Cache Memories

CS:APP 6.5 intro

These slides adapted from materials provided by the textbook authors.

The Memory Mountain

- **Read throughput** (read bandwidth)
  - Number of bytes read from memory per second (MB/s)

- **Memory mountain:** Measured read throughput as a function of spatial and temporal locality.
  - Compact way to characterize memory system performance.

Memory Mountain Test Function

```c
long data[MAXELEMS];  /* Global array to traverse */

/* test – Iterate over first "elems" elements of
 *        array "data" with stride of "stride", using
 *        using 4x4 loop unrolling.
 */
int test(int elems, int stride) {
    long i, sx2=stride*2, sx3=stride*3, sx4=stride*4;
    long acc0 = 0, acc1 = 0, acc2 = 0, acc3 = 0;
    long length = elems, limit = length – sx4;

    /* Combine 4 elements at a time */
    for (i = 0; i < limit; i += sx4) {
        acc0 = acc0 + data[i];
        acc1 = acc1 + data[i+stride];
        acc2 = acc2 + data[i+sx2];
        acc3 = acc3 + data[i+sx3];
    }

    /* Finish any remaining elements */
    for (; i < length; i++) {
        acc0 = acc0 + data[i];
    }
    return ((acc0 + acc1) + (acc2 + acc3));
}
```

mountain/mountain.c

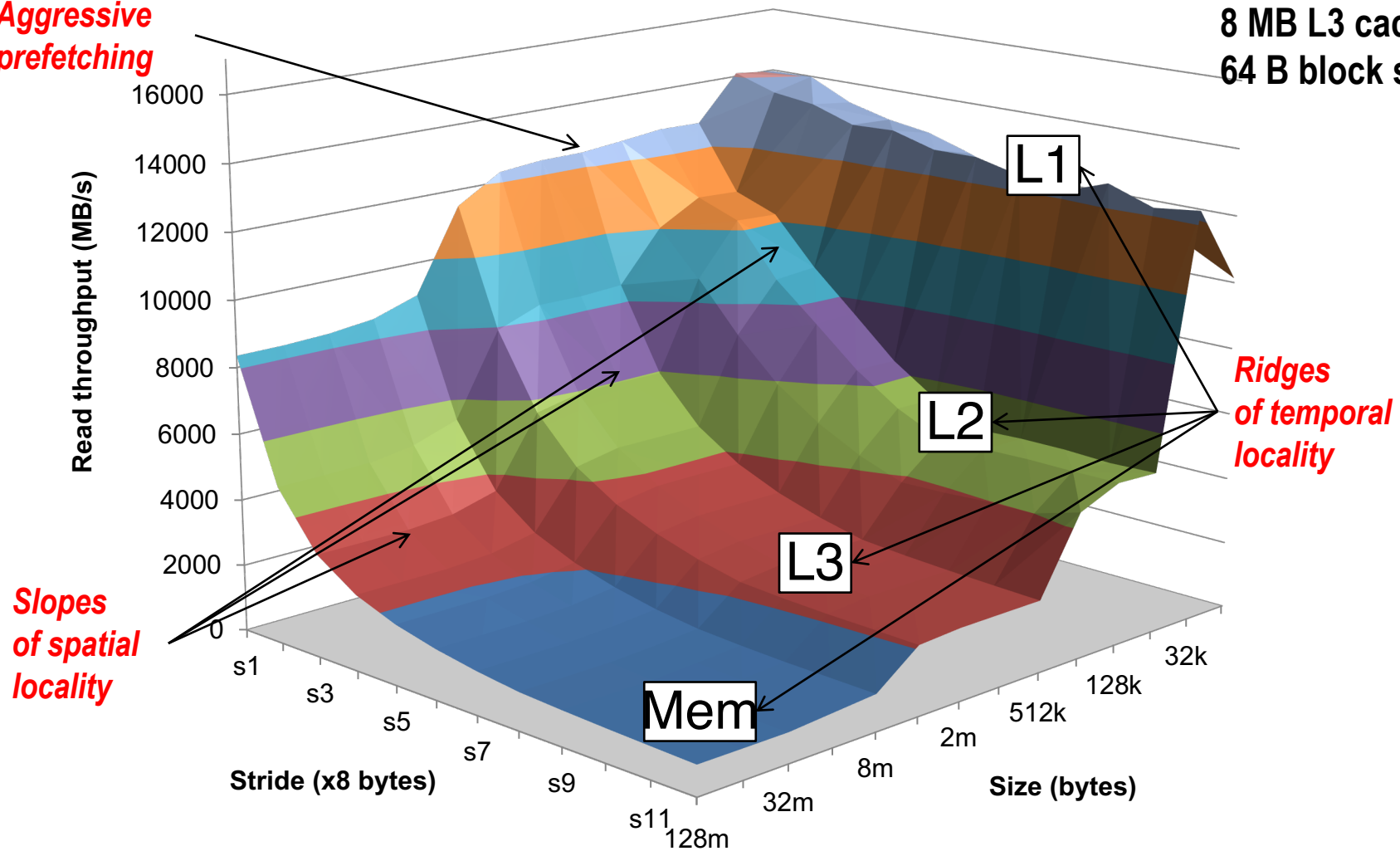Call `test()` with many combinations of `elems` and `stride`.

For each elems and stride:

1. Call test() once to warm up the caches.

2. Call test() again and measure the read throughput (MB/s)

The Memory Mountain

Core i7 Haswell
2.1 GHz
32 KB L1 d-cache
256 KB L2 cache
8 MB L3 cache
64 B block size

Aggressive prefetching

Ridges of temporal locality

Slopes of spatial locality

Read throughput (MB/s)

16000
14000
12000
10000
8000
6000
4000
2000
0

L1
L2
L3
Mem

Stride (x8 bytes)

s1 s3 s5 s7 s9 s11 128m

Size (bytes)

32k 128k 512k 2m 8m 32m

Prefetching

- **Algorithms in processors predict access patterns**
- **Try to *pre-fetch* memory**
  - Based on strides
  - Future: based on contents

Stride Detection

$\Delta = 2$

Prefetch Starts
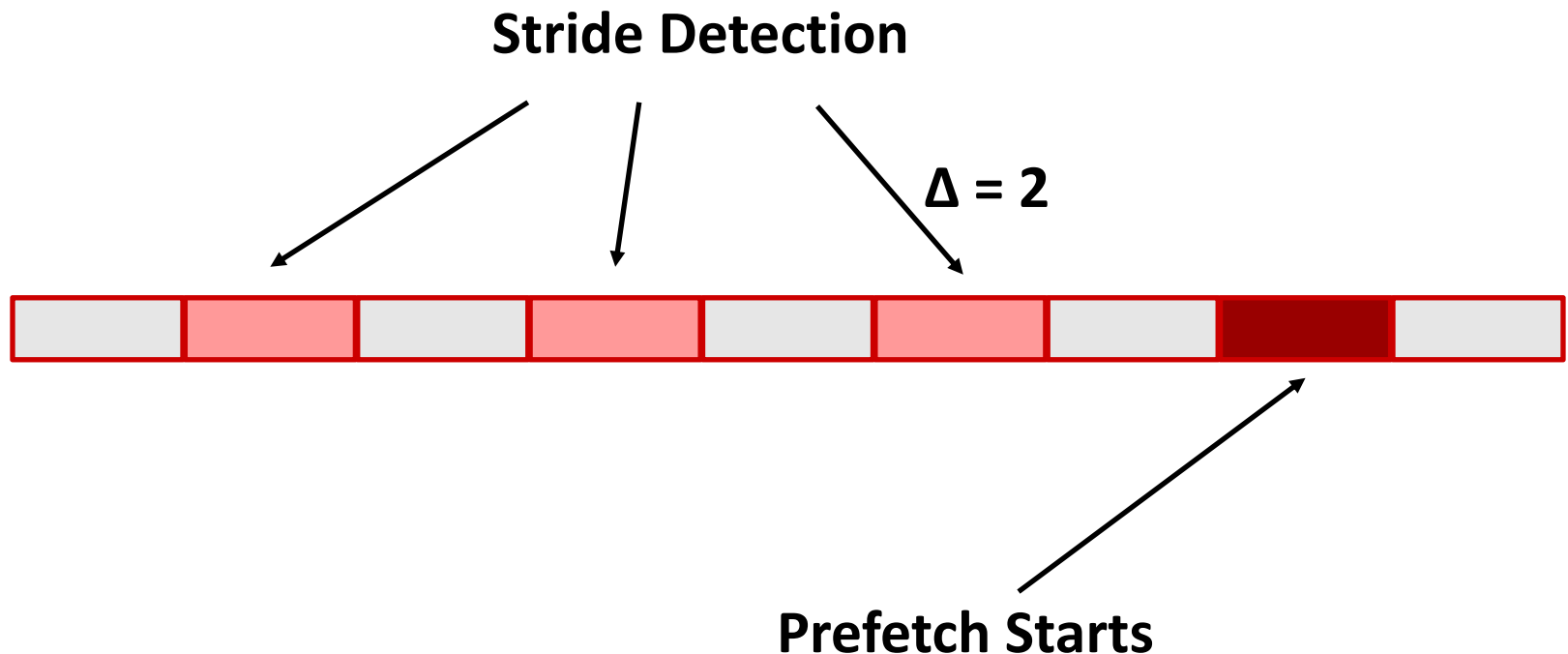
Cache Summary

- **Cache memories can have significant performance impact**

- **You can write your programs to exploit this!**
    - Focus on the inner loops, where bulk of computations and memory accesses occur.
    - Try to maximize spatial locality by reading data objects with sequentially with stride 1.
    - Try to maximize temporal locality by using a data object as often as possible once it's read from memory.