

High Level Cache Simulation for Heterogeneous Multiprocessors

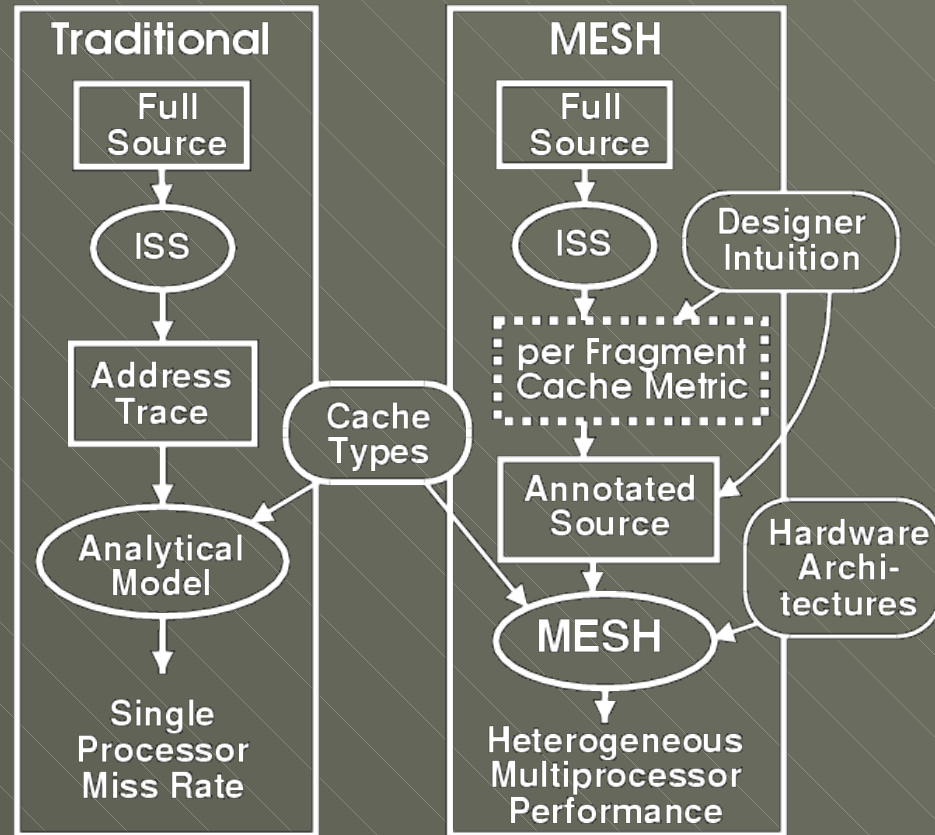
Joshua J. Pieper, Alain Mellan, JoAnn M. Paul,
Donald E. Thomas, Faraydon Karim

Carnegie Mellon University
STMicroelectronics

June 9, 2004

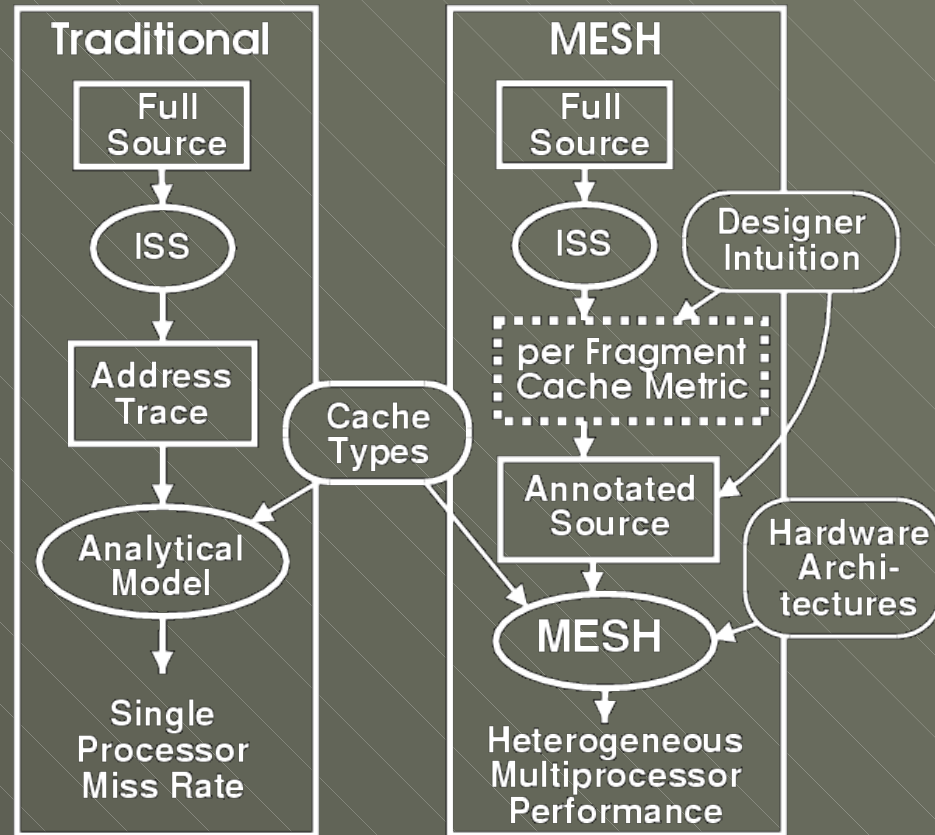
Motivation

- High-level (above the ISS) performance modeling will be important for virtually all future heterogeneous multiprocessors.
- A key challenge is high level cache modeling.



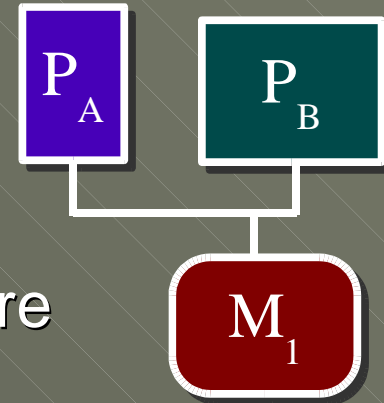
The Problem

- Given:
 - MESH: Modeling Environment for Software and Hardware
 - Traditional Cache Modeling: Stack Distance Histograms
- We compress stack distance histograms to enable merging analytical models with high level simulation.
- Results in greatly improved performance while maintaining accuracy.



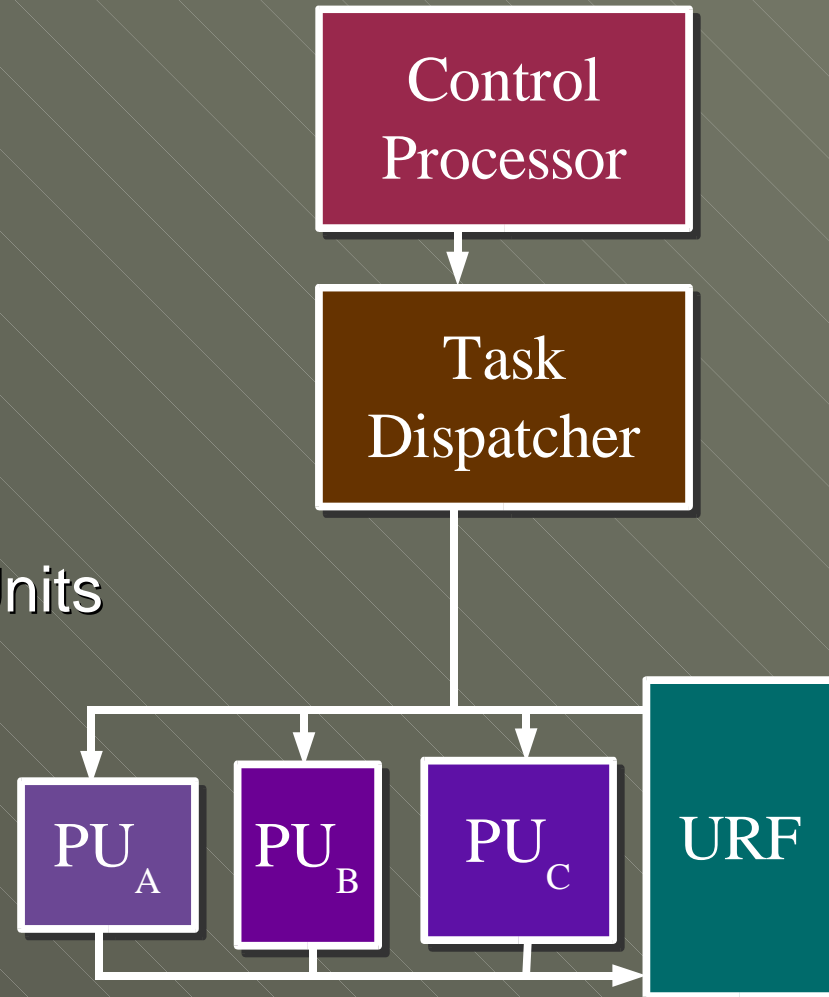
Heterogeneous Multiprocessors

- Embedded Multiprocessors
 - Many interacting programs
 - Many input datasets and usage scenarios
 - Heterogeneity from application specific nature
- Congestion and Contention
 - i.e. memories, busses, networks
 - Necessary to determine overall performance
 - Must know how performance of individual components varies with time to see their interactions
- Existing high-level cache tools give only one number for an entire program run, there is no information on how cache behavior changes over time.



Hyperprocessor Template

- Template architecture for heterogeneous embedded multiprocessors.
- Exploits task-level parallelism to reduce cost and design time.
- Superscalar task processor
 - Tasks \Leftrightarrow Instructions
 - Processors \Leftrightarrow Functional Units
- Possibly hundreds of unique processors.
- Design exploration at the ISS level is nearly impossible.



Proposed Simulation Technique

- Measure the cache behavior of program fragments
 - Retains localized information of cache behavior over time
 - Back-annotate the behavior into a high-level simulation
- Data dependent behavior
 - If the fragments are chosen well, many data-dependent behaviors can be captured as well.
 - Most data dependent behaviors relate to the amount of input data, or the number of loop iterations.

```
function1();  
if (condition)  
    function2();
```

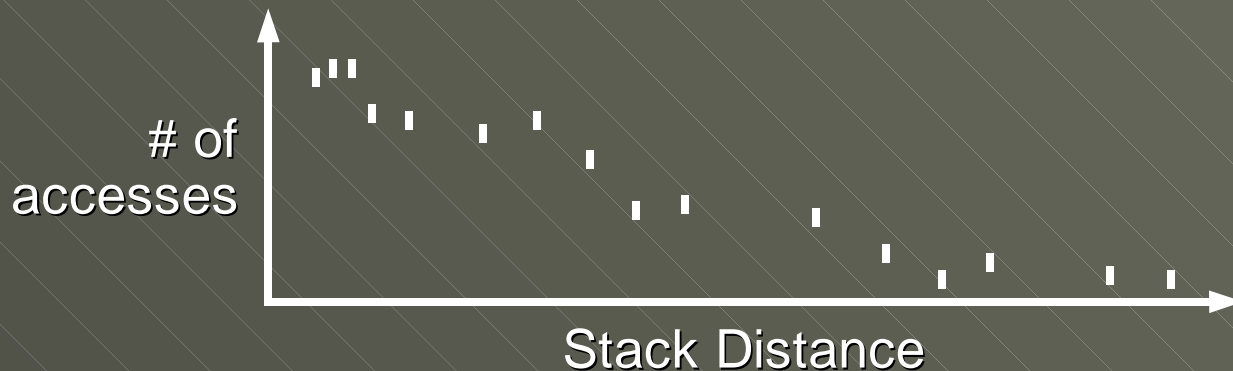
```
function3();
```

```
for (i=0;i<loop;i++)  
    function4();
```

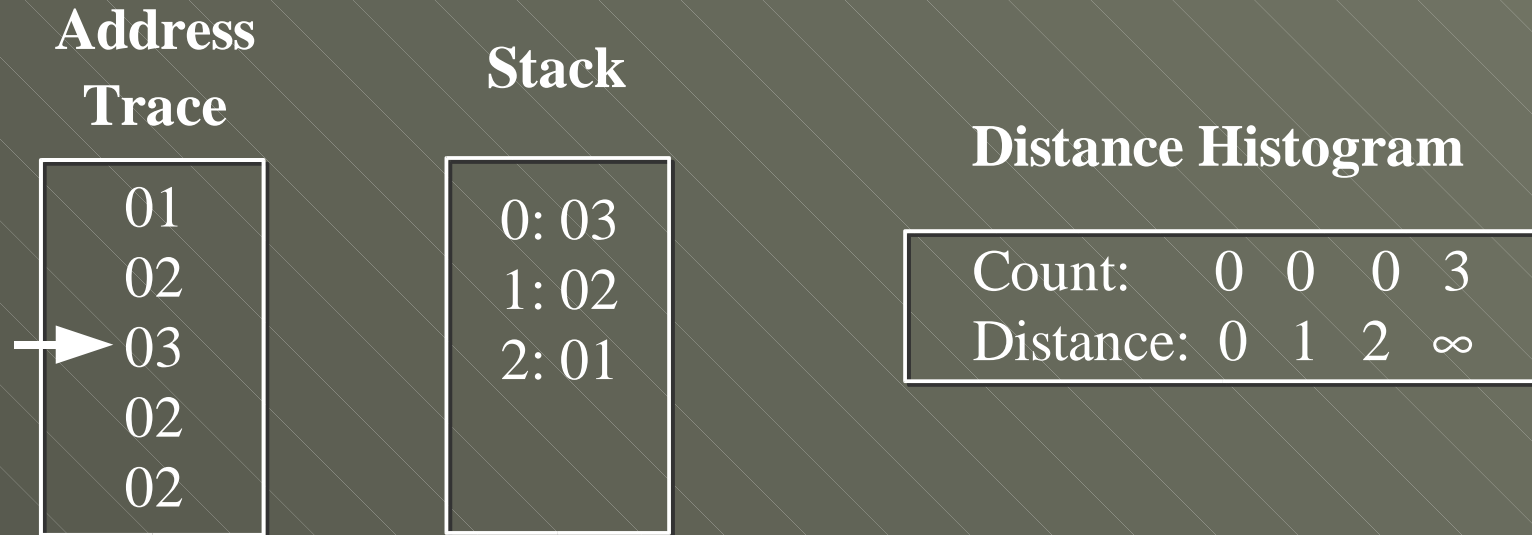
```
while (condition) {  
    function5();  
    function6();  
}
```

Stack Distance Histograms

- How to measure the behavior in a fragment?
- Stack distance histograms
 - Fast to evaluate
 - Exact for fully associative caches
 - Dependent on cache line size
 - Require large amounts of data to store
- We use compression to make this work for fragments

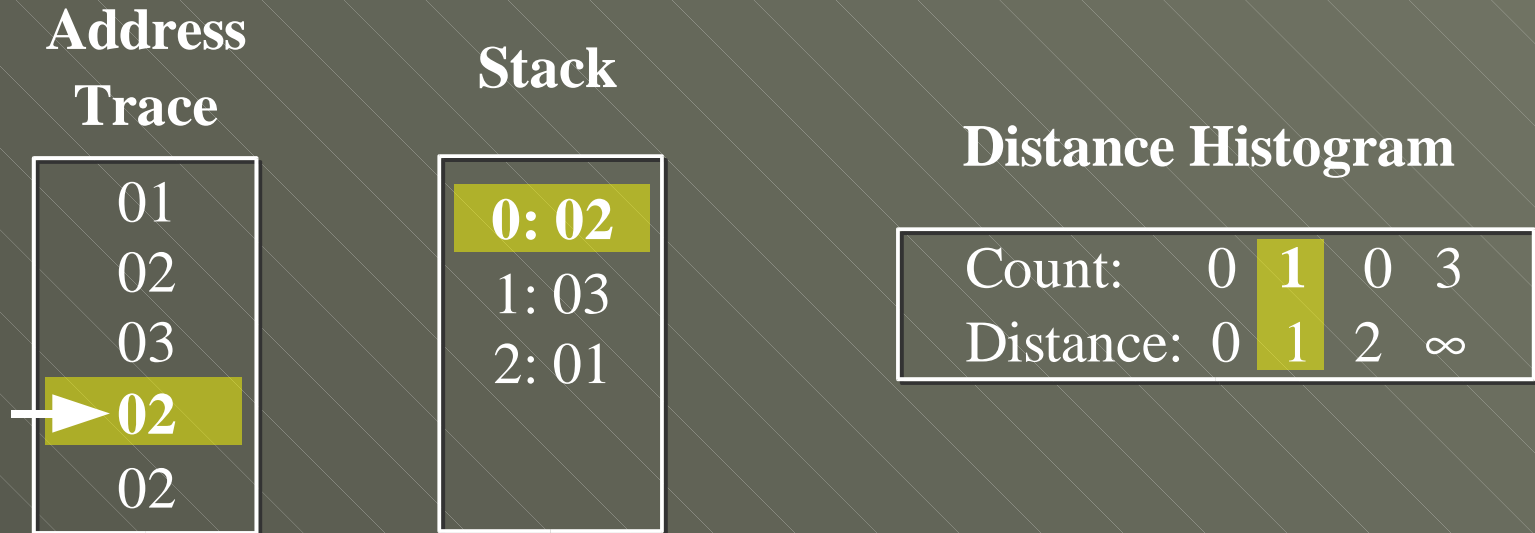


Example Histogram



- The address trace is an ordered set of cache lines.
- The stack is a simple FILO structure, with the ability to search for a particular item.
- The distance histogram records where in the stack each cache line is found.

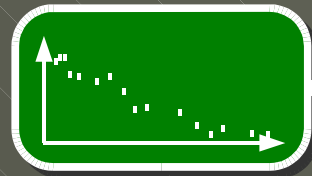
Example Histogram part 2



- To advance one step in the address trace, locate the cache line in the stack, and move it to the top.
- Record the position in the stack where the cache line was found in the distance histogram.
- To find the misses for a fully associative cache, sum all the distances greater than the cache size.

Compression

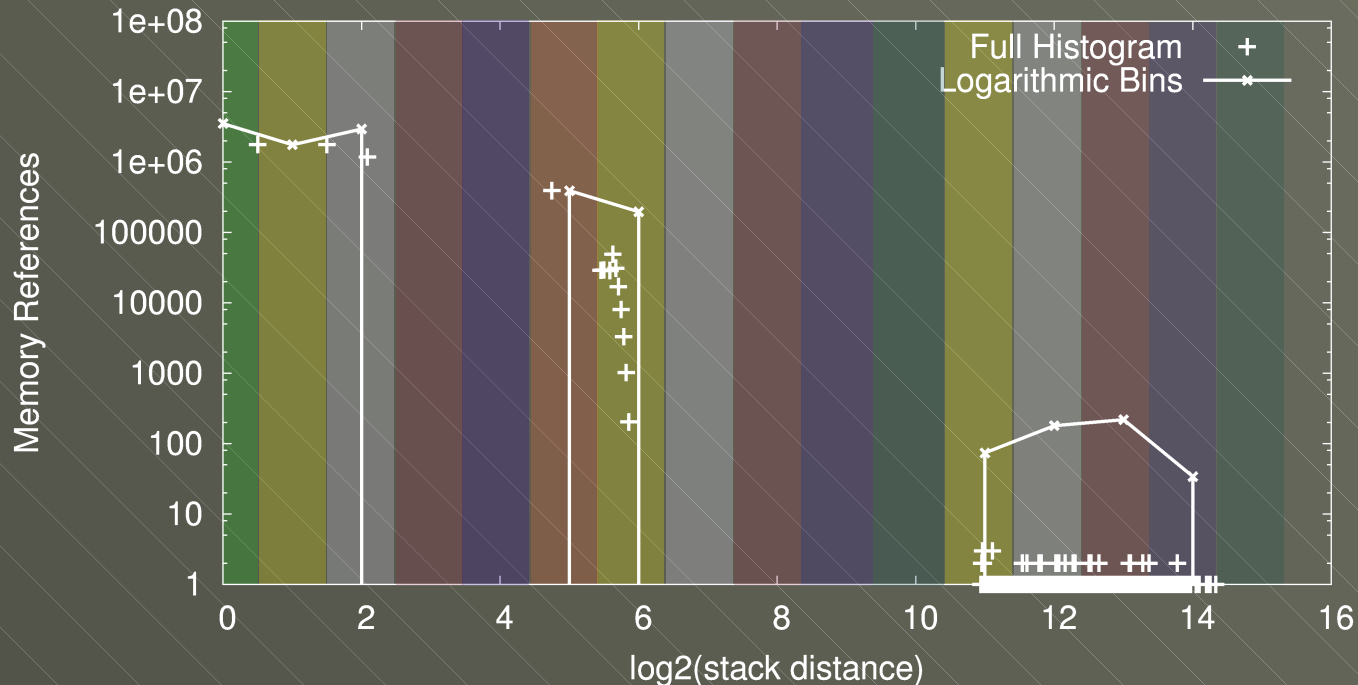
- Histograms require too much storage to annotate
- Patterns emerged when profiling sample applications
 - Many histograms had contiguous regions that were similar in access density
 - Most cache structures are efficiently implemented in powers of two
- We use two techniques to compress the histograms such that they can be back-annotated:
 - Logarithmic bins
 - Adaptive width averaging



```
while (condition) {  
    function5();  
    function6();  
}
```

Logarithmic Bins

- Enlarge bins to record distances from 2^n to 2^{n+1}
 - Reduces number of bins exponentially
 - Results will still be exact for any fully associative cache
 - Additional error is introduced for the set-associative approximation.

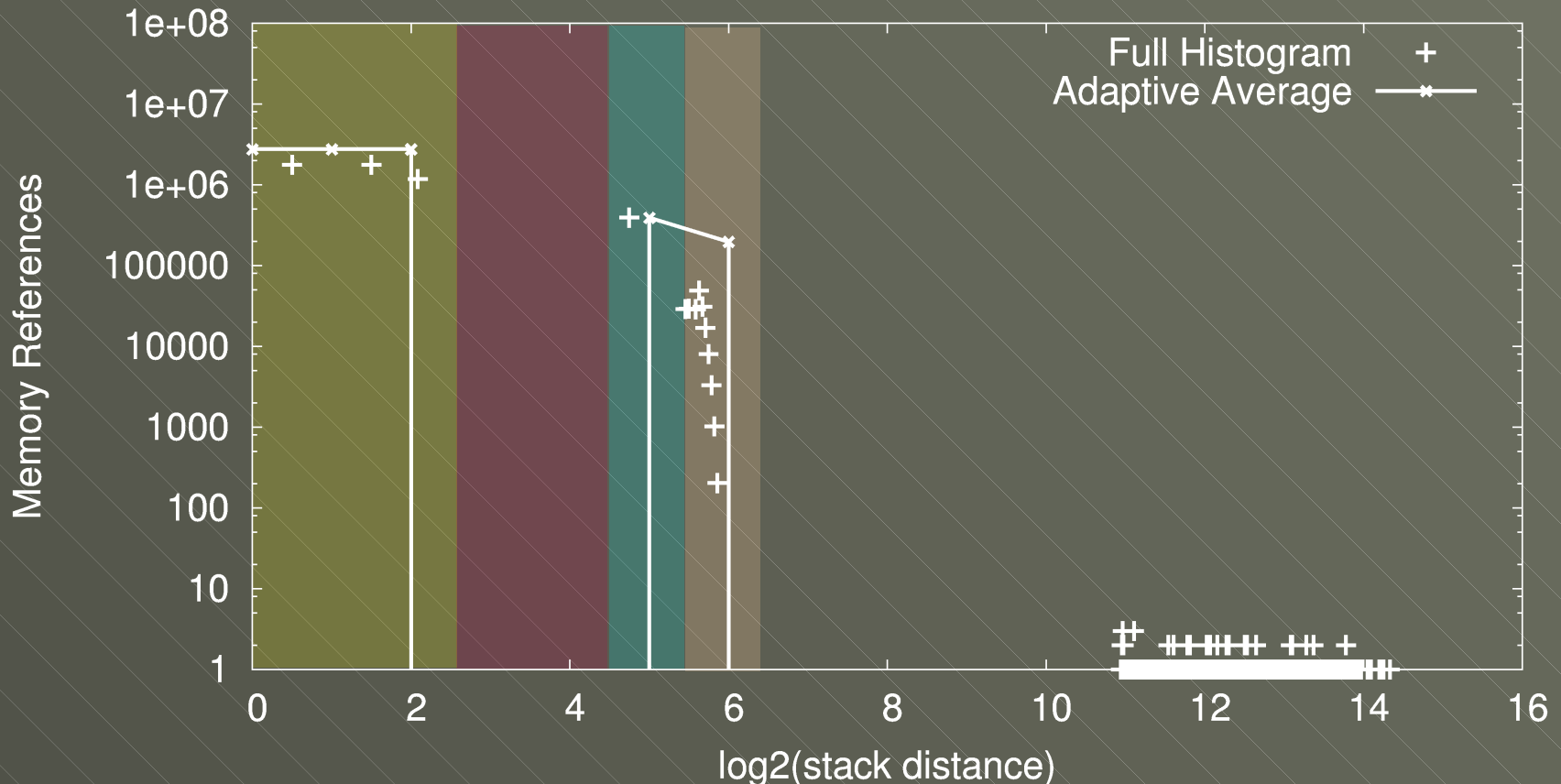


Adaptive Width Averaging

- Even with logarithmic binning, the number of bins to store is large for back-annotation, for Sphinx3 around 18.
- We further compress the histogram by selecting neighboring bins that have similar number of accesses, grouping them together, and only storing the average.
- For 3-5 such regions, the possible choices can be exhaustively searched to find the minimum error for every histogram.

Adaptive Width Averaging 2

- For the example program fragment shown before, the histogram after adaptive averaging with 4 regions looks like this.



Experiments

- Benchmarks: MiBench, Sphinx3
 - Blowfish, GSM: Small working sets, regular data accesses
 - Sphinx3: Large working set and data dependent memory accesses
- First, evaluated the two techniques for small examples.
- Second, annotated a single program and ran it on a uniprocessor.
- Last, executed a multiprocessor capable version of Sphinx3 on a Hyperprocessor architecture.

Error from Logarithmic Bins

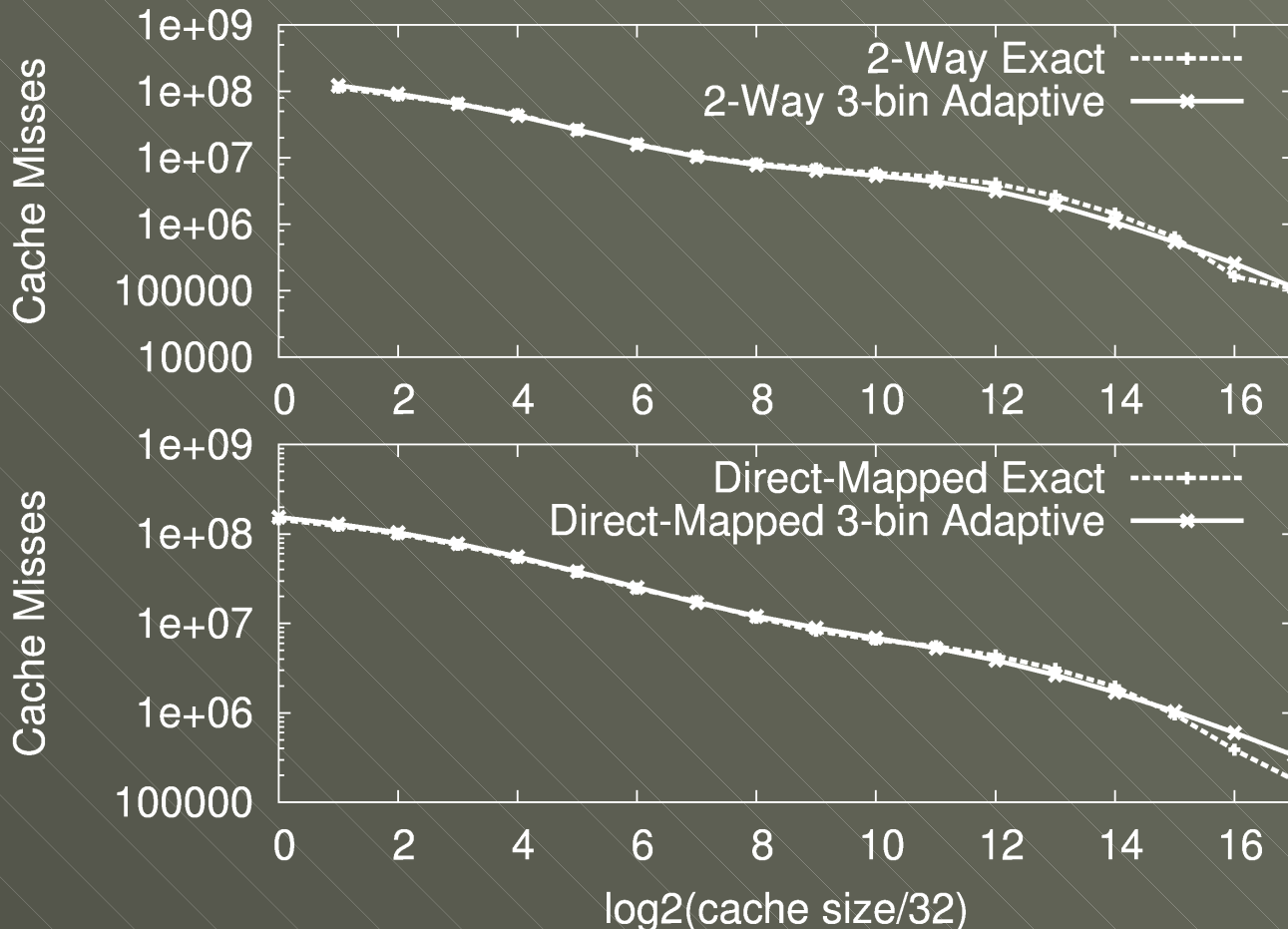
- Created single histograms representing entire programs, and logarithmically binned them.
- For some experiments the average error is greater, for others it decreases.

Average Error from Trace-Based Simulation

Experiment	Full Histogram	Logarithmic Bins
Sphinx Direct	8.9%	12.7%
Sphinx 2-way	6.4%	11.1%
Sphinx 4-way	4.0%	8.6%
Blowfish Direct	21.1%	16.1%
Blowfish 2-way	21.7%	11.6%

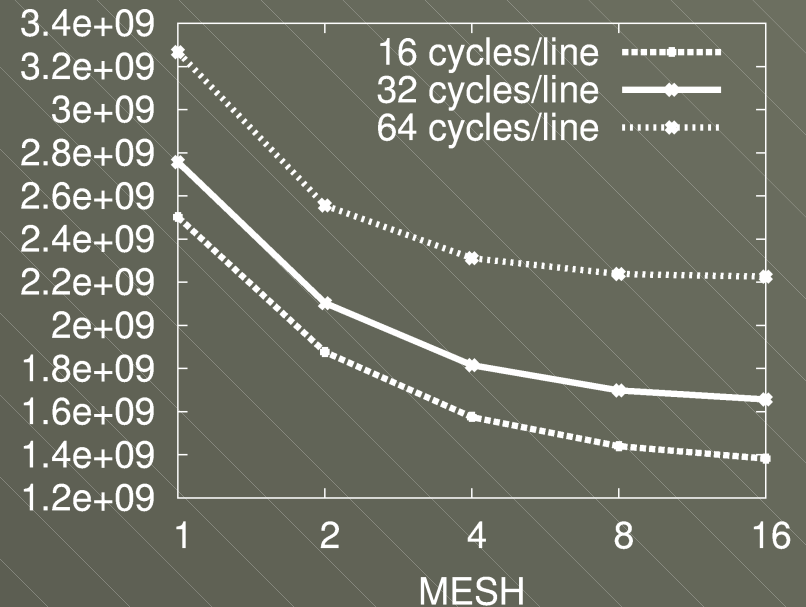
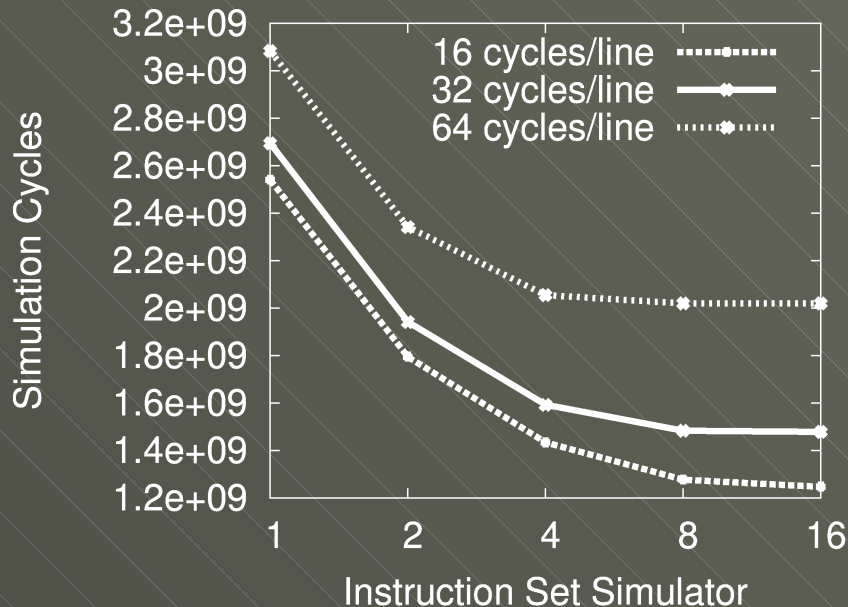
Full Metric Simulation

- Sphinx3 benchmark simulated on a uniprocessor.



Multiprocessor Example

- Multiprocessor simulation shows that the results track well with ISS implementations.
- These results also include the effects of our high level interconnect modeling approach as published in DATE04, so errors are not just due to cache effects.



Heterogeneous Multiprocessors

- Since the metric can quickly be evaluated against any cache architecture, exploring heterogeneous systems is feasible.
- As an example, we used the multiprocessor system from before, but modified the cache setup so that all but the first slave processor had only a 1k cache.

Performance (Simulation Cycles x 10⁹)

# of Processors	All 1k	One 8k	All 8k
1	3.02	2.76	2.76
2	2.36	2.11	2.10
4	2.07	1.82	1.82
8	1.95	1.71	1.70
16	1.90	1.66	1.66

Heterogeneous Multiprocessors

- The performance of the architecture with only one large cache is nearly identical to that of an architecture with all large caches!

Performance (Simulation Cycles x 10⁹)

# of Processors	All 1k	One 8k	All 8k
1	3.02	2.76	2.76
2	2.36	2.11	2.10
4	2.07	1.82	1.82
8	1.95	1.71	1.70
16	1.90	1.66	1.66

Simulation Performance

- Observations
 - For embedded systems, you simulate against many datasets and architectures.
 - It is desirable to have faster simulation time at the expense of simulation setup.
- Our techniques do just this.

CPU Time (seconds) Comparison

	Trace-Based	Metric-Based	
	Simulation	Profiling+Annotation	Simulation
GSM	72.4	157.0	0.24
Blowfish	27.6	72.5	0.52
Sphinx3	48.8	8266.5	1.87

Conclusions

- Logarithmic binning and adaptive average compression enable high-level software timing estimation to include the effects of cache architecture.
- There is a one-time software annotation cost, after which simulation is 20-50 times faster than trace-based.
- Accuracy is within 20%, often much closer, enough to guide early design space exploration and capture the first order effects of cache size and associativity.