Cache and TLB-aware Parallel Sorting

Kynan Shook

Sorting

- Sorting is used in many places
- Easy to understand, but hard to do well
- Many algorithms exist for different situations
- SMPs and CMPs have low communication cost
  - But a cache or TLB miss can be expensive

Techniques

- Baseline is a sorting program from last year
- Profile the code
- Modify the algorithms to improve locality
- Port from Objective-C to C, and test on other platforms

Radix Sort 101

- Look at one "digit" at a time (8 bits)
- Parallel radix sort uses counting sort locally
  - Counts frequency of each digit in parallel
  - Computes offset in destination array for each digit
  - Shuffles data; read in order, write to one of 256 locations

Improving Locality #1

- Only count a subset of digits at once
- Reduce random access
  - Array of counts
  - Destinations in data array
- Requires looping through input more times

Segmented Sorting

- Result was a significant slowdown
  - Buckets easily fit in cache
  - Improves write locality slightly
  - Significantly increases amount of work

Reducing the Radix

- Similar to previous technique
  - Count array is smaller
  - Possible destinations are fewer
  - More iterations are required
- Requires barriers between iterations
- Reduces memory used

Improving Locality #2

- Data shuffle phase is hard on cache and TLB
  - Moving data takes 3x to 16x longer than counting sort phase
  - IPC while counting is 3x to 13x higher than while moving data
- Try bucket sort instead of counting sort

Bucket Sort

- Bucket sort divides data into buckets
- Result is a concatenation of all buckets
- Still requires a random write
- Overhead turns out to be higher
  - Extra copy to move data back to array
  - Buckets need to be resized dynamically

Improving Locality #3

- Single-threaded radix sort can count all digits in first iteration
- Multi-threaded radix sort must wait at least until shuffle step
- Increment destination's count array while shuffling
  - Increases the working set size

Generic Optimizations

- Avoiding globals
  - Make a new local copy after global has changed
- Using appropriate locking libraries
- Reducing library calls

Results

- 4 times fewer TLB misses
  - 94% are while sorting; originally 60%
- L2 total misses are 10% lower
  - Miss rate is 40% higher
- L1 total misses are 20% lower
  - Miss rate is 30% higher

Port from Objective-C

- Replace Cocoa Threads with pthreads
- Replace NSConditionLock with pthread mutex and pthread condition variable
- Otherwise, very similar
  - Objective-C is a strict superset of C
  - Original code didn't have many objects, so there were few changes to make

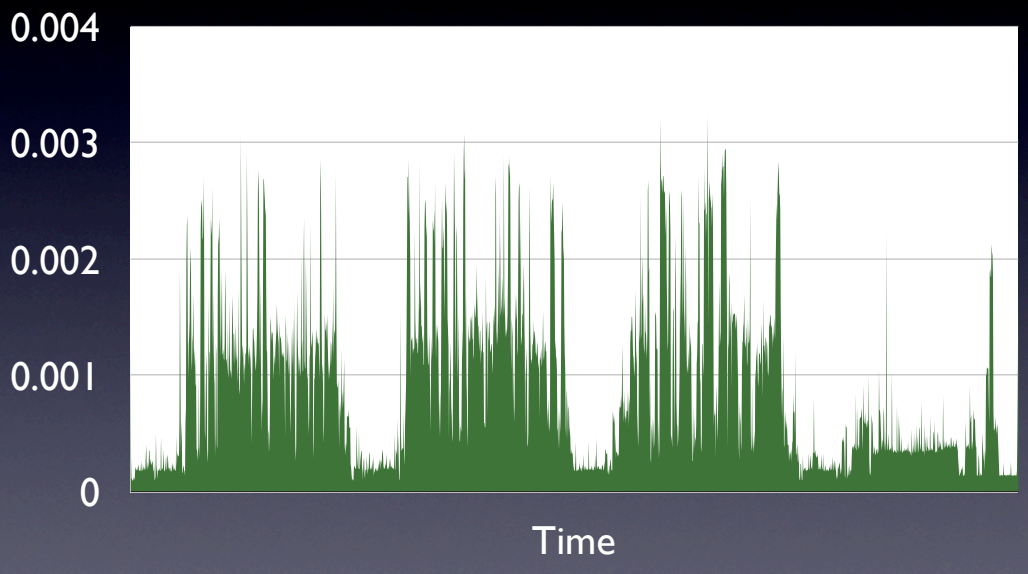
Objective-C versus C

- Surprisingly, Objective-C version ran faster than C version
- Code is nearly identical
- C version has higher TLB miss rate

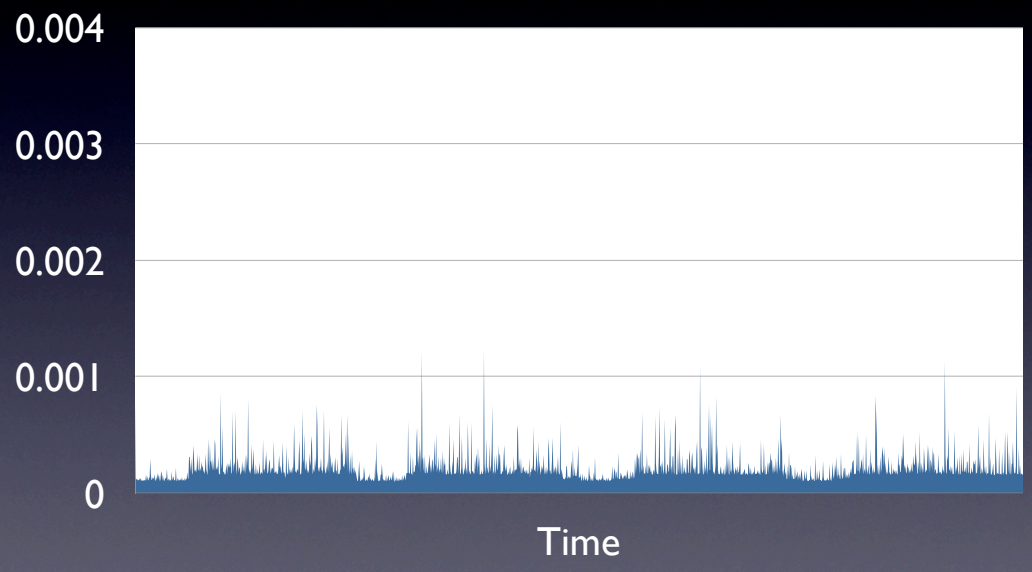
Best Performance

- Chianti: 1.88 seconds, 13.6x parallel speedup (32 threads)
- Clover: 2.49 seconds, 5.6x parallel speedup (8 threads)
- Dual PowerPC G5: 1.74 seconds, 1.3x parallel speedup (2 threads)

Initial TLB miss rate



Final TLB miss rate



Conclusions

- Can significantly improve TLB and cache performance without modifying algorithms
- Profiling different events can yield a wide variety of information
- It can be hard to judge cause and effect on real hardware