Characterizing Multi-Core Processors Using Micro-benchmarks

Christopher Celio, Krste Asanović, Dave Patterson

What does the hardware really look like?

GOAL: Design and implement portable software micro-benchmarks to determine the important characteristics of the hardware.

- number of cache levels
- data cache sizes at each level in the cache hierarchy
- access time to each level in the cache hierarchy
  - unit-stride, cache-line stride, random stride
- request bandwidth at each level
  - single-thread, multi-thread
- maximum outstanding requests
- cache-to-cache transfer latency
- cache-to-cache transfer bandwidth
- peak attainable flops

Motivation & Useful Applications

More Informed Autotuning

- narrow the search space
- application may be restricted to a subset of the hardware (e.g., bandwidth/memory partitioning, VMs)

Determining Hardware Specs

- Measure unpublished specs
  - cache-to-cache transfer latency, bandwidth
- Producing realistic specs
  - e.g., published theoretical bandwidth numbers may not be sustainable

Guiding Software Development

- A better understanding of a given machine's bottlenecks can direct implementation choices

Verification of Simulators

- Micro-benchmarks can characterize existing machines to provide a calibration for simulator targets
- Useful for finding bugs that render incorrect simulation results

Challenges

1. **Complex interactions of the memory hierarchy can yield convoluted results.**

   virtual memory, TLBs
   memory pre-fetchers (stream and spatial pre-fetchers)
   adaptive cache replacement policies
   cache coherence protocols
   memory controller interactions

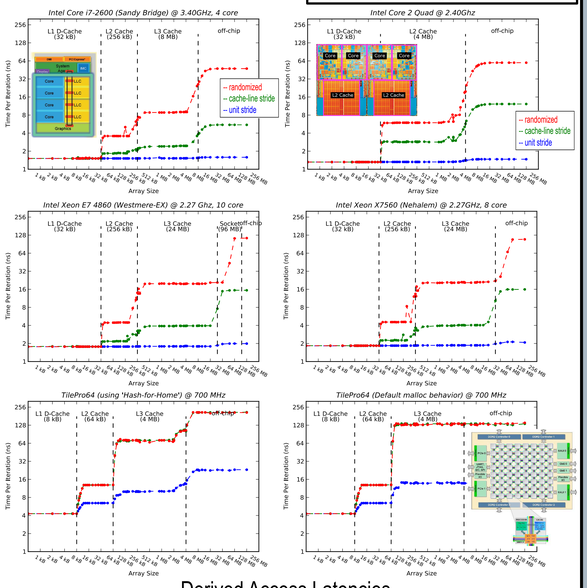
2. **Accurate measurements of runtime are limited by the precision of the system clock**
3. **Hyper-threading needs to be avoided for certain measurements (e.g., cache-to-cache latency)**
4. **Locking overhead**
5. **OS scheduling**
6. **Portability across platforms**

Cache Size & Access Latency

Micro-benchmark

- one thread performs a pointer chase on an array
  - unit stride (optimal locality)
  - 64 element stride (difference cache lines, avoids spatial prefetcher)
  - randomly sorted within a virtual page (avoids stream prefetching, spatial locality, maintains TLB locality)

```
start_time = get_seconds();

for (uint32_t k = 0; k < g_num_iterations; k++)
{
    idx = arr_n_ptr[idx];
}

stop_time = get_seconds();
```



Derived Access Latencies

Processor	access latency			
	L1	L2	L3	Off-chip
Intel i7 Sandy Bridge	1.52 ns 5.16 cycles	3.57 ns 12.16 cycles	8.82 ns 29.99 cycles	47.58 ns 161.78 cycles
Intel Core 2 Quad	1.32 ns 3.16 cycles	5.91 ns 14.19 cycles	- -	59.6 ns 143.0 cycles
Intel Xeon (Westmere-EX)	1.77 ns 4.0 cycles	4.47 ns 10.2 cycles	19.8 ns 44.9 cycles	115.5 ns 262.3 cycles
Intel Xeon (Nehalem-EX)	1.84 ns 4.17 cycles	4.55 ns 10.32 cycles	20.53 ns 46.61 cycles	109.0 ns 247.3 cycles
Tilera TILE64 (shared L3)	4.29 ns 3.00 cycles	12.9 ns 9.01 cycles	71.5 ns 50.1 cycles	202.7 ns 141.9 cycles
Tilera TILE64 (default)	4.29 ns 3.00 cycles	12.9 ns 9.01 cycles	135 ns 94.9 cycles	130.9 ns 91.6 cycles

* cycles are derived by multiplying time by the processor frequency

Request Bandwidth (single thread)

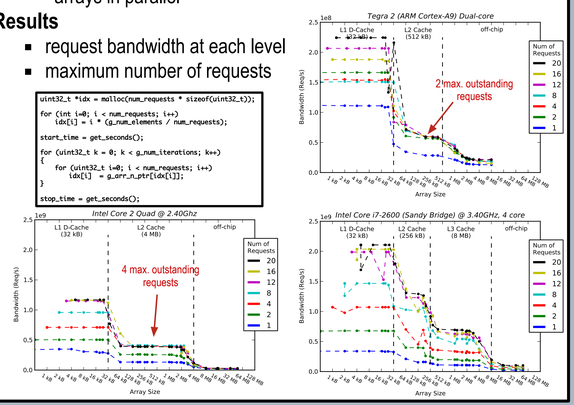
Micro-benchmark

- one thread performs a pointer chase on multiple, randomly sorted arrays in parallel

Results

- request bandwidth at each level
- maximum number of requests

```
uint32_t *idx = malloc(num_requests * sizeof(uint32_t));

for (int i=0; i < num_requests; i++)
    idx[i] = i * (g_num_elements / num_requests);

start_time = get_seconds();

for (uint32_t k = 0; k < g_num_iterations; k++)
{
    for (uint32_t i=0; i < num_requests; i++)
        idx[i]  = g_arr_n_ptr[idx[i]];
}

stop_time = get_seconds();
```



Inter-core Bandwidth

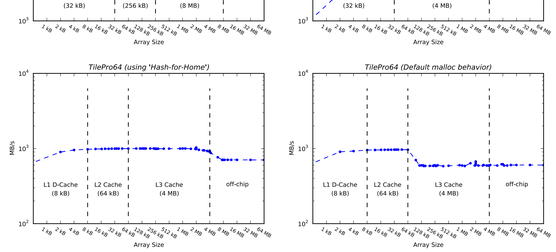
Micro-benchmark

- two threads ping pong an array back and forth (unit-stride)
- two threads ping pong an array back and forth
  - unit stride (optimal locality)
  - 64 element stride (difference cache lines, avoids spatial prefetcher)
  - randomly sorted within a virtual page (avoids stream prefetching, spatial locality, maintains TLB locality)

Results

- L1 to L1 bandwidth is on par with off-chip bandwidth
- LLC to L1 is typically highest bandwidth

```
//initialization code...
for(i=0; i<num_elements-1; i++)
    array[i] = i + 1;
array[num_elements -1] = 0;

//after initialization…
tid = pthread_self();
double start_time = get_seconds();

while(count < num_iters){
    if(tid == waiting_thread)
        while(*mylock == 0);

    *otherLock = 0;
    count++;
    int idx = 0;
    int i=0;
    __sync();

    for (int i=0; I < num_elements; i++)
    {
        array[i] = count;
    }

    __sync();
    waiting_thread = tid;
    *mylock = 0;
    *otherLock = 1;
}

double stop_time = get_seconds();
```



Total Processor Request Bandwidth

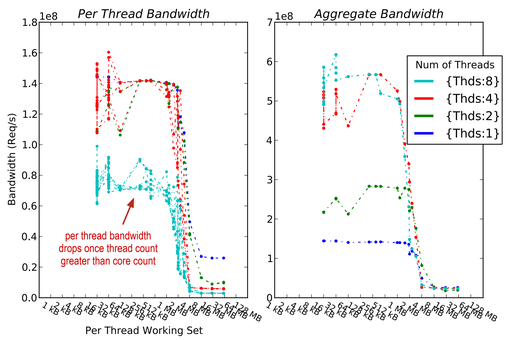
Micro-benchmark

- multiple threads perform a pointer chase on multiple arrays
  - each thread performs 64 independent memory loads in parallel

Results

- Aggregate off-chip bandwidth is saturated by a single thread (with enough instruction-level parallelism)

Core 2 Quad: Multi-thread Request Bandwidth



Auto-Roofline Models

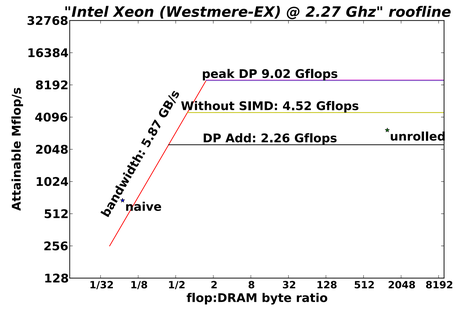
Problem: Roofline models typically require knowledge of the algorithm and its memory behavior on a given platform, as well as some details of the machine's pipeline.

GOAL: Use a mixture of micro-benchmarks and native hardware counters to construct roofline models.

Example: Matrix multiply

- naive implementation
- blocked implementation, with tranposed A matrix, padded arrays, and unrolled loops

Issues: Difficulty measuring off-chip traffic (prefetching traffic, write-back traffic, etc.)



Roofline model is entirely generated from micro-benchmarks.
The "matrix multiply" application flop/byte performance is gathered from native events thru PAPI.

Related Work

STREAM

- Sustainable Memory Bandwidth
- Measures unit-stride access bandwidth from off-chip

Roofline

- Visual representation of realistic performance and productivity expectations
- Determines hardware limitations for a specific kernel
- Prioritizes optimizations by showing potential benefits
- Uses STREAM to generate maximum FLOPS/Byte asymptote

MEMBENCH

- Single core focused
- measures 33 different transfer types
- main memory read/write

References

MEMBench. http://www.intelligentfirm.com/membench/membench.html

McCalpin, John D., 1995: "Memory Bandwidth and Machine Balance in Current High Performance Computers", IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter, December 1995.

S. Williams, A. Waterman, D. Patterson, "Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures", Communications of the ACM (CACM), April 2009.

CELIO@EECS.BERKELEY.EDU