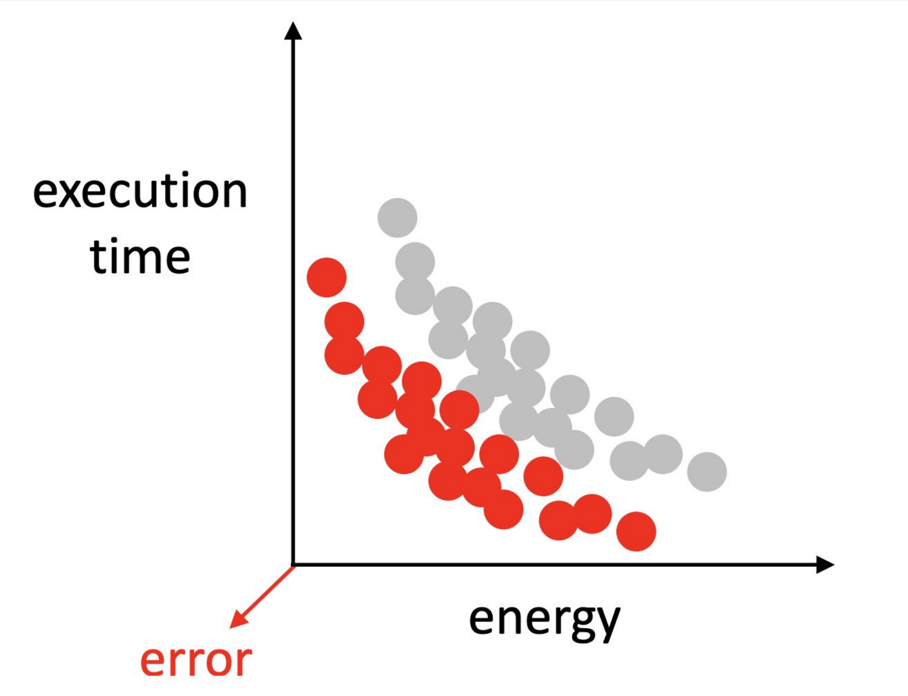
Load Value Approximation

Rahul Prabhu
Sai Mittapalli

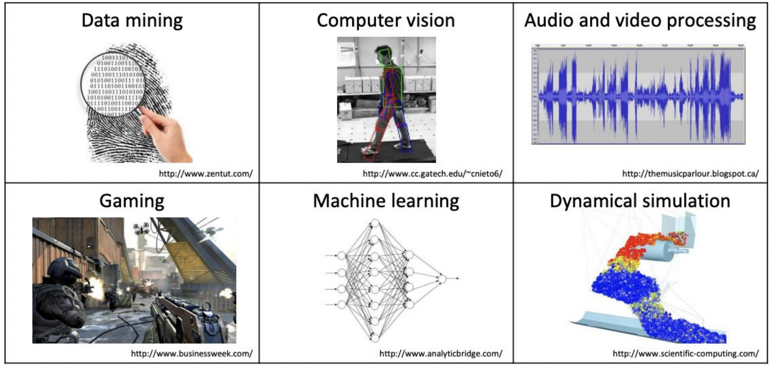


General Background

➔ A wide range of commercial, multimedia and scientific applications are inherently approximate

➔ They operate on noisy data and perform inexact computations (ex. Image processing, recognition, and mining applications)

➔ Applications exhibit value locality; they tend to reuse common values



Load Value Prediction

- Instead of waiting for data, the predictor generates a value and allows the processor to continue executing instructions speculatively.
- The prediction is validated against the actual value. If no match, the processor must rollback instructions.
- If the value is correct, the predictor increases its confidence for that value, same for the opposite
- In load value predictors, a load miss in the L1 cache still fetches the data from the next level of memory.

Problems with Existing Approaches

- ➔ Requires managing speculative values while risking costly rollbacks for inaccurate predictions
- ➔ Due to latencies of cache misses, processors need large buffers to store all speculative values for further validation
- ➔ Upon a misprediction, the processor must be able to quickly restore its registers and undo all speculative modifications
- ➔ Load value prediction typically performs poorly for floating-point values
- ➔ Possible for another thread to modify a speculative value, resulting in complications with the memory consistency model

The Authors

Joshua San Miguel

- Associate Professor in the Department of Electrical and Computer Engineering at the University of Wisconsin-Madison

Mario Badr

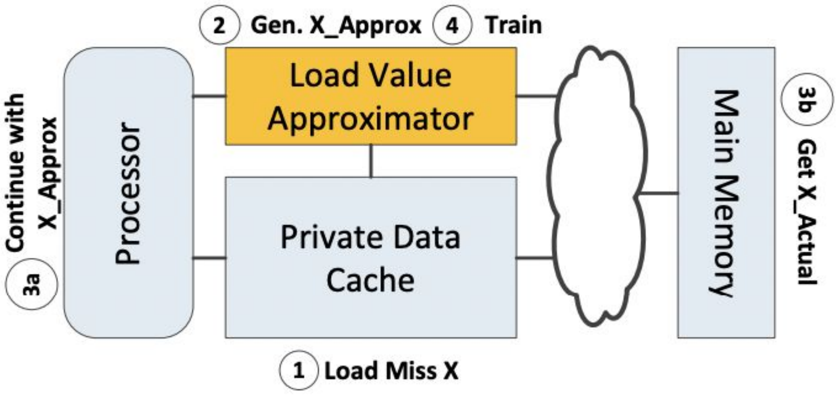
- Assistant Professor in the Department of Computer Science at the University of Toronto

Natalie Enright Jerger

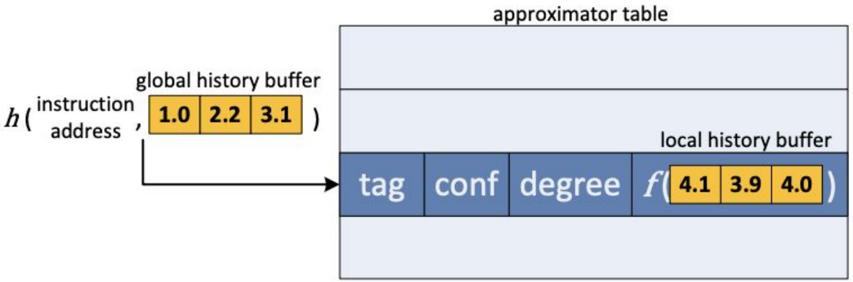
- Canada Research Chair in Computer Architecture
- Professor in the Department of Electrical and Computer Engineering at the University of Toronto.

Overall Implementation

- A load misses in the L1 data cache at **1**
- Load value approximator generates X_approx at **2**.
- The processor assumes this is the actual value of X and proceeds with its execution **3a** .
- A request is sent to the next level of the memory hierarchy to fetch the cache block containing the actual value of X at **3b** .

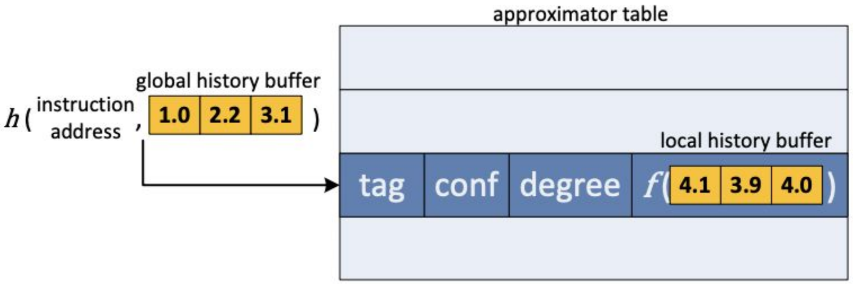


Approximator Design



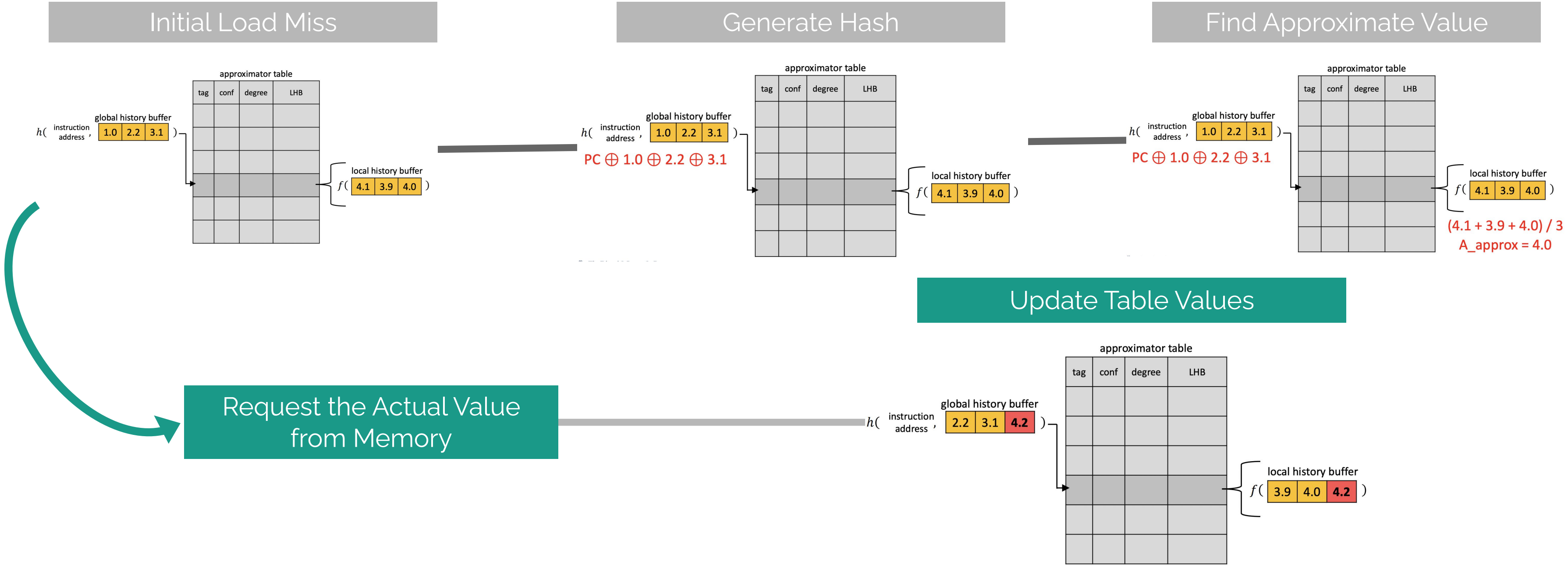
- Combine the computational and context based predictors into a single hardware structure
- The approximator consists of a global history buffer and an approximator table
- GHB is a FIFO queue that stores the values accessed by the most recent load instructions (Precise values not approximate)
- The hash value is the values in the GHB hashed together with the instruction address using a hash function
- The hash value indexes the direct mapped approximator table

Approximator Design cont.



- Each entry consists of a tag, a saturating confidence counter, a degree counter, and the LHB
- LHB stores the values accessed only by the previous loads that match the entry's tag
- LVA computes average of values for approximate values
- No rollbacks are needed since the actual value is used only to improve the accuracy

Overall Process Timeline



Relaxed Confidence Estimation

- The extent to which approximation can be tolerated is called the relaxed confidence window
- When approximating, use the approximate if the confidence counter is greater than 0
- The confidence counter is incremented by one if x_approx is close enough to x_actual, decremented by one otherwise

tag	conf	degree	LHB

Approximation Degree

- If an approximation is made, it is possible to not fetch the data block at all.
- The actual value's only purpose is to train the approximator for better accuracy
- Trades off approximation accuracy for better energy efficiency in the memory hierarchy
- The approximation degree specifies how many times we reuse a value generated by the approximator before we train it

Identifying Approximate Data

- Load value approximation requires programmers to annotate their code one should not approximate
- Data that directly affects an application's control flow
- Data used in the denominator of a division operation should not be approximated
- Memory addresses and pointers should never be approximated
- Identifying approximate data in frequently visited regions of code is the ideal scenario

Benchmarks

Approximate Floating-Point Data

- Blackscholes
- Ferret
- Fluidanimate
- Swaptions

Approximate Integer Data

- Bodytrack
- Canneal
- x264

What did the paper get right?

Methodology

- Two-Phase Evaluation
  - Design Space Exploration
  - Full-system Multiprocessor Simulation

Design Space Exploration

Benchmark	L1 MPKI	Instruction count variation
blackscholes	0.93	0.99%
bodytrack	4.93	0.05%
canneal	12.50	1.25%
ferret	3.28	0.60%
fluidanimate	1.23	0.17%
swaptions	4.92E-05	0.00%
x264	0.59	2.37%

- ➔ Uses Pin (dynamic binary instrumentation framework) to model private L1 data cache
- ➔ Pin simulator catches all load instructors that access approximate memory locations
- ➔ Pin allows rapid evaluation of performance, energy, and output error

Full-System Multiprocessor Simulation

Full System Configuration

Processor	4 IA-32 cores, 2 GHz, 4-wide OoO, 32-entry ROB
Private L1 cache	16 KB, 8-way, 1-cycle latency, 64 B blocks
Shared L2 cache	512 KB distributed, 16-way, 6-cycle latency
Main memory	1 GB, 160-cycle latency
Cache coherence	MSI protocol
Network-on-chip	2×2 mesh, 3-cycle routers
Technology node	32 nm

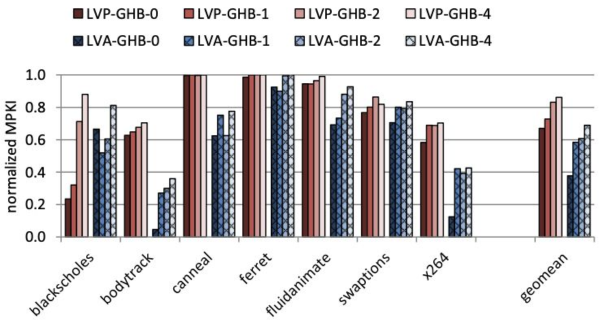
- ➔ Uses FeS2 cycle level x86 simulator
- ➔ Uses CACTI modeling tool to measure the dynamic energy consumptions of:
  - Caches
  - Main Memory
  - Approximator Tables

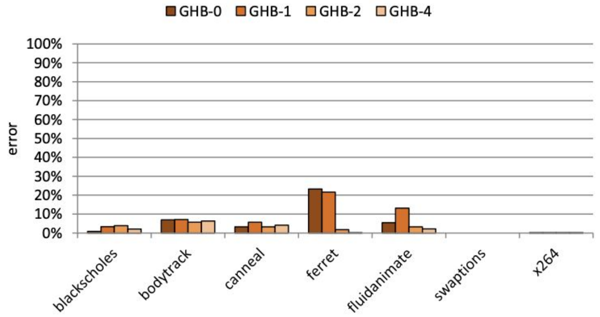
Evaluation

- Design Considerations:
  - Global History Buffer Size
  - Relaxed Confidence Thresholds
  - Value Delay
  - Approximation Degree
- Uses three metrics:
  - Misses-per-kilo-instructions (MPKI)
  - Blocks fetched into the L1 cache (fetches)
  - Output error
- Full-System Simulation:
  - Performance
  - Energy

Design Consideration: Global History Buffer Size

- ➔ Baseline LVA vs LVP for varying GHB sizes
  - ◆ On average, LVA achieves lower MPKI
  - ◆ MPKI increase with size b/c hashing more GHB values causes indexing challenges
- ➔ Impact of GHB size on output error
  - ◆ All <= 10% other than Ferret



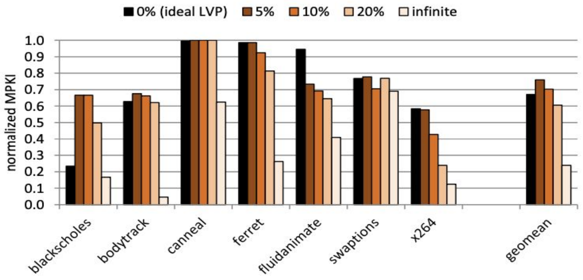


Relaxed Confidence Threshold

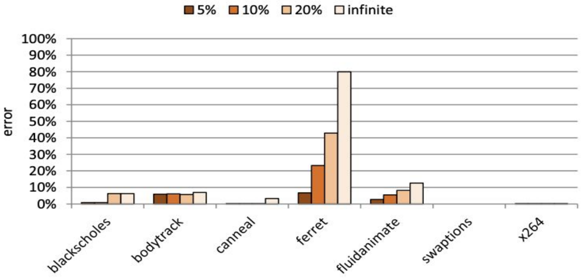
➔ Infinite relaxed confidence = data is always approximated according to values in LHB

➔ Key Takeaways:

- x264 has reduced MPKI and almost no error
  - Integer values are more open to approximation
- Ferret has increased error
  - Difficult to approximate vectors of floating-point data



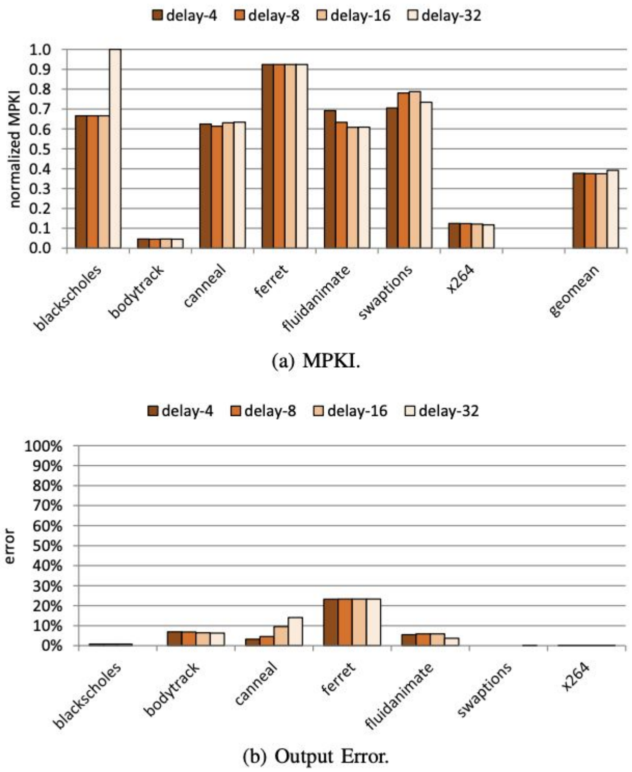
(a) MPKI.



(b) Output Error.

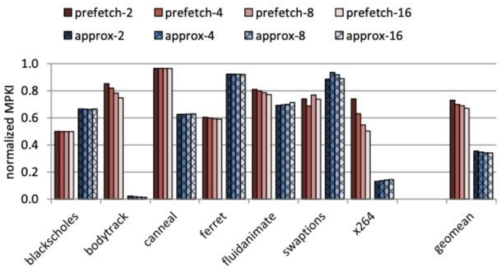
Value Delay

- ➔ LVA inherently tolerates inexactness
  - ◆ No significant impact on MPKI or error
- ➔ When data becomes too stale, approximation is rejected (blackscholes at delay-32)
- ➔ Output error is mostly constant except for canneal

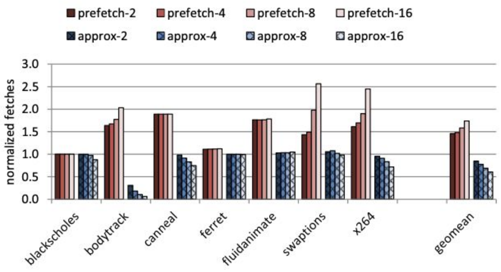


Approximation Degree

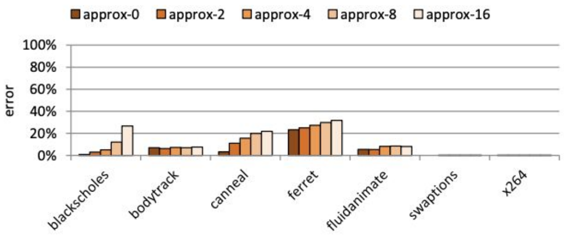
- ➔ Prefetching reduces MPKI at expense of increase in fetches and energy consumption
- ➔ LVA reduces both MPKI and # of fetches at expense of output error
  - ◆ Less frequent training of approximator



(a) MPKI.

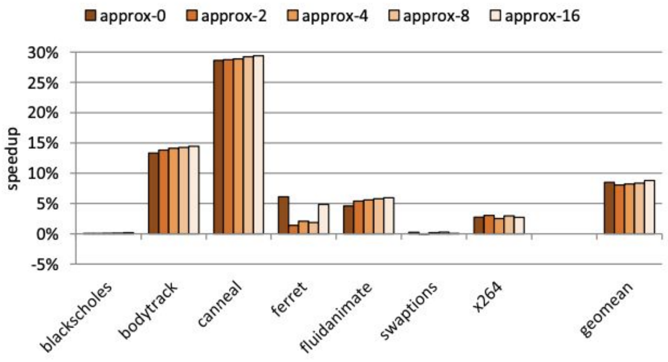


(b) Number of Fetches.

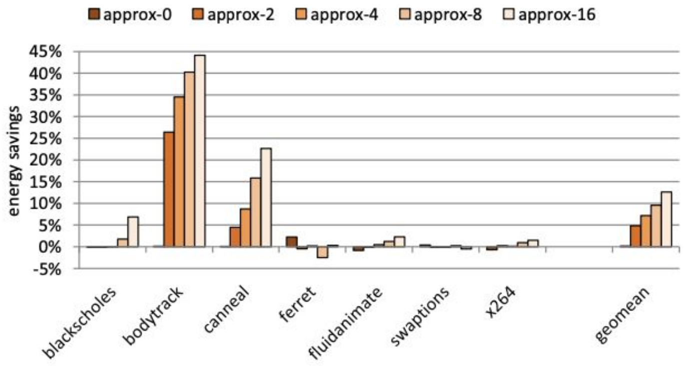


Full-System Simulation

- ➔ 8.5% performance improvement on average
- ➔ 41.0% reduction of L1 miss latency on average
- ➔ 12.6% energy saving on average
- ➔ Higher approximation degrees → greater energy savings



(a) Speedup.



(b) Energy Savings.

What did the paper get wrong?

Drawbacks

- ➔ Not sustainable for all types of applications
- ➔ Weak memory consistency - “If consistency … is a critical concern, [the] application is unlikely to be a candidate for approximation”
- ➔ High dependency on Approximation Degree
- ➔ Low chances of adoption
  - ◆ Willingly sacrificing accuracy in exchange for speed and energy

Questions?

References

- J. S. Miguel, M. Badr and N. E. Jerger, "Load Value Approximation," 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge, UK, 2014, pp. 127-139, doi: 10.1109/MICRO.2014.22.
- https://jsm.ece.wisc.edu/docs/sanmiguel-micro2014-presentation.pdf