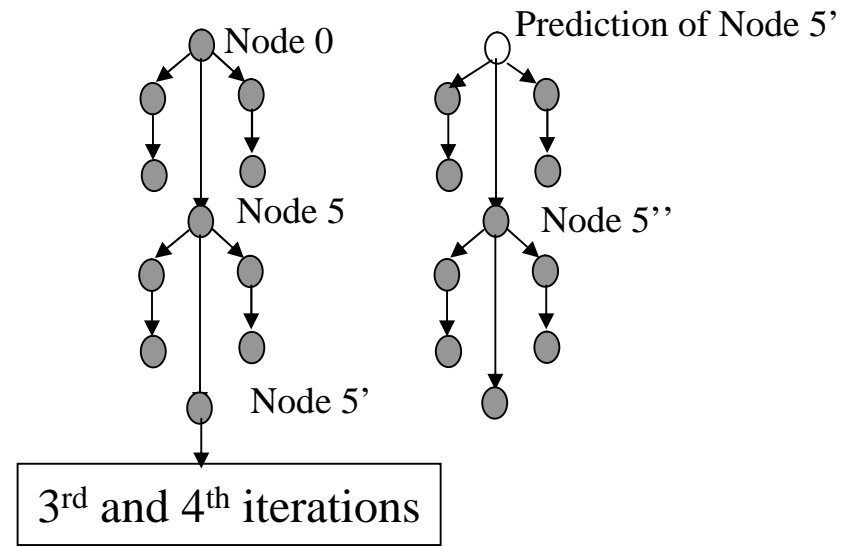
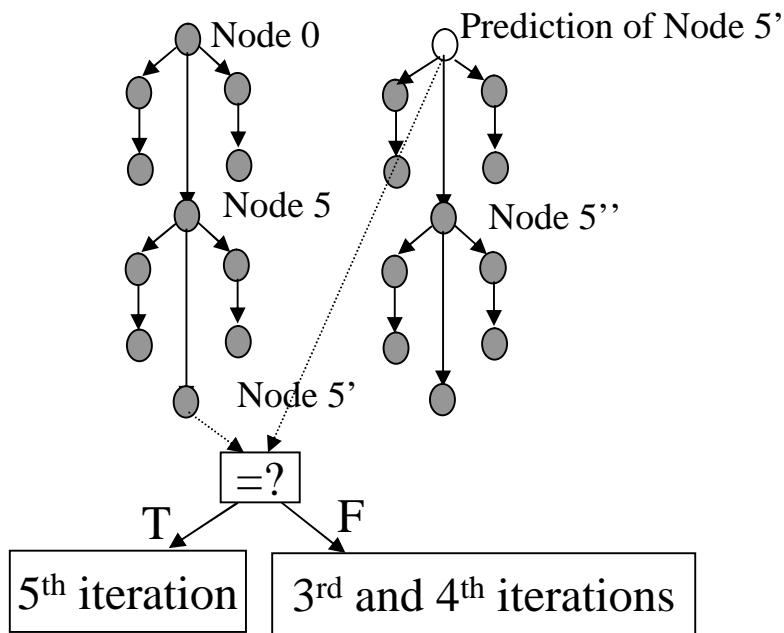
- Value prediction wins for long memory dependence chains
 - Key: use the fetched data to drive more dependent loads to be executed speculatively.
- Prefetch works better for short chains
- Current microprocessor design trend (large instruction window)
=> long memory dependence chain for pointer chasing workloads.

Outline

- Introduction
- **Recovery-Free Value Prediction**
- **Implementation of Recovery-Free Value Prediction**
- **Experimental Results**
- **Limitations**
- **Summary**

Enhancing MLP via Recovery-Free Value Prediction

- Value prediction can break true data dependence but involves complex validation and recovery mechanism.
- We propose: use value prediction and the speculative execution for *prefetch*.
- Scenarios.



Recovery-Free Value Prediction vs. Value Prediction

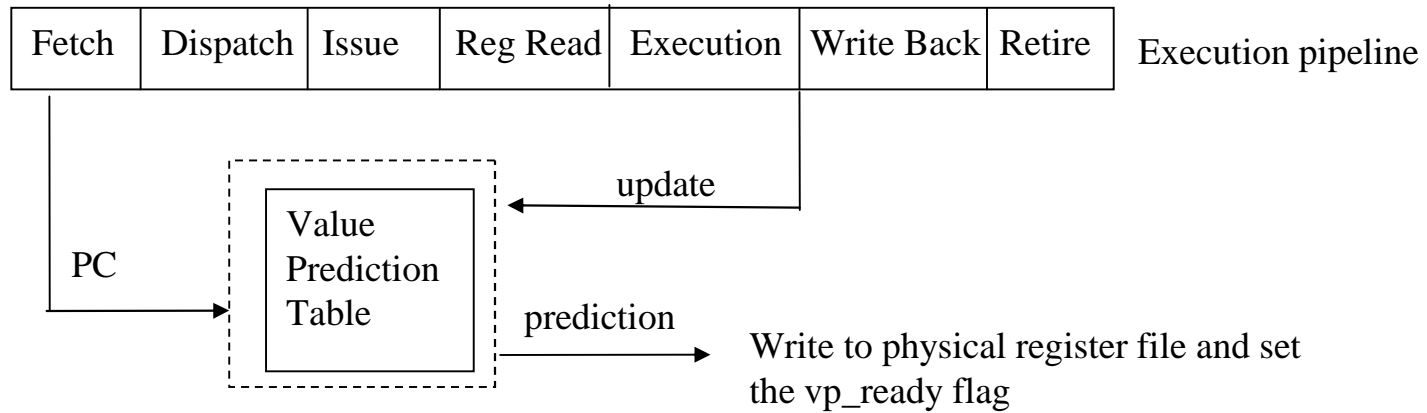
- **Complexity comparison:**
 - Much simpler hardware (low complexity)
- **Performance Comparison**
 - **Correct prediction: a small re-execution overhead**
 - **Misprediction: no validation and recovery latencies**
 - **Same hardware also enables aggressive memory disambiguation**
=> better performance

Outline

- Introduction
- Recovery-Free Value Prediction
- **Implementation of Recovery-Free Value Prediction**
- Experimental Results
- Limitations
- Summary

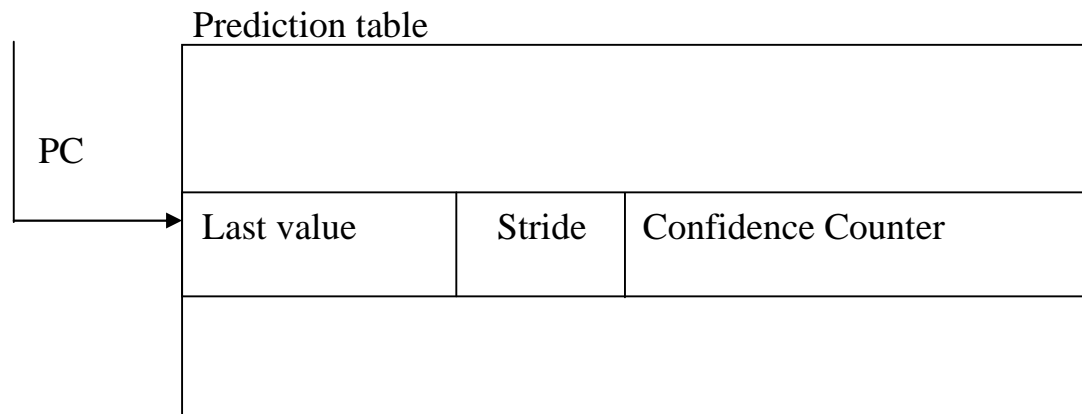
Recovery-Free Value Prediction

- Proposed design (based on a MIPS R10000 superscalar processor model)



Hardware Design of Recovery-Free Value Prediction

- **Value predictor**
 - A simple stride predictor [Lipasti, Wilkerson, and Shen]



- A speculative update scheme similar to [Lee, Yew] is also used to improve performance
- More powerful predictors: context predictor [Wang and Franklin], differential FCM [Goeman, etc.], gDiff [Zhou, Bodine and Conte]

Hardware Design of Recovery-Free Value Prediction

- **Two flag bits**
 - Value speculative (vp) is added to every entry of the issue window.
 - Value prediction ready (vp_ready) is added to every physical register

- **Speculative execution**

Condition	Action
Src. operands ready	Issue un-speculatively and use the result to update value predictor
Src. not ready, but vp_ready is set	Issue speculatively(set vp) and write the result to physical reg file and set vp_ready bit

- Speculatively issued instruction resides in the issue window until it is issued un-speculatively

- **Issue logic:** Prioritize the un-speculative instructions in the issue logic and not performing stores and branches in (value) speculative execution.

Hardware Design of Recovery-Free Value Prediction

Example: (a = a->next)

1: load r1 = mem[A]

2: add r2 = r1 + 4

3: load r3 = mem[r2]

↓ Predicting r1

1. r1 = mem[A]	2'. r2 = r1' + 4
	3'. r3 = mem[r2]

.....

2. r2 = r1 + 4

3. r3 = mem[r2]

ready(r1) = false; vp_ready(r1) = true

ready(r2) = false; vp_ready(r2) = true

Hardware Design of Recovery-Free Value Prediction

- Enabling aggressive memory disambiguation for prefetch
 - Issue the stalled load instruction speculatively

Example:

1: load r1 = mem[A]

2: store [r1] = r2

3: load r4 = mem[r3]



“predicting r3”

1. r1 = mem[A] 3'. r4 = mem[r3]

.....

2. mem[r1] = r2

3. r4 = mem[r3]

A Special Form of Pre-Execution

- **Background: pre-execution or pre-computation [P. Wang et. al.] [A. Roth, et. al.] [C. K. Luk] [Zilles & Sohi]**
- **Multi-threaded architectural support (SMT, Hyper-Threading)**

```
//initialization of a
While (a != NULL)
{
  //processing the list
  ...
  //traverse the list
  a = a->next;
}
```

Main thread

```
//initialization of a
While (a != NULL)
{
  a = a->next;
}
```

Pre-execution thread

A Special Form of Pre-Execution

- No need for the multi-threaded architectural support
- Construct the pre-execution thread
 - each predicted value (or a presumably disambiguated load instruction) enables a set of dependent instructions to be executed speculatively => forming a pre-execution thread.
 - Dynamically formed based on the data dependence relationship thus taking advantage of the dynamic branch prediction.
- Trigger the pre-execution thread
 - Predicting a value or speculative disambiguating a load
- Execute the pre-execution thread
 - Reuse the existing dynamic instruction scheduling logic
 - Prioritize the main thread => no slow down of the main thread due to resource conflicts.
- Terminate the pre-execution thread.
 - When the normal execution (main thread) catches up with the pre-execution thread at the same instruction.

Outline

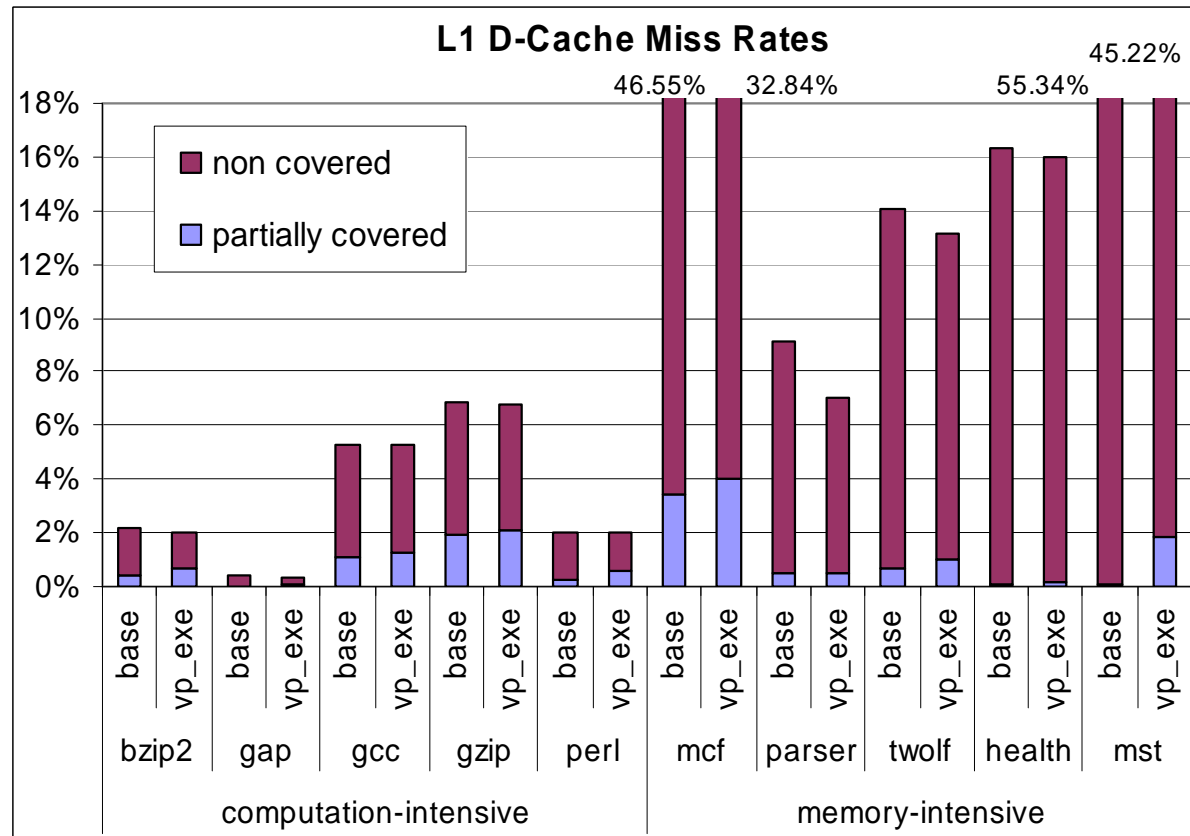
- Introduction
- Recovery-Free Value Prediction
- Implementation of Recovery-Free Value Prediction
- **Experimental Results**
- Limitations
- Summary

Simulation Methodology

- A MIPS R10000-like, dynamically scheduled, 4-way/64 entry issue superscalar processor
- L1 Data caches
 - 2-way 32 KB, 64-byte blocks, 32 MHSRs for D-cache
 - hit time = 2 cycles; miss penalty 10 cycles
- I-cache 4-way 64 KB, hit time = 1 cycle; miss penalty 10 cycles
- Unified L2 cache: 8-way 512 KB; miss penalty 80 cycles
- Disambiguation: Load waits until *all prior* store addresses are known
- Spec2000 INT, reference input data set, skip 800M instructions and simulate next 200M instructions; Olden, skip 2B for mst and simulate next 200M, for health, from start to completion.
- 4k entry tag-less stride value predictor.

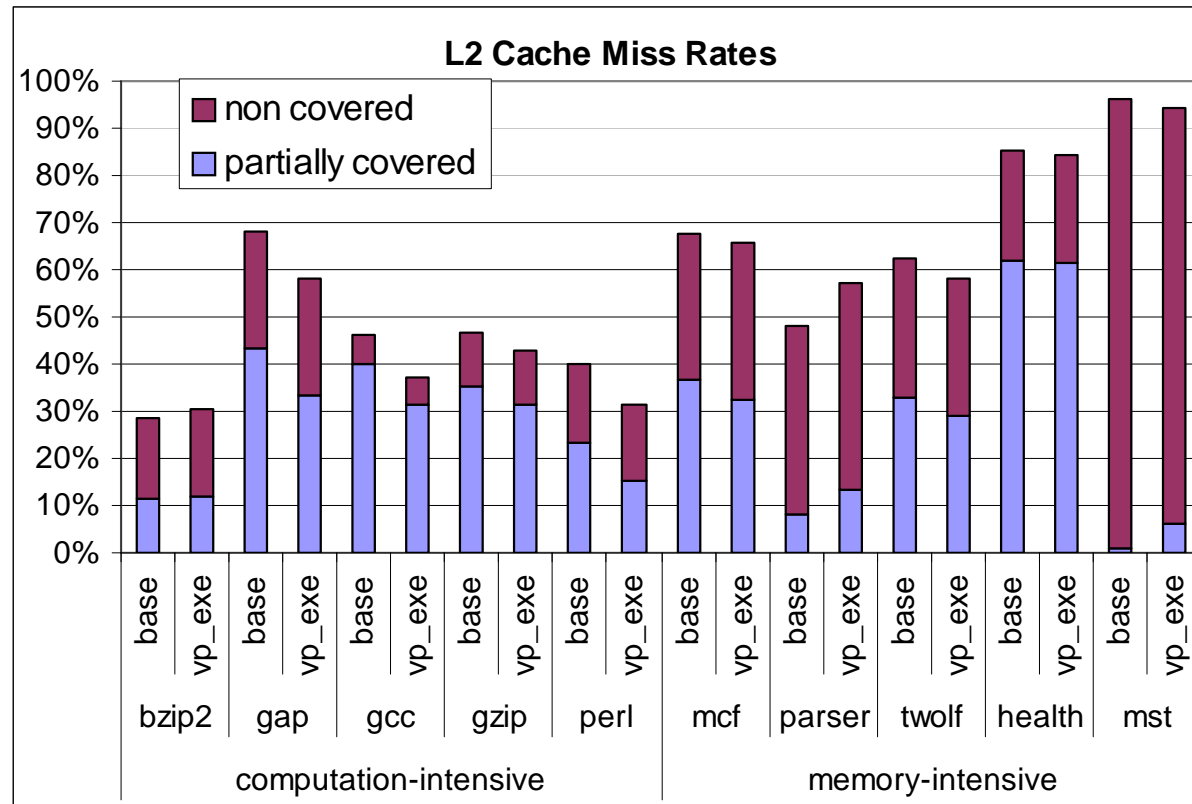
D-cache Results

- The L1 D-cache miss rates

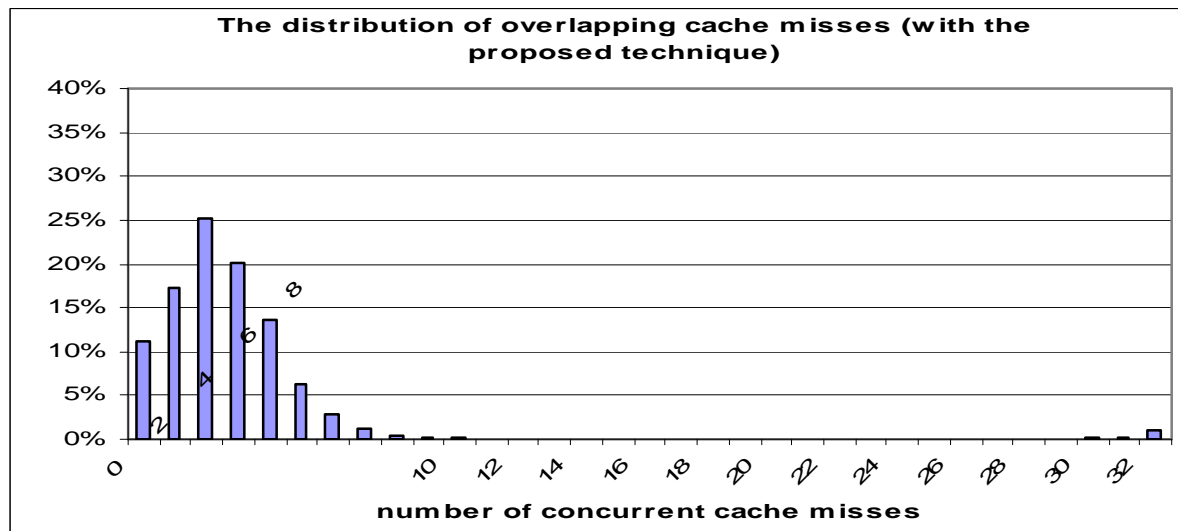
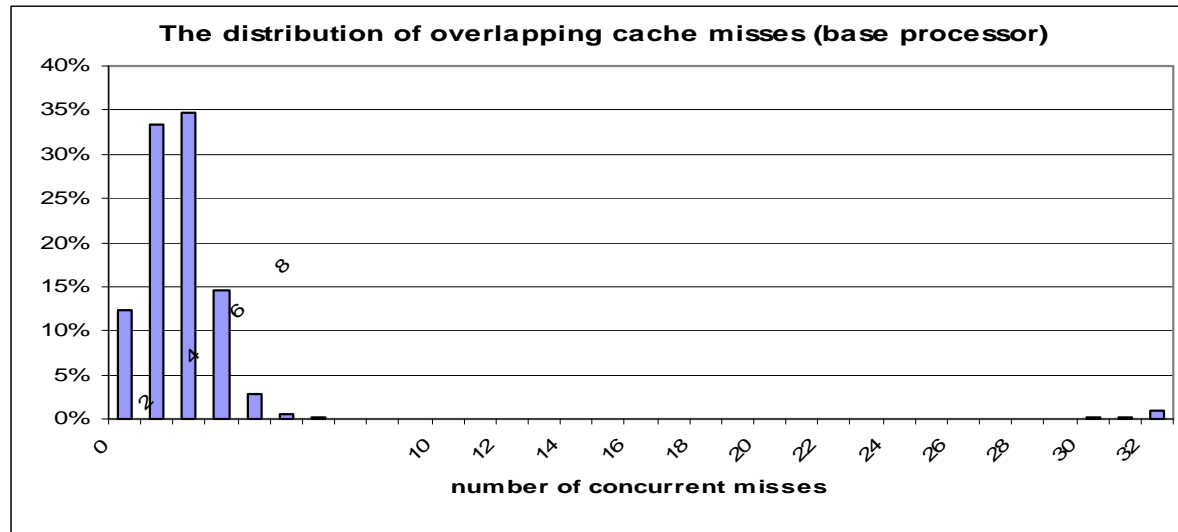


D-cache Results

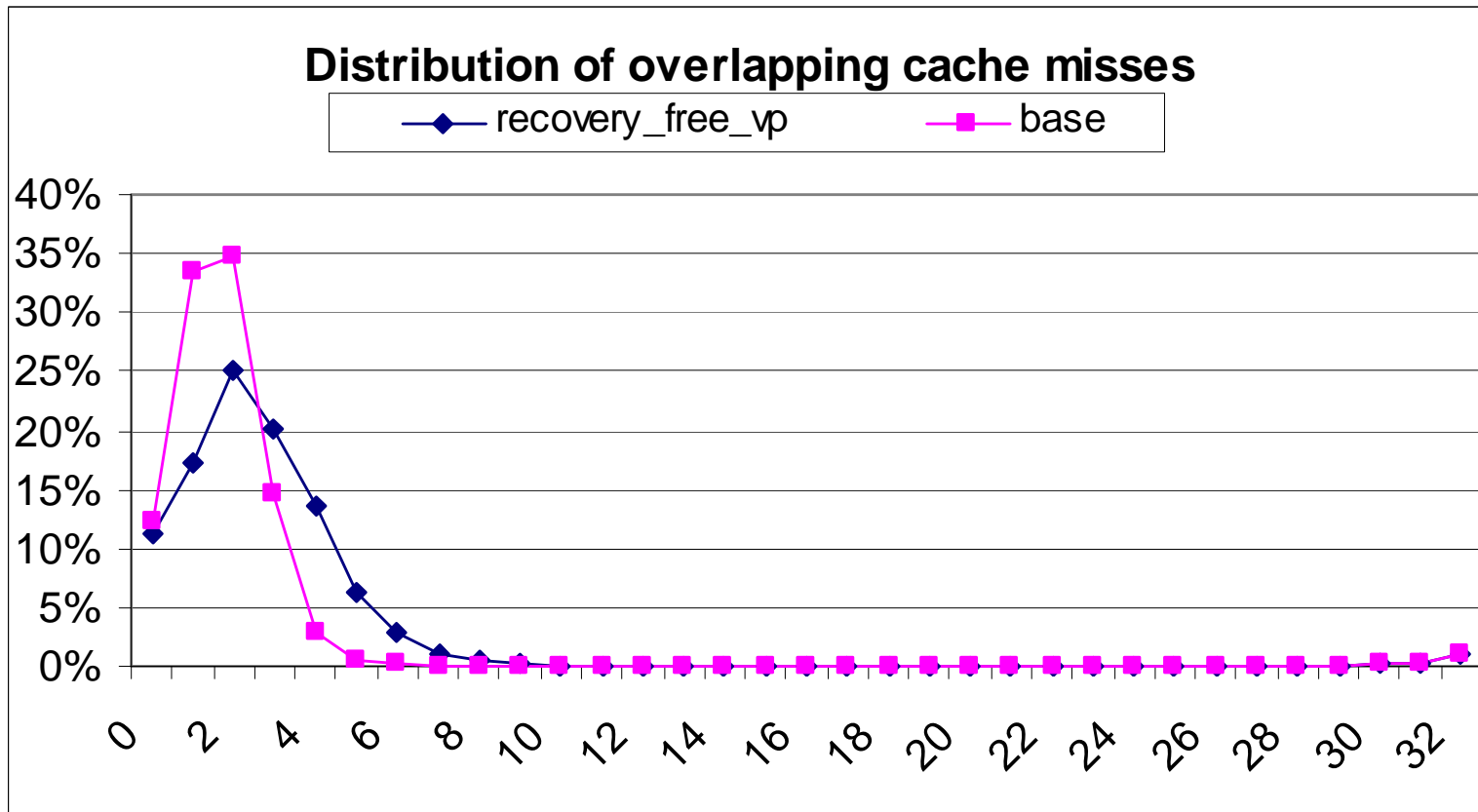
- L2 cache miss rates



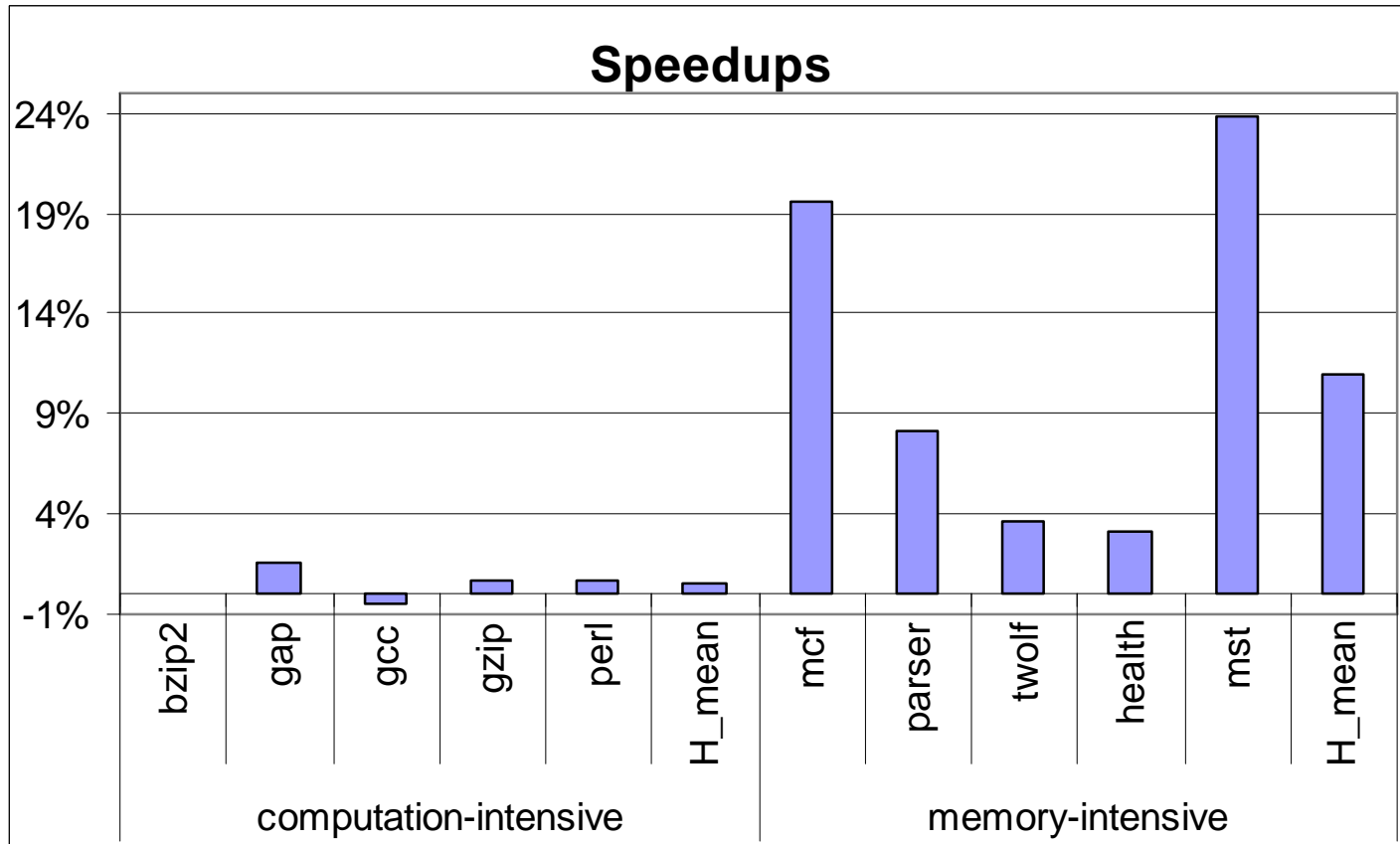
Memory Level Parallelism (MLP) Measurement (*mcf*)



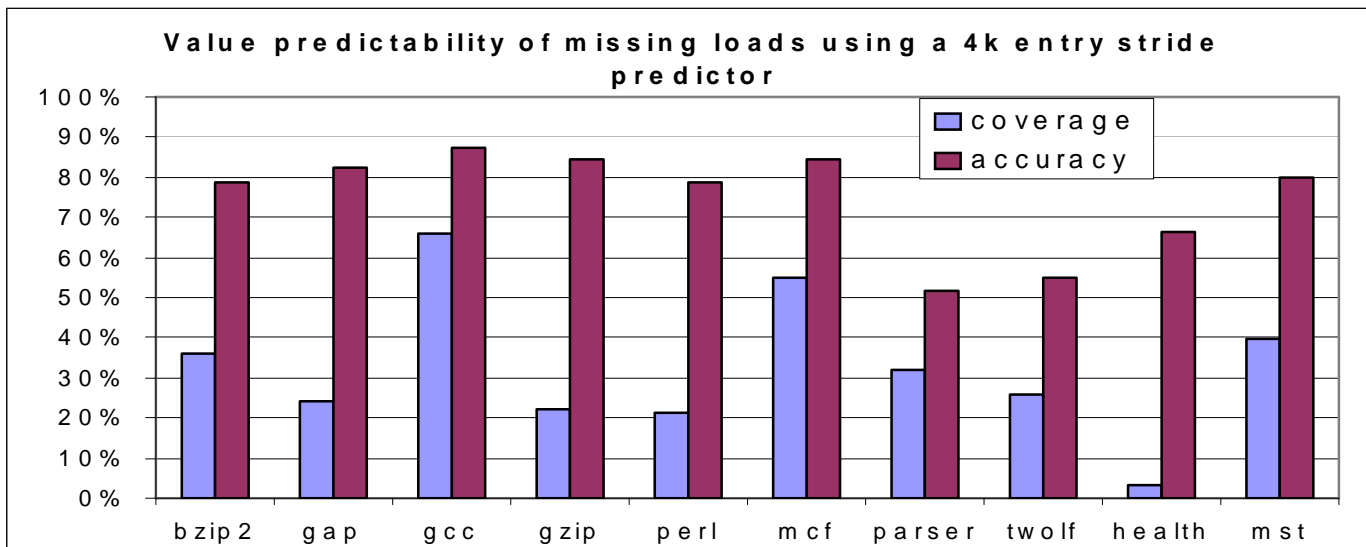
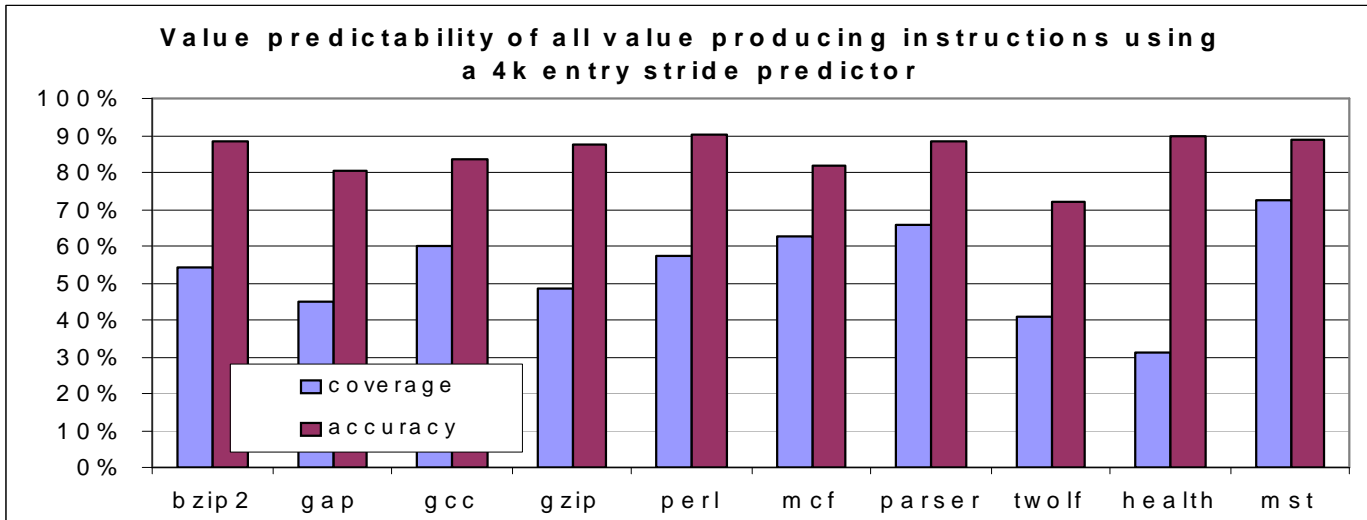
MLP Enhancement



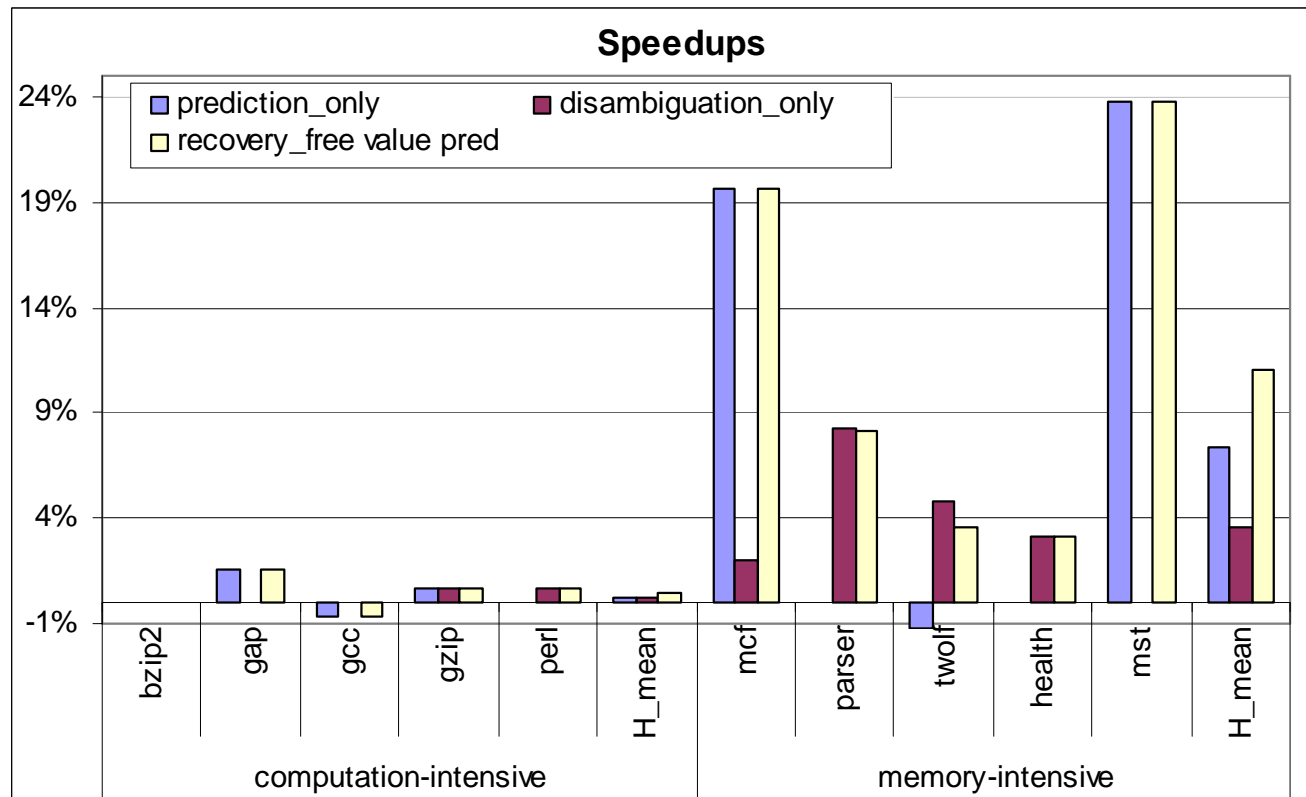
Speedup Results



Value Prediction Results

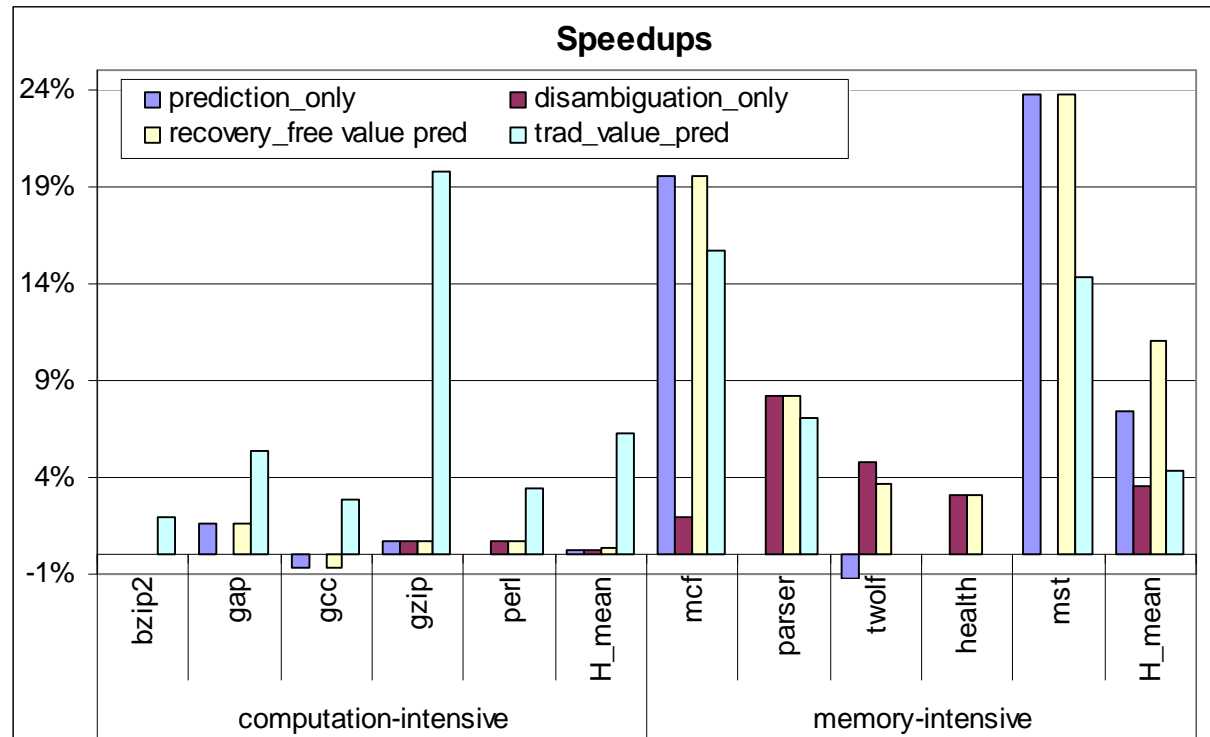


Which Dependence to Break



- Need to break both dependence to achieve best results. Similar observation is made in [Calder and Reinman]

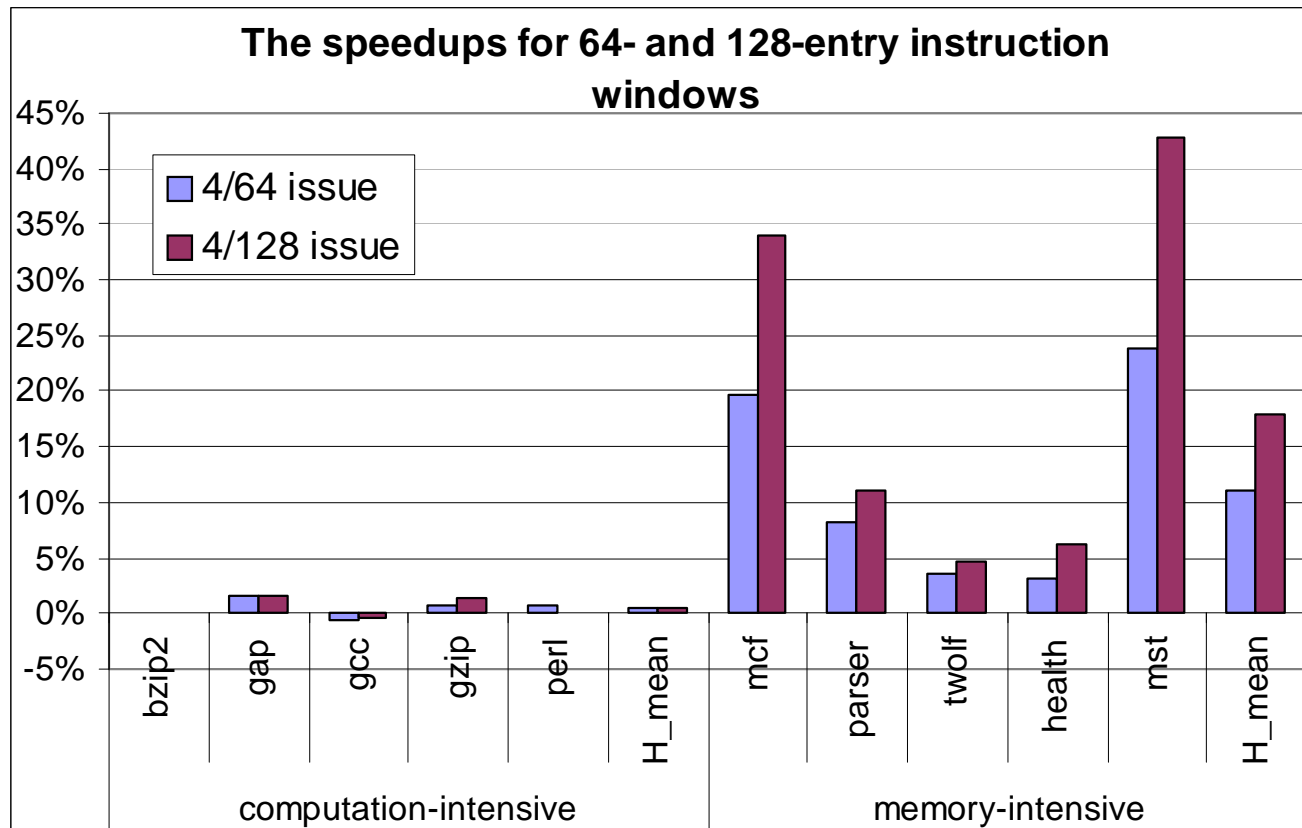
Comparison with Traditional Value Prediction



Recovery-free value prediction:

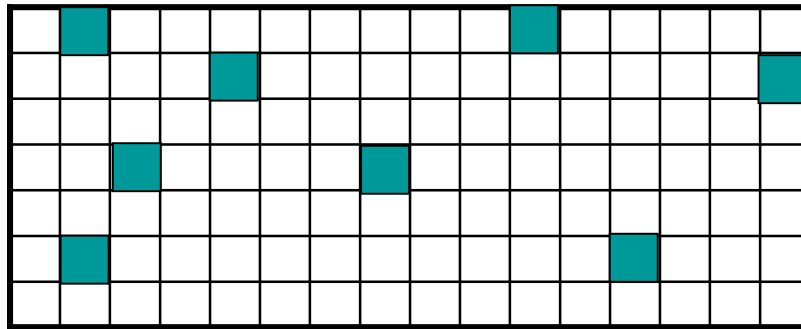
1. removes the associated validation/recovery/re-issue penalties
2. benefits from speculative disambiguation
3. Achieves better prediction results

Different Issue Windows



Limitations

- Most effective for long memory dependence chain (pointer chasing, memory intensive workloads)



- Far-flung parallelism
 - limited by instruction window size

Summary & Future Work

- **Proposed recovery-free value prediction to enhance memory-level-parallelism.**
- **Using recovery-free value prediction to focus on important loads only.**
 - **Only a few static loads are responsible for majority of dynamic cache misses [Abraham et. al.]**
- **Combine recovery-free value prediction with run-ahead execution [Dundas & Mudge, Mutlu et. al.].**