Characterizing Memory Hierarchies Using Microbenchmarks
Christopher Celio, Krste Asanovic, Dave Patterson

Problem
How can we (software) figure out the important characteristics of our hardware?
• How many cores (hardware threads) are available?
• What is the size and latency of each level in the cache hierarchy?
• How are the caches shared between cores?
• What are the communication costs between cores?

Motivation/Applications
Published hardware specs may not tell the whole story:
• Some parameters are unpublished (e.g., cache-to-cache transfer characteristics).
• Parameters may be misleading (e.g., offchip bandwidth may be unsubstantiable).

Simulators are difficult to verify and tune.
• Microbenchmarks can characterize existing machines.
• Simulator can then be tested to verify that performance/behavior matches.
• Useful for finding bugs that render incorrect simulation results.

Auto-tuners may have incomplete knowledge of the system.
• Auto-tuned software may be restricted to run on a sub-set of the hardware (a VM or an OS partition)

Challenges
Complex interactions of the memory hierarchy pollute the results.
• virtual memory, TLBs
• memory prefetchers
• adaptive cache replacement policies
• cache coherence protocols
• memory controller interactions
• Operating System, schedulings

required run-time to accurately measure the system can be significant (what is feasible for real hardware may yield less reliable results for simulators).

Parameters
The following parameters are some of the characteristics that can be ascertained by software:

number of cores (hardware threads)
• run private algorithm on each thread
• increase number of threads until performance drops

cache size
• pointer-chase on array, measure size of array when performance drops (see Example on far-right)

access latency
• pointer-chase on array, measure time-per-iteration when all elements are missing in the cache

bandwidth
• sum elements in array
• scale loads issued until performance plateaus

sharing of caches between cores
• two threads pointer-chase on very large array
• performance dramatically worse if caches shared

cache-to-cache transfer latency
• load array on core #1
• read array on core #2
• compare execution time to a baseline (array located off-chip, then read by core #2)

Example
The following microbenchmark (cache_size) ascertains:
• cache size
• access latency

Methodology
• One thread performs a pointer chase on a randomly sorted array (to minimize prefetching effects).
• Measure cycles to complete fixed amount of work.
• Increase array size, and repeat.

Berkeley Parlab
CELIO@EECS.BERKELEY.EDU

Prometheus