Lab Report

ECPE 170 – Computer Systems and Networks – Spring 2016

Name: Dominic Lesaca

Lab Topic: Performance Optimization (Memory Hierarchy) (Lab #: 7)

Question #1:
Describe how a two-dimensional array is stored in one-dimensional computer memory.
Answer:
A two-dimensional array is stored column (inner array) first with memory addresses being ordered for the inner array before the outer array.
Ex: ar[0][1] is before ar[1][0]

Question #2:
Describe how a three-dimensional array is stored in one-dimensional computer memory.
Answer:
A three-dimensional array is sorted using the inner most array first then moving outwards.
Ex: ar[0][0][1] is before ar[0][1][0] which is before ar[1][0][0]

Question #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.
Answer:
array1[0][0] is at address 0x7ffee97555a0
array1[0][1] is at address 0x7ffee97555a4
array1[0][2] is at address 0x7ffee97555a8
array1[0][3] is at address 0x7ffee97555ac
array1[0][4] is at address 0x7ffee97555b0
array1[1][0] is at address 0x7ffee97555b4
array1[1][1] is at address 0x7ffee97555b8
array1[1][2] is at address 0x7ffee97555bc
array1[1][3] is at address 0x7ffee97555c0
array1[1][4] is at address 0x7ffee97555c4
array1[2][0] is at address 0x7ffee97555c8
array1[2][1] is at address 0x7ffee97555cc
array1[2][2] is at address 0x7ffee97555d0
array1[2][3] is at address 0x7ffee97555d4
array1[2][4] is at address 0x7ffee97555d8

array2[0][0][0] is at address 0x7ffee97555e0
array2[0][0][1] is at address 0x7ffee97555e4
array2[0][0][2] is at address 0x7ffee97555e8
array2[0][0][3] is at address 0x7ffee97555ec
array2[0][0][4] is at address 0x7ffee97555f0
array2[0][1][0] is at address 0x7ffee97555f4
array2[0][1][1] is at address 0x7ffee97555f8
array2[0][1][2] is at address 0x7ffee97555fc
array2[0][1][3] is at address 0x7ffee9755600
array2[0][1][4] is at address 0x7ffee9755604
array2[0][2][0] is at address 0x7ffee9755608
array2[0][2][1] is at address 0x7ffee975560c
array2[0][2][2] is at address 0x7ffee9755610
array2[0][2][3] is at address 0x7ffee9755614

array2[0][2][4] is at address 0x7ffee9755618
array2[0][3][0] is at address 0x7ffee975561c
array2[0][3][1] is at address 0x7ffee9755620
array2[0][3][2] is at address 0x7ffee9755624
array2[0][3][3] is at address 0x7ffee9755628
array2[0][3][4] is at address 0x7ffee975562c
array2[0][4][0] is at address 0x7ffee9755630
array2[0][4][1] is at address 0x7ffee9755634
array2[0][4][2] is at address 0x7ffee9755638
array2[0][4][3] is at address 0x7ffee975563c
array2[0][4][4] is at address 0x7ffee9755640
array2[1][0][0] is at address 0x7ffee9755644
array2[1][0][1] is at address 0x7ffee9755648
array2[1][0][2] is at address 0x7ffee975564c
array2[1][0][3] is at address 0x7ffee9755650
array2[1][0][4] is at address 0x7ffee9755654
array2[1][1][0] is at address 0x7ffee9755658
array2[1][1][1] is at address 0x7ffee975565c
array2[1][1][2] is at address 0x7ffee9755660
array2[1][1][3] is at address 0x7ffee9755664
array2[1][1][4] is at address 0x7ffee9755668
array2[1][2][0] is at address 0x7ffee975566c
array2[1][2][1] is at address 0x7ffee9755670
array2[1][2][2] is at address 0x7ffee9755674
array2[1][2][3] is at address 0x7ffee9755678
array2[1][2][4] is at address 0x7ffee975567c
array2[1][3][0] is at address 0x7ffee9755680
array2[1][3][1] is at address 0x7ffee9755684
array2[1][3][2] is at address 0x7ffee9755688
array2[1][3][3] is at address 0x7ffee975568c
array2[1][3][4] is at address 0x7ffee9755690
array2[1][4][0] is at address 0x7ffee9755694
array2[1][4][1] is at address 0x7ffee9755698
array2[1][4][2] is at address 0x7ffee975569c
array2[1][4][3] is at address 0x7ffee97556a0
array2[1][4][4] is at address 0x7ffee97556a4
array2[2][0][0] is at address 0x7ffee97556a8
array2[2][0][1] is at address 0x7ffee97556ac
array2[2][0][2] is at address 0x7ffee97556b0
array2[2][0][3] is at address 0x7ffee97556b4
array2[2][0][4] is at address 0x7ffee97556b8
array2[2][1][0] is at address 0x7ffee97556bc
array2[2][1][1] is at address 0x7ffee97556c0
array2[2][1][2] is at address 0x7ffee97556c4
array2[2][1][3] is at address 0x7ffee97556c8
array2[2][1][4] is at address 0x7ffee97556cc
array2[2][2][0] is at address 0x7ffee97556d0
array2[2][2][1] is at address 0x7ffee97556d4
array2[2][2][2] is at address 0x7ffee97556d8

array2[2][2][3] is at address 0x7ffee97556dc
array2[2][2][4] is at address 0x7ffee97556e0
array2[2][3][0] is at address 0x7ffee97556e4
array2[2][3][1] is at address 0x7ffee97556e8
array2[2][3][2] is at address 0x7ffee97556ec
array2[2][3][3] is at address 0x7ffee97556f0
array2[2][3][4] is at address 0x7ffee97556f4
array2[2][4][0] is at address 0x7ffee97556f8
array2[2][4][1] is at address 0x7ffee97556fc
array2[2][4][2] is at address 0x7ffee9755700
array2[2][4][3] is at address 0x7ffee9755704
array2[2][4][4] is at address 0x7ffee9755708

When reading the addresses of each spot in the array, we can see that each address increments by 4. This shows that the array is column split as the addresses go in order based on changes to the inner most array first.

Question #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.
Answer:

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

Question #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.
Answer:
The array has good spatial locality as it is read sequentially but bad temporal locality as each element is only read once. Both i and j have good temporal locality (especially j since it is called 3 times for every time I is called) as they are called multiple times but poor temporal locality as they do not use nearby memory addresses. Therefore, sumarrayrows() has good spatial and temporal locality

Question #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.
Answer:

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	3	5	2	4	6

Question #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.
Answer:
The array has bad spatial locality as it is not read sequentially and bad temporal locality as each element is only read once. Both i and j have good temporal locality (especially i since it is called 2 times for every time I is called) as they are called multiple times but poor temporal locality as they do not use nearby memory addresses. Therefore, sumarrayrows() has bad spatial and good temporal locality

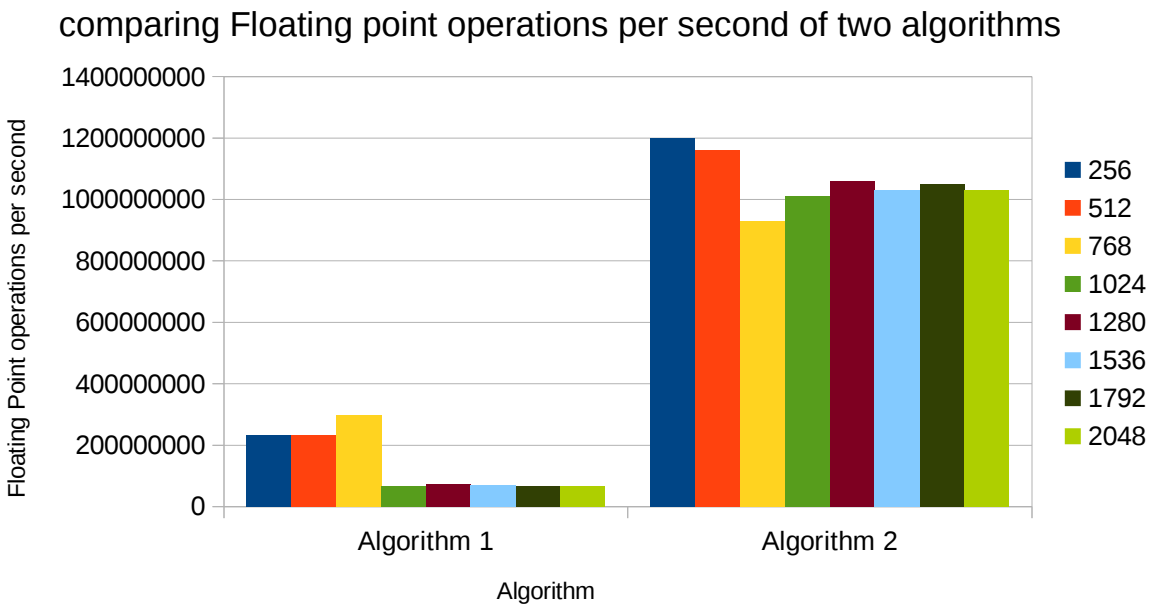
Question #8:
Inspect the provided source code. Describe how the *two*-dimensional arrays are stored in memory, since the code only has one-dimensional array accesses like: a[element #].
Answer:
Each matrix is stored as a sequence of n^2 values. The first n constitute the first row of the matrix, the next n the second row, etc. As such, given that we are numbering the rows and columns starting from 0, to reach a position that is in row number i requires skipping over i full rows of n elements. Thus, array[row][column] is the same as array[row*n+column]

Question #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.

Answer:

	256	512	768	1024	1280	1536	1792	2048
Algorithm 1	2.33E+08	2.34E+08	3.00E+08	6.67E+07	7.25E+07	6.99E+07	6.73E+07	6.74E+07
Algorithm 2	1.20E+09	1.16E+09	9.28E+08	1.01E+09	1.06E+09	1.03E+09	1.05E+09	1.03E+09

Question #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.
Answer:



Question #11:
send script to mercurial
Answer:
its there

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

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 69
model name      : Intel(R) Core(TM) i3-4020Y CPU @ 1.50GHz
stepping        : 1
microcode       : 0x1d
cpu MHz                 : 1501.000
cache size      : 3072 KB
physical id     : 0
siblings        : 1
core id         : 0
```

cpu cores : 1
apicid : 0
initial apicid : 0
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 : 3002.00
clflush size : 64
cache_alignment : 64
address sizes : 42 bits physical, 48 bits virtual
power management:

Question #13:
Based on the processor type reported, obtain the following specifications for your CPU from cpu-world.com or cpudb.stanford.edu
Answer:
(a) L1 instruction cache size: 2 x 32 KB
(b) L1 data cache size: 2 x 32 KB
(c) L2 cache size: 2 x 256 KB
(d) L3 cache size: 3 MB
(e) What URL did you obtain the above specifications from?: http://www.cpu-world.com/CPUs/Core_i3/Intel-Core%20i3-4130.html

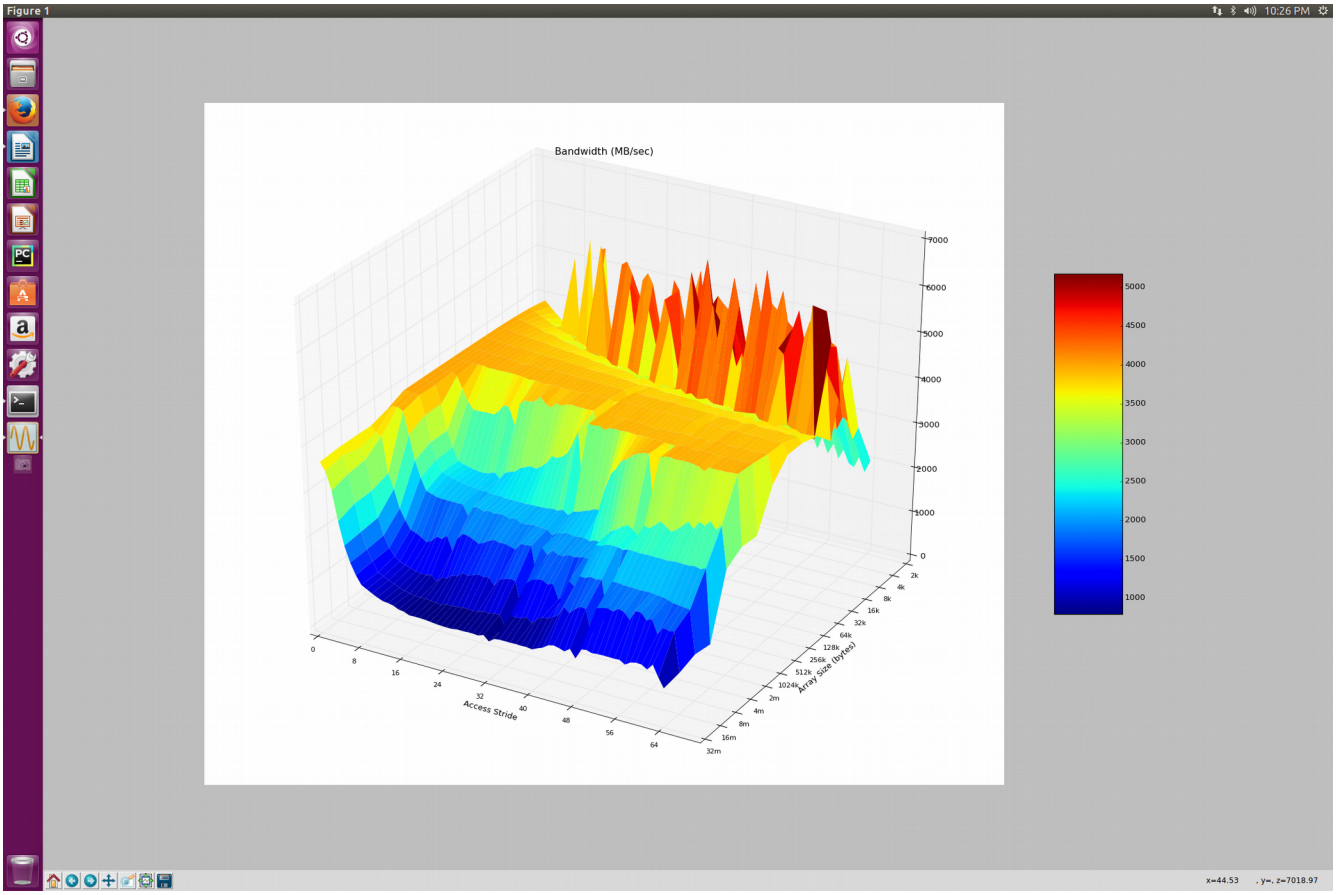
Question #14:
Why is it important to run the test program on an idle computer system?
Explain what would happen if the computer was running several other active programs in the background at the same time, and how that would impact the test results.
Answer:
Having the computer being idle would have allowed the program to analyze the entire processor. Having other active programs would have decreased the amount of the processor that could actually be analyzed.

Question #15:
What is the size (in bytes) of a data element read from the array in the test?
Answer:
the data elements were 4 bytes

Question #16:
What is the range (min, max) of *strides* used in the test?
Answer:
the strides has a range from 0 to 64

Question #17:
What is the range (min, max) of *array sizes* used in the test?
Answer:
the array size goes from 2000 to 32 million elements

Question #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
Answer:



Question #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.
Answer:
the highest consistent performance is from the array sizes from 128K to 16K. The bandwidth is about

3700.

Question #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.
Answer:
the lowest performance is when the array size passes 1024K and the stride is above 8. the bandwidth here is about 1000

Question #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)
Answer:

Question #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.
Answer:
the array size goes from 2000 to 32 million elements

Question #23:
What is temporal locality? What is spatial locality?
Answer:
temporal locality is is when variables will be used multiple times in the near future
Spatial locality is when one memory address is used, it is likely for other nearby addresses to be used.

Question #24:
Adjusting the total array size impacts temporal locality - why? Will an increased array size increase or decrease temporal locality?
Answer:
changing the size of the array means that the variables we have called will be called less often, reducing temporal locality. Increased array size will decrease temporal locality.

Question #25:

Adjusting the read *stride* impacts spatial locality - why? Will an increased read stride increase or decrease spatial locality?

Answer:

read stride effect how far we move between different variables in the array therefore effecting the space traveled. This decreases spatial locality since we skip over lots of nearby spots in the array.

Question #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.

Answer:

1) Keep read stride relatively short
2) keep the array size to a medium size, avoid going into the high thousands and millions of elements