Lab Report

ECPE 170 – Computer Systems and Networks – Spring 2017

Name: STEVE GUERRERO

Lab Topic: MEMORY OPTIMIZATION (LAB #: 07)

(1) Describe how a two-dimensional array is stored in one-dimensional computer memory.
The code I wrote displayed the column first from end to end then moved down to the next row. The memory addresses, which were in hexadecimal form, coincided with the display order.

(2) Describe how a three-dimensional array is stored in one-dimensional computer memory.
The order goes through the table (3rd dimension), then columns, then on to the next row.. Pretty much in the reverse order of declaration.
Int array[row][col][table];

(3) Copy and paste the output of your program into your lab report, and be sure that the source code and Makefile is included in your Mercurial repository.

```
2-D array addresses in order                    uint32_t array1[3][5];          3 rows   5 columns

array1[0][0] address->  0x7fffc0391e40
array1[0][1] address->  0x7fffc0391e44
array1[0][2] address->  0x7fffc0391e48
array1[0][3] address->  0x7fffc0391e4c
array1[0][4] address->  0x7fffc0391e50
array1[1][0] address->  0x7fffc0391e54
array1[1][1] address->  0x7fffc0391e58
array1[1][2] address->  0x7fffc0391e5c
array1[1][3] address->  0x7fffc0391e60
array1[1][4] address->  0x7fffc0391e64
array1[2][0] address->  0x7fffc0391e68
array1[2][1] address->  0x7fffc0391e6c
array1[2][2] address->  0x7fffc0391e70
array1[2][3] address->  0x7fffc0391e74
array1[2][4] address->  0x7fffc0391e78


3-D array addresses in order                    uint32_t array2[3][5][7];    3 rows   5 columns 7 tables

array2[0][0][0] address->  0x7fffc0391e80
array2[0][0][1] address->  0x7fffc0391e9c
array2[0][0][2] address->  0x7fffc0391eb8
array2[0][0][3] address->  0x7fffc0391ed4
array2[0][0][4] address->  0x7fffc0391ef0
array2[0][0][5] address->  0x7fffc0391f0c
array2[0][0][6] address->  0x7fffc0391f28
array2[0][1][0] address->  0x7fffc0391e84
array2[0][1][1] address->  0x7fffc0391ea0
array2[0][1][2] address->  0x7fffc0391ebc
array2[0][1][3] address->  0x7fffc0391ed8
array2[0][1][4] address->  0x7fffc0391ef4
array2[0][1][5] address->  0x7fffc0391f10
array2[0][1][6] address->  0x7fffc0391f2c
array2[0][2][0] address->  0x7fffc0391e88
array2[0][2][1] address->  0x7fffc0391ea4
array2[0][2][2] address->  0x7fffc0391ec0
array2[0][2][3] address->  0x7fffc0391edc
array2[0][2][4] address->  0x7fffc0391ef8
array2[0][2][5] address->  0x7fffc0391f14
array2[0][2][6] address->  0x7fffc0391f30
array2[0][3][0] address->  0x7fffc0391e8c
array2[0][3][1] address->  0x7fffc0391ea8
array2[0][3][2] address->  0x7fffc0391ec4
array2[0][3][3] address->  0x7fffc0391ee0
array2[0][3][4] address->  0x7fffc0391efc
array2[0][3][5] address->  0x7fffc0391f18
array2[0][3][6] address->  0x7fffc0391f34
array2[0][4][0] address->  0x7fffc0391e90
array2[0][4][1] address->  0x7fffc0391eac
array2[0][4][2] address->  0x7fffc0391ec8
array2[0][4][3] address->  0x7fffc0391ee4
array2[0][4][4] address->  0x7fffc0391f00
array2[0][4][5] address->  0x7fffc0391f1c
array2[0][4][6] address->  0x7fffc0391f38
array2[1][0][0] address->  0x7fffc0391f0c
```

```
array2[1][0][1] address->  0x7fffc0391f28
array2[1][0][2] address->  0x7fffc0391f44
array2[1][0][3] address->  0x7fffc0391f60
array2[1][0][4] address->  0x7fffc0391f7c
array2[1][0][5] address->  0x7fffc0391f98
array2[1][0][6] address->  0x7fffc0391fb4
array2[1][1][0] address->  0x7fffc0391f10
array2[1][1][1] address->  0x7fffc0391f2c
array2[1][1][2] address->  0x7fffc0391f48
array2[1][1][3] address->  0x7fffc0391f64
array2[1][1][4] address->  0x7fffc0391f80
array2[1][1][5] address->  0x7fffc0391f9c
array2[1][1][6] address->  0x7fffc0391fb8
array2[1][2][0] address->  0x7fffc0391f14
array2[1][2][1] address->  0x7fffc0391f30
array2[1][2][2] address->  0x7fffc0391f4c
array2[1][2][3] address->  0x7fffc0391f68
array2[1][2][4] address->  0x7fffc0391f84
array2[1][2][5] address->  0x7fffc0391fa0
array2[1][2][6] address->  0x7fffc0391fbc
array2[1][3][0] address->  0x7fffc0391f18
array2[1][3][1] address->  0x7fffc0391f34
array2[1][3][2] address->  0x7fffc0391f50
array2[1][3][3] address->  0x7fffc0391f6c
array2[1][3][4] address->  0x7fffc0391f88
array2[1][3][5] address->  0x7fffc0391fa4
array2[1][3][6] address->  0x7fffc0391fc0
array2[1][4][0] address->  0x7fffc0391f1c
array2[1][4][1] address->  0x7fffc0391f38
array2[1][4][2] address->  0x7fffc0391f54
array2[1][4][3] address->  0x7fffc0391f70
array2[1][4][4] address->  0x7fffc0391f8c
array2[1][4][5] address->  0x7fffc0391fa8
array2[1][4][6] address->  0x7fffc0391fc4
array2[2][0][0] address->  0x7fffc0391f98
array2[2][0][1] address->  0x7fffc0391fb4
array2[2][0][2] address->  0x7fffc0391fd0
array2[2][0][3] address->  0x7fffc0391fec
array2[2][0][4] address->  0x7fffc0392008
array2[2][0][5] address->  0x7fffc0392024
array2[2][0][6] address->  0x7fffc0392040
array2[2][1][0] address->  0x7fffc0391f9c
array2[2][1][1] address->  0x7fffc0391fb8
array2[2][1][2] address->  0x7fffc0391fd4
array2[2][1][3] address->  0x7fffc0391ff0
array2[2][1][4] address->  0x7fffc039200c
array2[2][1][5] address->  0x7fffc0392028
array2[2][1][6] address->  0x7fffc0392044
array2[2][2][0] address->  0x7fffc0391fa0
array2[2][2][1] address->  0x7fffc0391fbc
array2[2][2][2] address->  0x7fffc0391fd8
array2[2][2][3] address->  0x7fffc0391ff4
array2[2][2][4] address->  0x7fffc0392010
array2[2][2][5] address->  0x7fffc039202c
array2[2][2][6] address->  0x7fffc0392048
array2[2][3][0] address->  0x7fffc0391fa4
array2[2][3][1] address->  0x7fffc0391fc0
array2[2][3][2] address->  0x7fffc0391fdc
```

```
array2[2][3][3] address->  0x7fffc0391ff8
array2[2][3][4] address->  0x7fffc0392014
array2[2][3][5] address->  0x7fffc0392030
array2[2][3][6] address->  0x7fffc039204c
array2[2][4][0] address->  0x7fffc0391fa8
array2[2][4][1] address->  0x7fffc0391fc4
array2[2][4][2] address->  0x7fffc0391fe0
array2[2][4][3] address->  0x7fffc0391ffc
array2[2][4][4] address->  0x7fffc0392018
array2[2][4][5] address->  0x7fffc0392034
array2[2][4][6] address->  0x7fffc0392050
```

(4) Provide an Access Pattern table for the sumarrayrows() function assuming ROWS=2 and COLS=3.
The table should be sorted by ascending memory addresses, not by program access order.

Memory Address	0	4	8	12	16	20
Memory Contents	a[0][0]	a[0][1]	a[0][2]	a[1][0]	a[1][1]	a[1][2]
Program Access Order	1	2	3	4	5	6

(5) Does sumarrayrows() have good temporal or spatial locality?
For your answer to receive full credit, you must discuss the locality of both the array itself, and the scalar variables such as i that are present in the function.
The sumarrayrows() function appears to have good spatial locality because it transcends through the columns in the order of ascending memory addresses. So it locates the data in the neighboring memory address. The i and j variables are examples of good temporal locality in that they are being constantly utilized for each memory address.

(6) Provide an Access Pattern table for the sumarraycols() function assuming ROWS=2 and COLS=3.
The table should be sorted by ascending memory addresses, not by program access order.

Memory Address	0	4	8	12	16	20
Memory Contents	a[0][0]	a[1][0]	a[0][1]	a[1][1]	a[0][2]	a[1][2]
Program Access Order	1	2	3	4	5	6

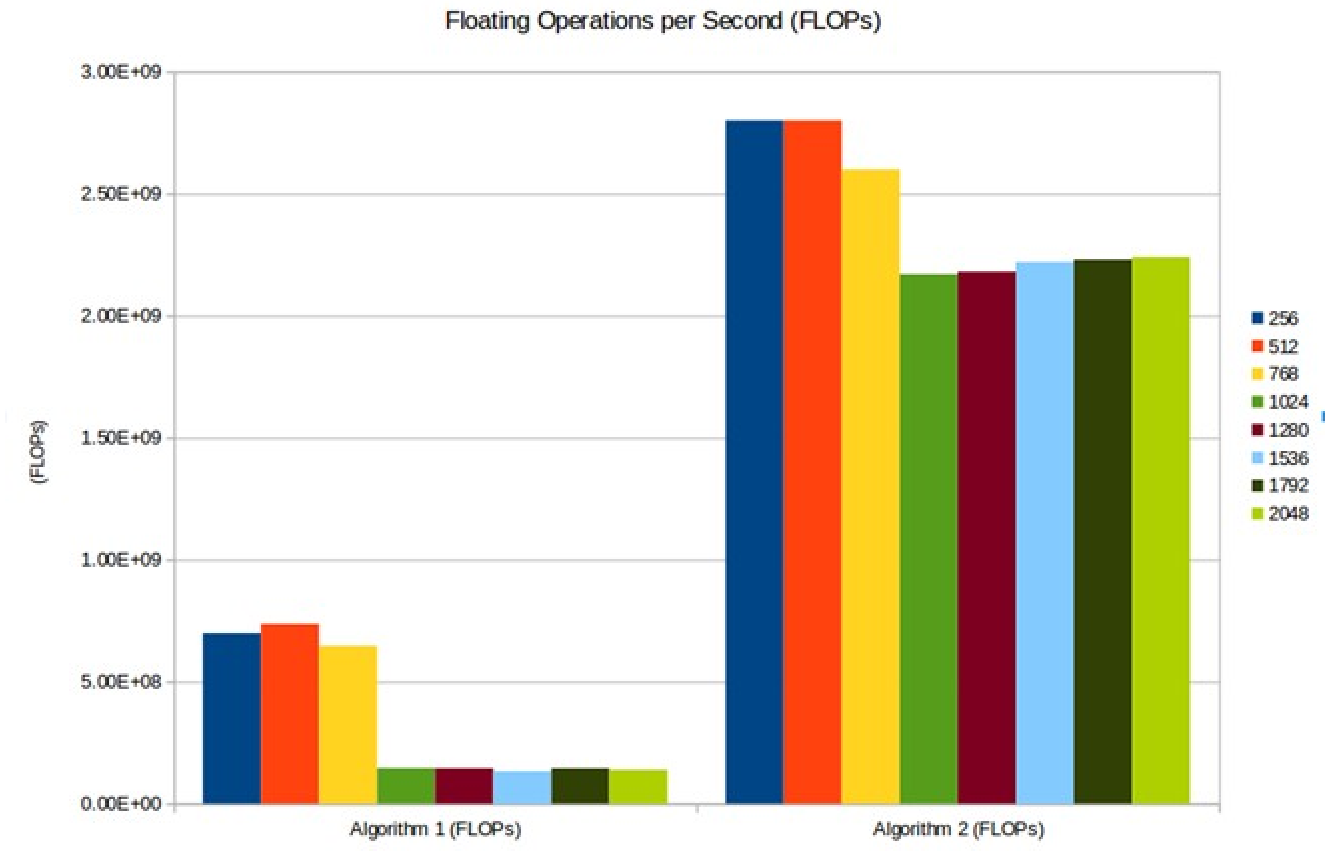
(7) Does sumarraycols() have good temporal or spatial locality?
For your answer to receive full credit, you must discuss the locality of both the array itself, and the scalar variables such as i that are present in the function.
The sumarraycols() function appears to have poor spatial locality because it transcends through the rows then the columns which is out of order with the way the memory addresses are stored. The i and j variables are examples of good temporal locality in that they are being used and changed for each memory address.

(8) Inspect the provided source code. Describe how the two-dimensional arrays are stored in memory, since the code only has one-dimensional array accesses like: a[element #].
The size of the one dimensional array is squared to equate to the size of the 2-d array.

(9) After running your experiment script, create a table that shows floating point operations per second for both algorithms at the array sizes listed in Table 2.

Array Size	Algorithm 1 (FLOPs)	Algorithm 2 (FLOPs)
256	6.99E+08	2.80E+09
512	7.37E+08	2.80E+09
768	6.47E+08	2.60E+09
1024	1.46E+08	2.17E+09
1280	1.45E+08	2.18E+09
1536	1.33E+08	2.22E+09
1792	1.45E+08	2.23E+09
2048	1.39E+08	2.24E+09

(10) After running your experiment script, create a graph that shows floating point operations per second for both algorithms at the array sizes listed in Table 2.
Note: No credit will be given for sloppy graphs that lack X and Y axis labels, a legend, and a title.



(11) Be sure that the script source code is included in your Mercurial repository.
Script code is included.

(12) Place the output of /proc/cpuinfo in your report. (I only need to see one processor core, not all the cores as reported)

```
processor	: 1
vendor_id	: GenuineIntel
cpu family	: 6
model		: 60
model name	: Intel(R) Core(TM) i7-4702MQ CPU @ 2.20GHz
stepping	: 3
microcode	: 0x1a
cpu MHz		: 2194.930
cache size	: 6144 KB
physical id	: 2
siblings	: 1
core id		: 0
cpu cores	: 1
apicid		: 2
initial apicid	: 2
fpu		: yes
fpu_exception	: yes
cpuid level	: 13
wp		: yes
flags		: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts mmx fxsr sse sse2 ss syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts nopl xtopology tsc_reliable nonstop_tsc aperfmperf eagerfpu pni pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand hypervisor lahf_lm abm epb fsgsbase tsc_adjust bmi1 avx2 smep bmi2 invpcid xsaveopt dtherm ida arat pln pts
bugs		:
bogomips	: 4389.86
clflush size	: 64
cache_alignment	: 64
address sizes	: 42 bits physical, 48 bits virtual
power management:
```

(13) Based on the processor type reported, obtain the following specifications for your CPU from cpu-world.comor cpudb.stanford.edu
You might have to settle for a close processor from the same family. Make sure the frequency and L3 cache size match the results from /proc/cpuinfo!
(a) L1 instruction cache size
4 x 32 KB
(b) L1 data cache size
4 x 32 KB
(c) L2 cache size
4 x 256 KB
(d) L3 cache size
6 MB
(e) What URL did you obtain the above specifications from?
Cpu-world.com

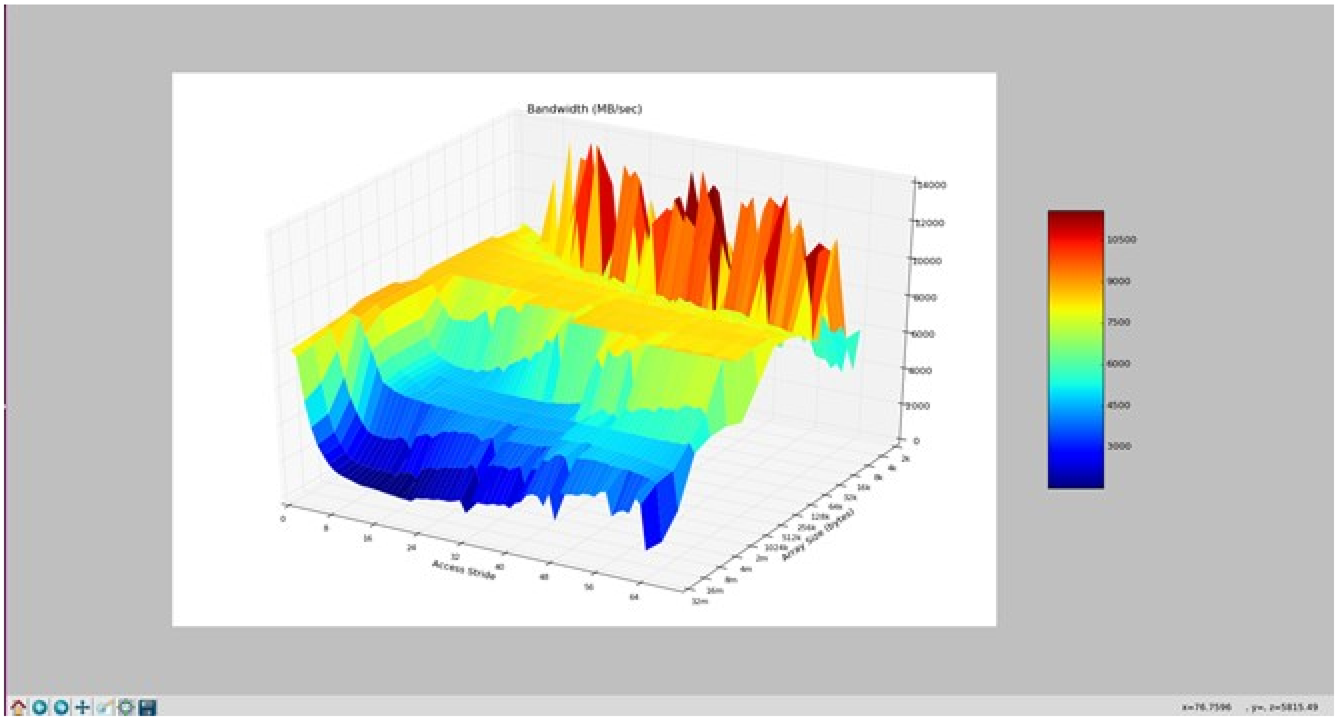
(14) Why is it important to run the test program on an idle computer system?
Explain what would happen if the computer was running several other active programs in the background at the same time, and how that would impact the test results.
By running the program on an idle system you're allowing the program to utilize more bandwidth rather than sharing it with other running applications.

(15) What is the size (in bytes) of a data element read from the array in the test?
32 bytes.

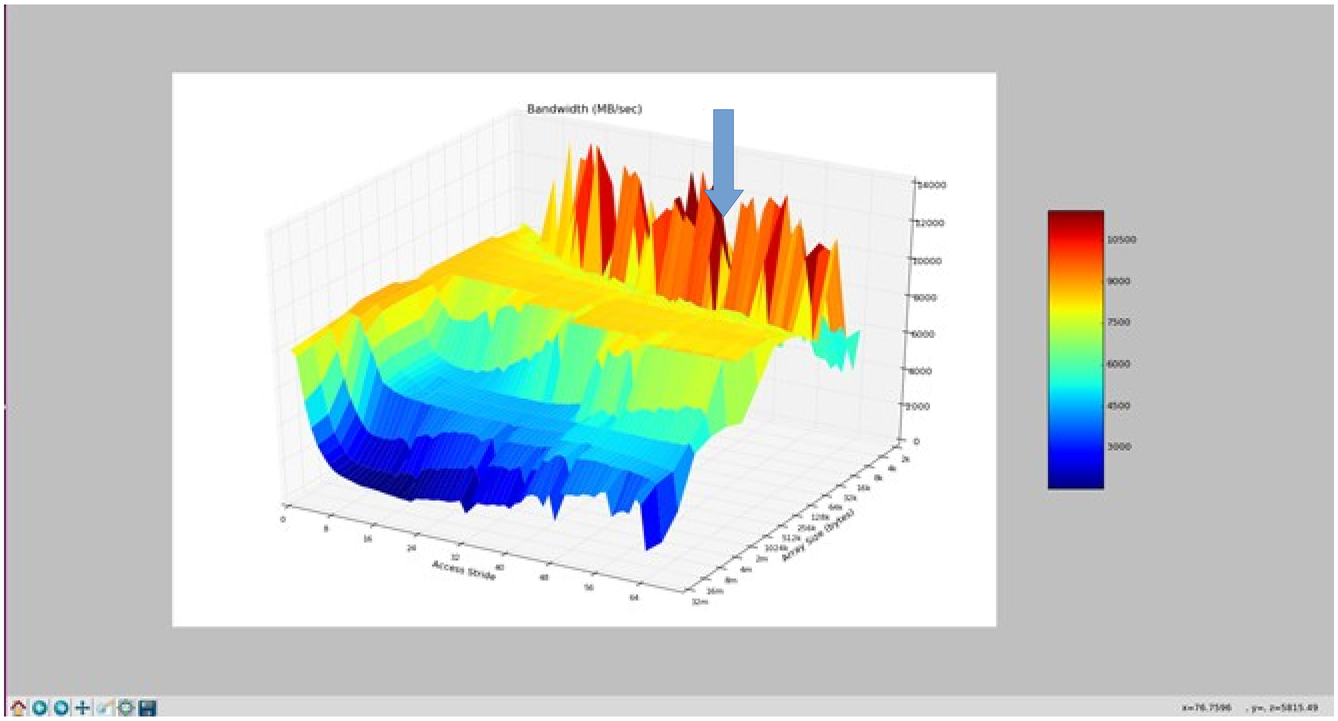
(16) What is the range (min, max) of strides used in the test?
The minimum stride is 0
The maximum stride is 64

(17) What is the range (min, max) of array sizes used in the test?
The minimum array size is 2k.
The maximum array size is 32M.

(18) Take a screen capture of the displayed "memory mountain" (maximize the window so it's sufficiently large to read), and place the resulting image in your report

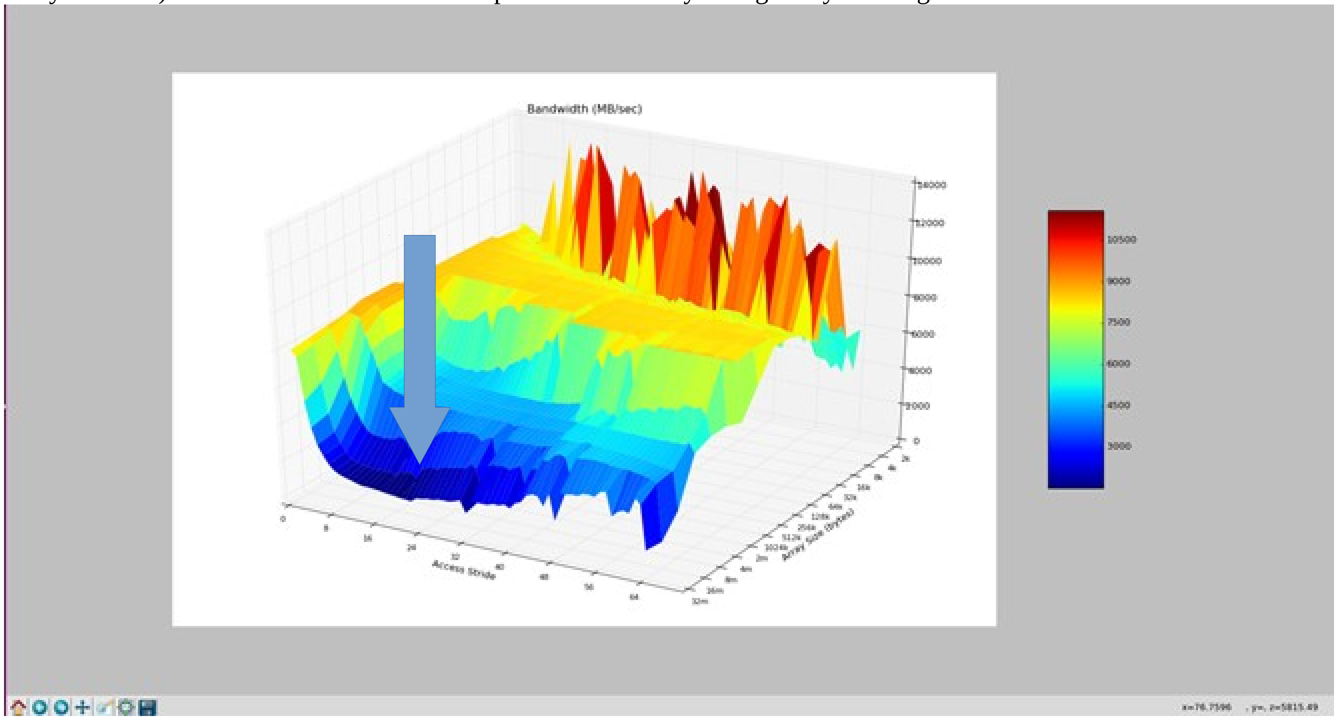


(19) What region (array size, stride) gets the most consistently high performance? (Ignoring spikes in the graph that are noisy results...) What is the read bandwidth reported? Annotate your figure by drawing an arrow on it.



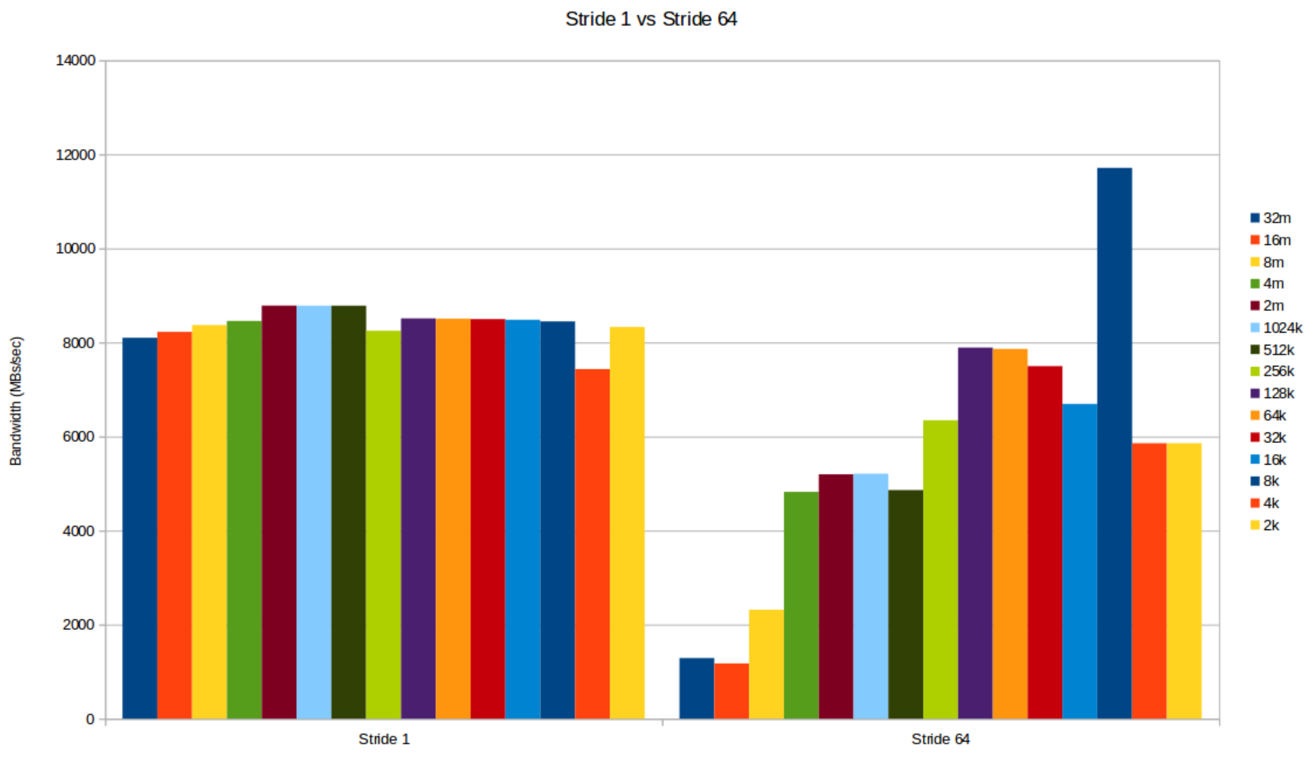
Between the 2k and 8k array sizes are the highest performing regions.

(20) What region (array size, stride) gets the most consistently low performance? (Ignoring spikes in the graph that are noisy results...) What is the read bandwidth reported? Annotate your figure by drawing an arrow on it.



The 32 Meg array size between the 8 and 24 strides are the lowest performing.

(21) Using LibreOffice calc, create two new bar graphs: One for stride=1, and the other for stride=64. Place them side-by-size in the report.
No credit will be given for sloppy graphs that lack X and Y axis labels and a title.
You can obtain the raw data from results.txt. Open it in gedit and turn off Text Wrapping in the preferences. (Otherwise, the columns will be a mess)



(22) When you look at the graph for stride=1, you (should) see relatively high performance compared to stride=64. This is true even for large array sizes that are much larger than the L3 cache size reported in /proc/cpuinfo.
How is this possible, when the array cannot possibly all fit into the cache? Your explanation should include a brief overview of hardware prefetching as it applies to caches.
The hardware prefetcher will grab what it thinks the next data set that will be used before it gets processed. So it can replace an already used bit of data with the next bit of data that will be processed.

(23) What is temporal locality? What is spatial locality?
Temporal locality accesses the same memory address over and over.
Spatial locality accesses the neighboring addresses and moves to the next one.

(24) Adjusting the total array size impacts temporal locality - why? Will an increased array size increase or decrease temporal locality?
This should not affect the temporal locality because there is only one piece of data being accessed over and over.

(25) Adjusting the read stride impacts spatial locality - why? Will an increased read stride increase or decrease spatial locality?
An increase in stride will affect spatial locality because the neighboring elements are being pushed further away and are harder to access.

(26) As a software designer, describe at least 2 broad "guidelines" to ensure your programs run in the high-performing region of the graph instead of the low-performing region.
A smaller stride and a smaller array size should allow the cache to do its job well.