Memory-Efficient Hardware Performance Counters with Approximate-Counting Algorithms

Jingyi Xu, Sehoon Kim, Borivoje Nikolic, Yakun Sophia Shao

Berkeley
UNIVERSITY OF CALIFORNIA

Berkeley Architecture Research

Motivation

- Hardware performance counters provide valuable info for many applications:
  - Workload characterization
  - Performance optimization
  - Task scheduling
  - Power modelling
- Challenge: 200+ performance events, only 10-20 performance counters
- Rerunning workloads multiple times:
  - Costly
  - Inaccurate(e.g. on cloud computing platforms)
- Idea: **trade accuracy for efficiency**

Previous works

- **Multiplexing performance counters**
  - Dedicate one counter to different events during separate time slices and extrapolate the full behavior
- **Multiplexing with rate-of-change scheduler**
  - Schedule events using cost functions based on recent hardware activities
- **Multiplexing with data-cleaning-based post-processing(CounterMiner)**
  - Clean data by removing outliers and filling missing values

Issue: error rate 5%~30%, no guarantee on the error bound

Sketching Algorithms

- A family of algorithm that compress large datasets into sketches
- Sketches:
  - Data structures
  - Much smaller than actual data
  - Surrogates of the original datasets for queries and approximate computations
  - Problem-specific, each answers one unique question and has unique structures
- Data compression is typically done using probabilistic techniques
- Statistically provable accuracy-memory tradeoffs
- Streaming: no revisits to data

Morris’ Algorithm

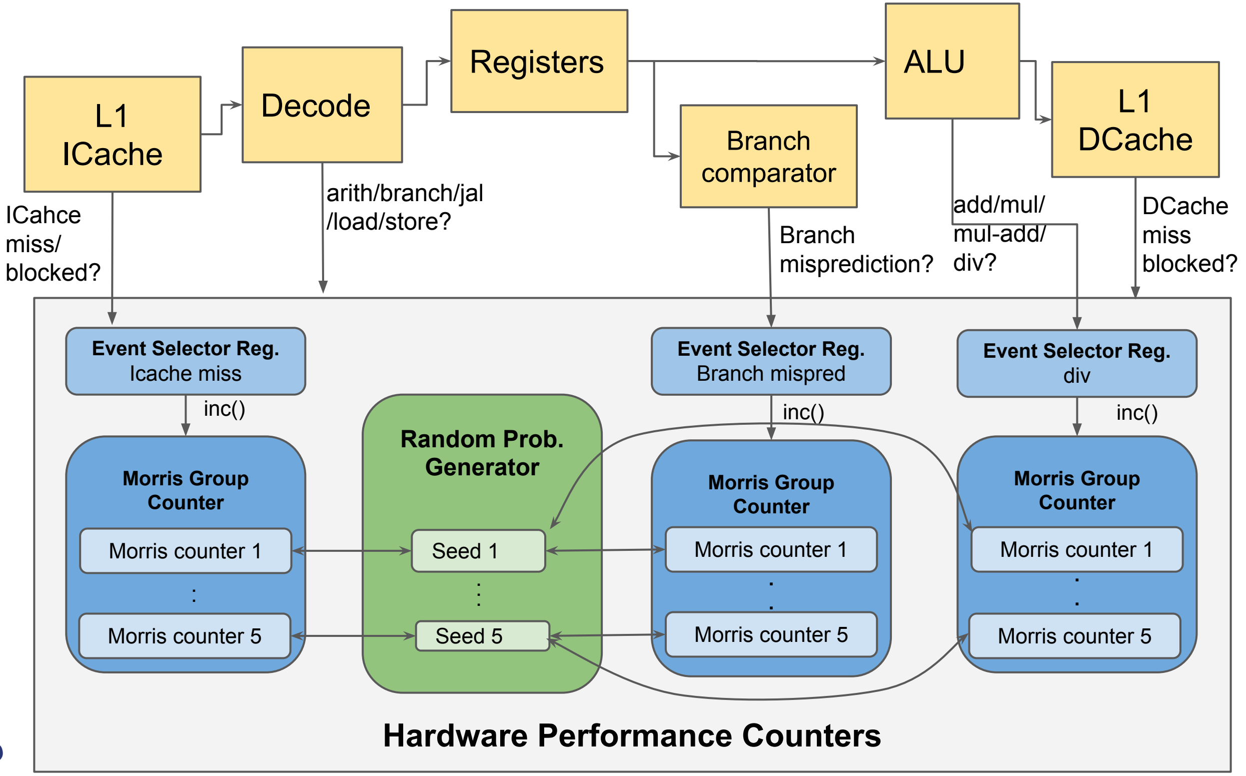
- Counts the number of events in a data stream
- One of the first **approximate-counting algorithms** that increment counts probabilistically instead of deterministically to save memory bits
- parameter $\alpha \in [1, 2]$
- Adjust accuracy by picking update base exponent and by averaging
- Bound on failure probability:

$$P(|c' - c| > \epsilon c) < \frac{1}{2\epsilon^2}$$

Pseudocode:

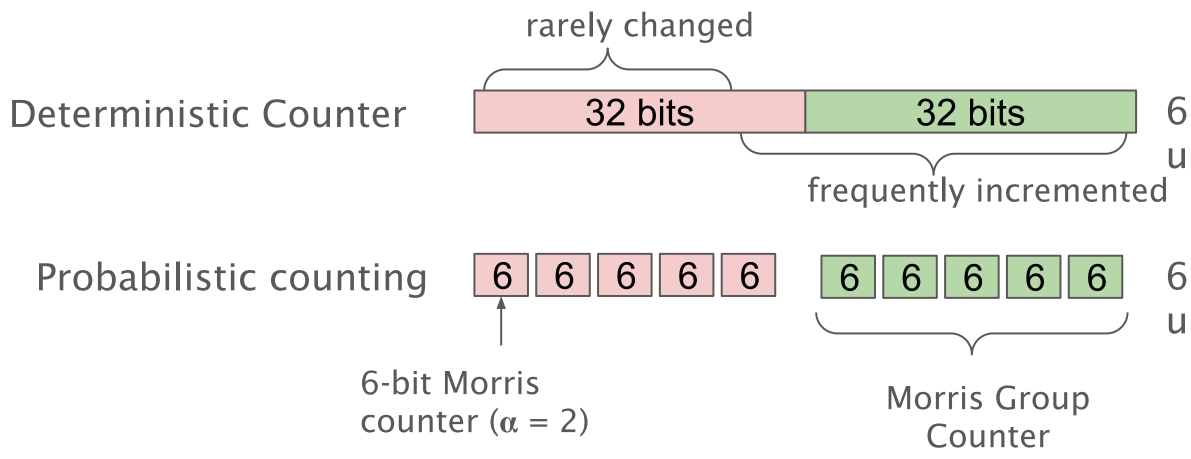
```
init():
    $X \leftarrow 0$
inc():
    Increment $X$ with probability $\frac{1}{\alpha^X}$
    and do nothing with probability $1 - \frac{1}{\alpha^X}$
query():
    Return $\alpha^X$
```

Rocket Core Integration



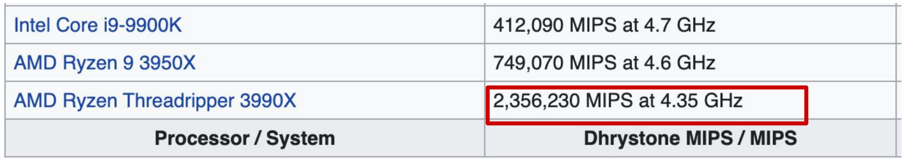
RISC-V in-order pipelined CPU core with 29 general 64-bit hardware performance counters and 29 event selector registers

Hardware Morris Counter Design



64 bits store 1 event, up to 2^{64} counts

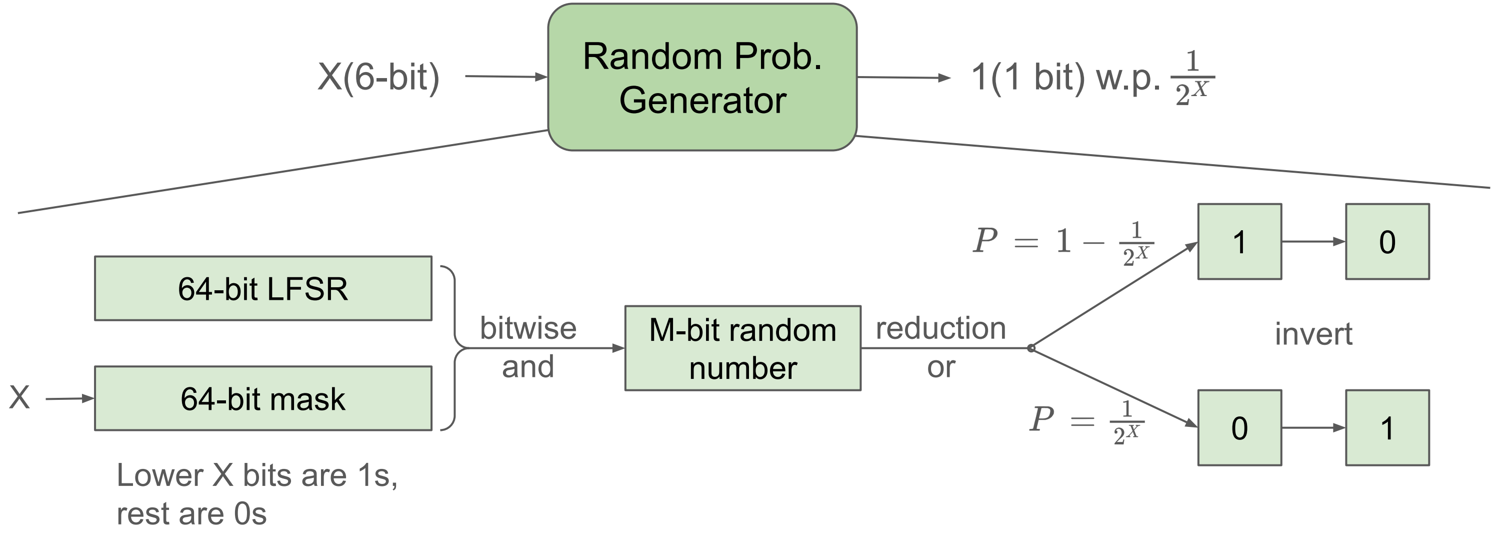
60 bits store 2 events, up to $2^{2^6} = 2^{64}$ counts



Processor / System	Dhrystone MIPS / MIPS
Intel Core i9-9900K	412,090 MIPS at 4.7 GHz
AMD Ryzen 9 3950X	749,070 MIPS at 4.6 GHz
AMD Ryzen Threadripper 3990X	2,356,230 MIPS at 4.35 GHz

event that occurs with probability of 0.1% requires 32 bits per second

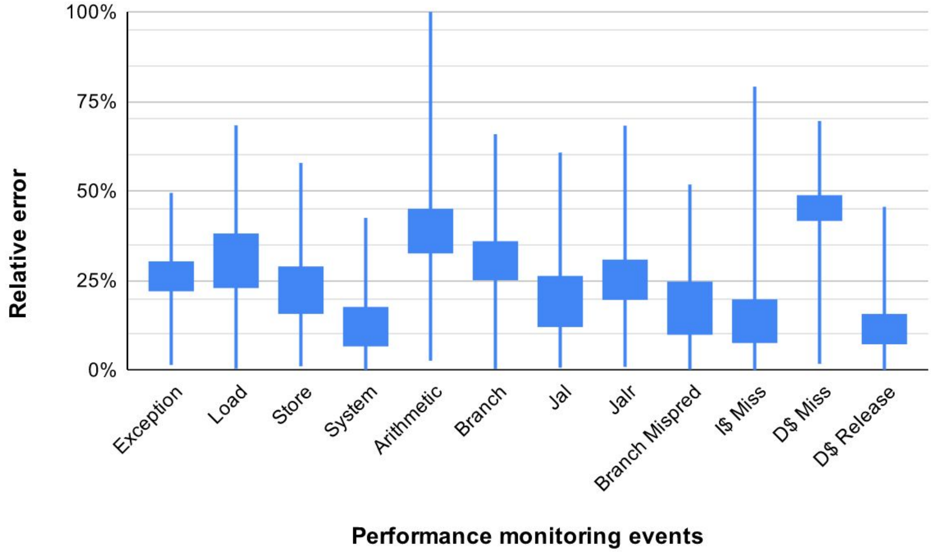
Hardware Probability Generation



- Multiple LFSRs with different initial values can give more random results than a single LFSR
- Need to generate at least 5 independent probabilities for Morris Group Counters
- Reduce overhead by shuffling LFSR's bits

Evaluation results

- Baseline: Rocket core's deterministic hardware performance counter
- Testbenches: spmv and vvadd, each simulated 50 times on Rocket Core with Morris performance counters
- Minimum errors stay consistently within 5%
- More variability in maximum errors, can get over 100%, but unlikely to happen
    - Probability of < 75% relative error is 89% in the worst case



Conclusion & Future work

- Preliminary results show that relatively simple morris counters can achieve accuracy comparable to the state-of-the-art multiplexed counter while using only half the memory of deterministic counters
- Next steps:
  - Experiment with different hardware pseudo-random number generators and test morris counter’s sensitivity to random sequences’ quality
  - Explore design space of Morris Group Counters
  - Reduce overhead of probability generation