Student: Minh Pham
EID: mlp2279
CSID: minhpham

Computer Analyzed:
charity.cs.utexas.edu

Memory Throughput Graph:
Please see next page.

CPU:
Intel Xeon X3460 - Quad Cores
L1 cache size: 25kB instruction + 25kb data per core
L2 cache size: 256kB unified cache per core
L3 cache size: 8MB unified cache shared among 4 cores

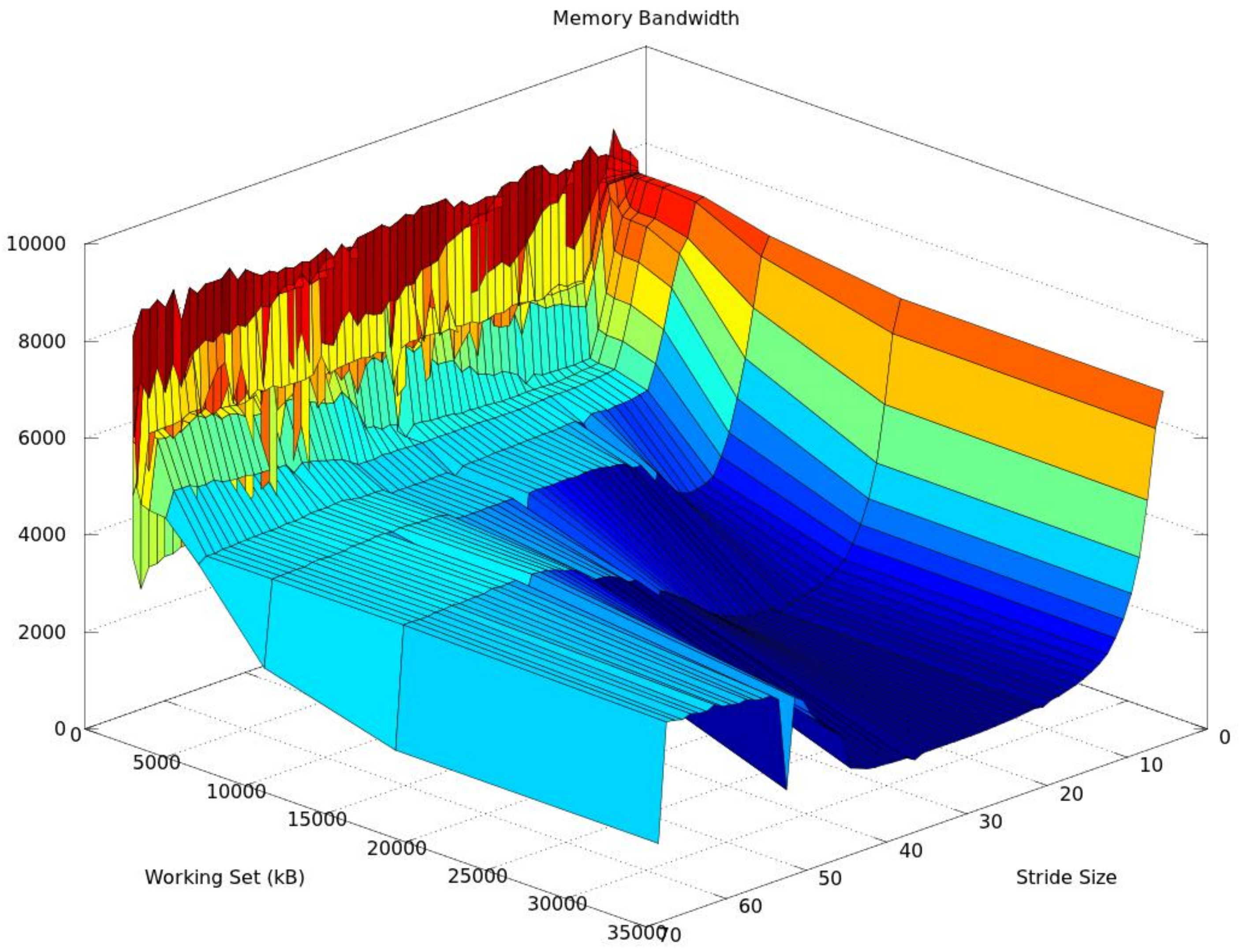
(from Datasheet for X3400 Series, link:
http://www.intel.com/Assets/PDF/datasheet/322371.pdf?wapkw=x3460+datasheet)

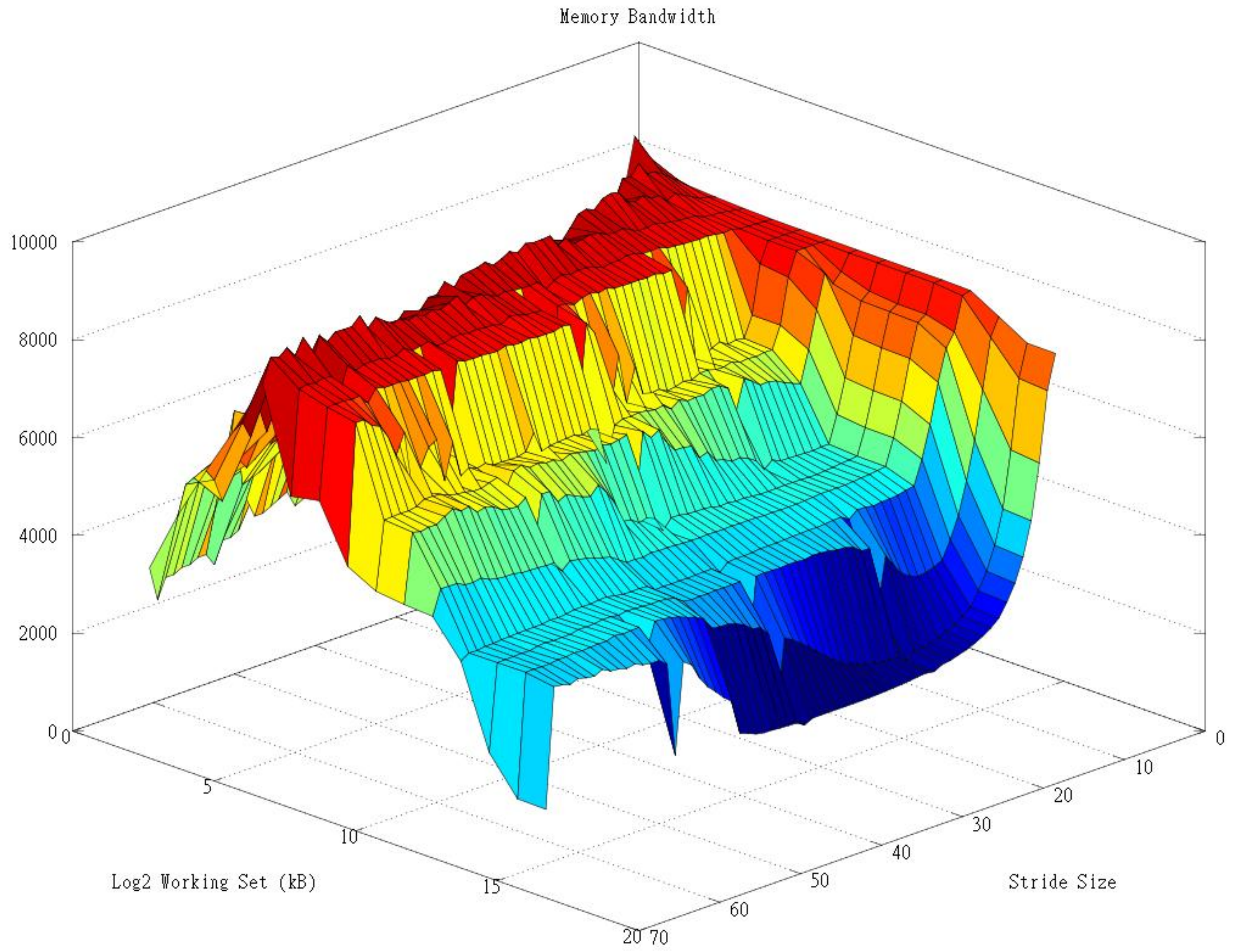
Memory Behavior:

We expected memory throughput to decrease with decreases in temporal and spatial locality. Sharp declines (ridges) should appear near cache boundaries as working set size increases. Smoother declines (slopes) should form as stride size increases.

What is observed generally follows this ideal case, with one major anomaly: as the working set increases, the optimum stride size seems to increase as well. The whole graph, looking from the top, appears to be "rotated" a certain angle from the "ideal" graph. I do not understand how this comes about. My conjecture is that it has something to do with the system having 4 cores: when the working set exceeds the L1 cache of one core, the memory "spills over" and is loaded into the L1 cache of the 2nd core, and so on. The optimum stride size would then increase because if it's too low it will just cause a cache miss in the first core without getting a hit in the 2nd core. Of course, this has a lot to do with how addresses are mapped onto the cache. This information I could not find in the Datasheet I have.

Beside the major anomaly, we also have a minor anomaly: cache behavior becomes very bad at stride size a multiple of 16: 16, 32, and 48, forming what looks like fault lines. The "fault lines" also gets worse the bigger the stride size gets. This is probably due to the block size of the CPU. According to the datasheet, each of the 4 Intel Xeon X3460 cores has 2 cache lines of 64 bits each. The block size is therefore 128 bytes. Since we are traversing arrays of doubles, each element is 8 bytes in size. A stride size of 16 doubles is equivalent to 128 bytes. At this stride size we will fill up the whole cache and then miss at every request. The reason it seems to get worse is related to the major anomaly above. Each "fault line" brings memory through put down to about the same level. However, due to the graph being "rotated", at large working set size (closer from our perspective) the performance at the large stride size is actually better than the performance at small stride size. The difference, therefore, looks greater. However, the performance at each "fault line" is really about the same. It's a matter of perspective.


Memory Bandwidth
10000
8000
6000
4000
2000
0
0
5000
10000
15000
20000
25000
30000
35000
Working Set (kB)
0
10
20
30
40
50
60
70
Stride Size


Memory Bandwidth
10000
8000
6000
4000
2000
0
0
5
10
15
20
Log2 Working Set (kB)
0
10
20
30
40
50
60
70
Stride Size