Writing Cache Friendly Code
Writing Cache Friendly Code

- Make the common case go fast
  - Focus on the inner loops of the core functions

- Minimize the misses in the inner loops
  - Repeated references to variables are good (temporal locality)
  - Stride-1 reference patterns are good (spatial locality)

Key idea: Our qualitative notion of locality is quantified through our understanding of cache memories.
Today

- Performance impact of caches
  - The memory mountain
  - Rearranging loops to improve spatial locality
  - Using tiling to improve temporal locality
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.
/** The test function */
void test(int elems, int stride) {
    int i, result = 0;
    volatile int sink;

    for (i = 0; i < elems; i += stride)
        result += data[i];
    sink = result; /* So compiler doesn't optimize away the loop */
}

/* Run test(elems, stride) and return read throughput (MB/s) */
double run(int size, int stride, double Mhz) {
    double cycles;
    int elems = size / sizeof(int);

    test(elems, stride); /* warm up the cache */
    cycles = fcyc2(test, elems, stride, 0); /* call test(elems,stride) */
    return (size / stride) / (cycles / Mhz); /* convert cycles to MB/s */
}
The Memory Mountain

Read throughput (MB/s)

Stride (x8 bytes)

Working set size (bytes)

Intel Core i7
32 KB L1 i-cache
32 KB L1 d-cache
256 KB unified L2 cache
8M unified L3 cache
All caches on-chip
The Memory Mountain

Slopes of spatial locality

Read throughput (MB/s)

Stride (x8 bytes)

Working set size (bytes)

Intel Core i7
32 KB L1 i-cache
32 KB L1 d-cache
256 KB unified L2 cache
8M unified L3 cache
All caches on-chip
The Memory Mountain

- Intel Core i7
- 32 KB L1 i-cache
- 32 KB L1 d-cache
- 256 KB unified L2 cache
- 8M unified L3 cache
- All caches on-chip

Ridges of Temporal locality
Slopes of spatial locality
Warm-up example

- Description:
  - 1 array of length n
  - Walk array m times

- Assumptions
  - 32B cache blocks (fit **four**, 64-bit (8B) doubles)
  - n is large (array much bigger than cache)
  - m is large (approximate 1/m as 0)

- Average miss rate?
  - A) 0    B) 1/8    C) 1/4    D) 1/2    E) 1

```c
double sum = 0.0;
double a[n];
for (i = 0; i < m; i ++) {
    for (j = 0; j < n; j ++) {
        sum += a[j];
    }
}
```
Loop inversion swaps the indexing variable of the inner loop (j i indexes the inner loop)

- Same Assumptions
  - 32B cache blocks (fit four, 64-bit (8B) doubles)
  - n is large (array much bigger than cache)
  - m is large (approximate 1/m as 0)

- Average miss rate?
  - A) 0    B) 1/8    C) 1/4    D) 1/2    E) 1

```c
double sum = 0.0;
double a[n];
for (j = 0; j < n; j ++) {
    for (i = 0; i < n; i ++) {
        sum += a[j];
    }
}
```