Improving the World’s Fastest Cache Simulator
Andreas Sandberg, Peter Vestberg, Erik Hagersten

Motivation
- Traditional cache simulators incur a large runtime overhead; statistical cache simulators promise to significantly reduce the overhead.
- A low overhead simulator can be used to enable online optimizations. For example, disabling caching for streaming memory accesses.

Results
We have:
- Implemented a memory access sampler that runs on existing hardware.
- Demonstrated an average overhead of 17%, which is 2x better than previous samplers.

Statistical Cache Modeling

### Sampled Memory Reuse Distances

Statistical cache models use sampled reuse distances, i.e., the number of memory accesses between two accesses to a cache line, to estimate cache usage.

### Stack Distance Distribution

The reuse distance distribution is transformed into a stack distance distribution. A stack distance is the number of unique memory accesses between two accesses to a cache line.

### Miss Ratio Curves

An application’s stack distance distribution describes its cache miss behavior for all possible cache sizes. This information can be used to guide application optimizations.

Memory Access Sampling—Theory

- Program a performance counter to generate an interrupt after a random number of accesses.
- Measure memory accesses using performance counters.
- Detect reuse using some hardware mechanism, e.g., page protection.

Memory Access Sampling—Practice

#### Starting Samples

**Problem**
Counter overflows are delayed. The delay depends on the instructions in the pipeline and introduces bias.

**Solution**
Program the counter to overflow some number of events before the desired sample point. Read the counter and single step the remaining instructions.

#### Counting Accesses

**Problem**
Hardware idiosyncrasies introduce spurious counts when executing some operations, e.g., single stepping.

**Solution**
Measure the events known to cause spurious counts and account for their effect.

#### Detecting Reuse

**Problem**
Sampling requires several simultaneously active cache line watchpoints, but x86 only supports a few small watchpoints.

**Solution**
Use page protection to emulate cache-line sized watchpoints by protecting the entire page where the cache line resides and handle any false positives.

Performance and Accuracy

### Average Runtime Overhead

Average runtime overhead for a subset of the long-running SPEC2006 benchmarks. The sampler was configured to take 20000 samples spread over the entire execution. This resulted in an average overhead of approximately 17%.

### Runtime Breakdown

The overhead from watchpoint handling stems from the fact that we need to protect entire pages, even though we are only interested in a single cache line, while the single stepping overhead is due to compensation for performance counter inaccuracies.

### Accuracy

Miss ratio curves generated with the statistical cache model from both a reference sampler and our sampler. The accuracy of the sampler is generally very good.