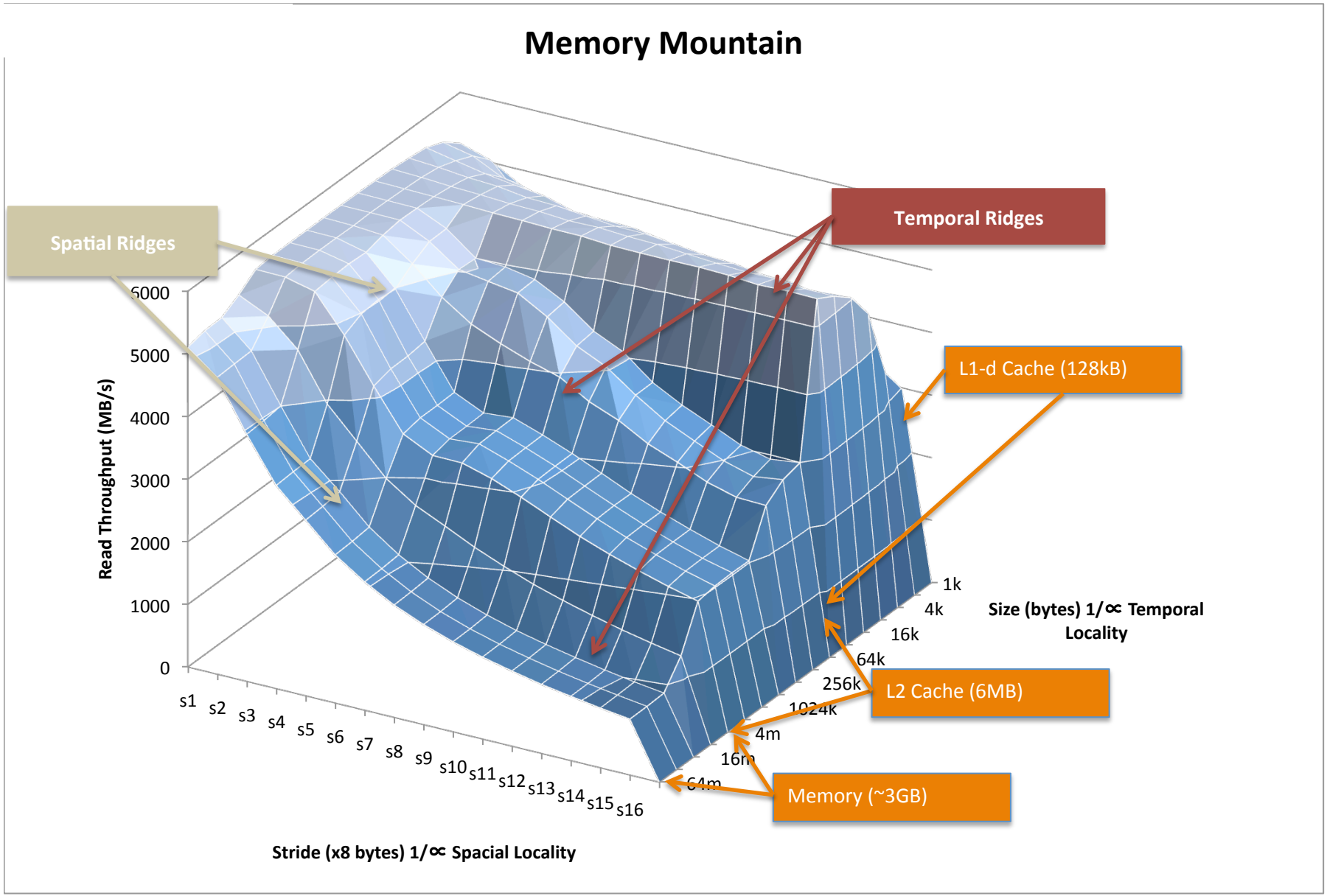
Memory Mountain
Spatial Ridges
Temporal Ridges
L1-d Cache (128kB)
L2 Cache (6MB)
Memory (~3GB)
Read Throughput (MB/s)
6000
5000
4000
3000
2000
1000
0
s1
s2
s3
s4
s5
s6
s7
s8
s9
s10
s11
s12
s13
s14
s15
s16
Stride (x8 bytes) 1/∝ Spacial Locality
1k
4k
16k
64k
256k
1024k
4m
16m
64m
Size (bytes) 1/∝ Temporal Locality

Relevant information:

CPU:

- Name and version: Intel Core 2 Duo E8500
- Clock Speed: 3.16 GHz
- Two cores
- “Wolfdale” Architecture
- Caches:
  - L1-d and L1-i: 128kB
  - L2: 6MB, unified
- Front Bus Speed: 1333 MHz
- 64-bit Instruction set that is compatible with 32-bit IS and 16-bit IS
- Address Size: 36-bits physical, 48-bits virtual
-
- Information Sources:
  - “cat /proc/cpuinfo”
  - http://hothardware.com/Reviews/Wolfdale/
  - http://ark.intel.com/products/33911/Intel-Core2-Duo-Processor-E8500-6M-Cache-3_16-GHz-1333-MHz-FSB

Memory (RAM):

- Size: 3 GB
- Source
  - “cat /proc/cpuinfo”

- **Chart Analysis:**

We can see three peaks or ridges when we view the Read Througput crossed with the size axis. These peaks reveal how temporal locality impacts locality in differnet levels of caches. The L1-d cache peaks, as my research would suggest, at its limit, 128kBs with speed 6GB/s. This ridge or peak suggests that there are practically no misses in that data set size, since all of array elements or data chunks load to L1 cache. The lowest point on the L1 cache hill is when we start loading the data (1 kB). The book argues that it has this low because setting up the code, “overheading” costs this performance. The peaks in the rigdges similarly happen for L2 cache: they illustrate temporal locality, and the fastest clocking happens when we start loading all the data we can to L1 and L2 cache, (128k to 6MB) @ 6GB/s when stride=1 and 2-3 GB/s as the stride increases. There is something interesting in the L2 cache geography. There is a drop of clocking performance around 512k when the stride is larger than 2. This may be because other users logged in to the same machine and used up some of the resources while I ran the test over three times.
Things level out in memory. Things are slow.

The peaks or ridges when we view the Read Throughput crossed with stride illustrate spacial locality. The smaller the stride, the faster the system reads. In the previous paragraph I showed how spacial and temporal locality can save each other—having larger stides that are read sequentially can make up for having not reading the same chunk (temporal locality) more than once.

My information on the RAM is limited. I know its size and can deduce its largest clock speed from the Bus Speed of the CPU, which is 1333 MHz. If we assume the motherboard handles 1333 MHz of bus speed, we can assume that this is how fast the RAM clocks (with, let's say, 100 cycle cost). The Ram is relatively slow compared to L1 and L2 clocking. The graph illustrates how slow the memory is compared to the caches—1 GB/s compared to 6 GB/s for L1-d and ~5GB/s to ~2.5 GB/s for L2.

The lesson then is, cache speeds up data access by a factor greater than 2, and helps us utilize a CPU. And spatial and temporal locality optimize accesses. It's far more wasteful to thrash due to cache-unfriendly code than it is to spend time thinking on how to optimize our code to get the most out of our system and the energy it consumes.