Lab 7 Report

Ethan Coe-Renner

October 20, 2021

Contents

1. A two dimensional array is stored linearly such that the last element in the first array is followed by the first element of the second array, and so on.

2. A three dimensional array is stored linearly in the same way as two dimensional arrays, with the last element of the first 2d array followed by the first element of the second 2d array, and so on.

3.
   ```
   This is a two dimensional array displayed graphically in rows and columns
   10 11 12 13 14
   15 16 17 18 19
   20 21 22 23 24
   This is a two dimensional array displayed as it is in memory
   10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
   This is a three dimensional array displayed graphically as layers
   100 101 102 103 104 105 106
   107 108 109 110 111 112 113
   114 115 116 117 118 119 120
   121 122 123 124 125 126 127
   128 129 130 131 132 133 134

   135 136 137 138 139 140 141
   142 143 144 145 146 147 148
   149 150 151 152 153 154 155
   156 157 158 159 160 161 162
   163 164 165 166 167 168 169

   170 171 172 173 174 175 176
   ```

```
177 178 179 180 181 182 183
184 185 186 187 188 189 190
191 192 193 194 195 196 197
198 199 200 201 202 203 204

This is a three dimensional array displayed as it is stored in memory
100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 12
```

4.

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

5. `sumarrayrows()` has good temporal locality with the variables i and j. It also has good spatial locality with the array because it accesses elements in the same order as they are stored in memory.

6.

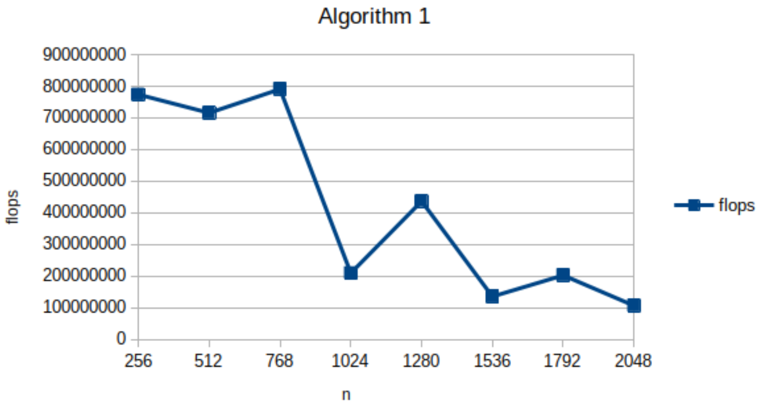
Memory Access	0	4	8	12	16	20
Memory Contents	a[0][0]	a[0][1]	a[1][0]	a[1][1]	a[2][0]	a[2][1]
Program Access Order	1	4	2	5	3	6

7. `sumarraycols()` has good temporal locality with the variables i and j. It does not access elements in the same order as they are stored in memory.

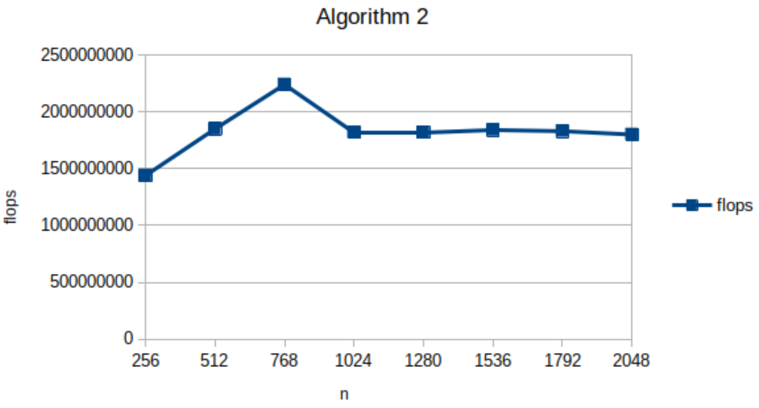
8. They are stored linearly, where rows are lined up in a line, rather than two dimensionally.

9.

algorithm	n	flops
1	256	7.80E+08
2	256	2.73E+09
1	512	7.21E+08
2	512	2.88E+09
1	768	7.77E+08
2	768	2.79E+09
1	1024	2.50E+08
2	1024	2.26E+09
1	1280	5.06E+08
2	1280	2.28E+09
1	1536	1.85E+08
2	1536	2.34E+09
1	1792	2.96E+08
2	1792	2.28E+09
1	2048	1.43E+08
2	2048	2.19E+09



10.



11. `./part3/script.sh`

12.
```
processor : 0
vendor_id : GenuineIntel
cpu family : 6
model : 142
model name : Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz
stepping : 10
microcode : 0xe0
cpu MHz : 800.131
cache size : 6144 KB
physical id : 0
siblings : 8
core id : 0
cpu cores : 4
apicid : 0
initial apicid : 0
fpu : yes
fpu_exception : yes
cpuid level : 22
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 cl
vmx flags : vnmi preemption_timer invvpid ept_x_only ept_ad ept_1gb flexpriority t
bugs : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs itlb_m
bogomips : 3600.00
clflush size : 64
cache_alignment : 64
```

```
address sizes : 39 bits physical, 48 bits virtual
power management:
```

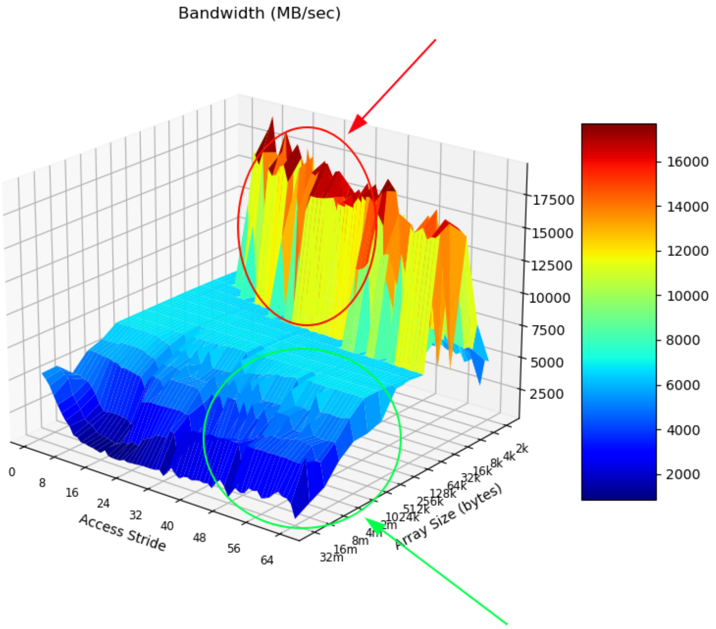
13. (a) instruction cache size: 4 * 32 KB
    (b) data cache size: 4 * 32 KB
    (c) L2 cache size: 2 * 256 KB
    (d) L3 cache size: 6MB

14. It is important to run on an idle system so that the cache does not get overwritten during the experiment, as this would cause the results to be flawed. It would likely result in lower bandwidth because the program would get cache misses more frequently.

15. The array is filled with doubles, which have size 8 bytes.

16. strides from 1 to 64 are used

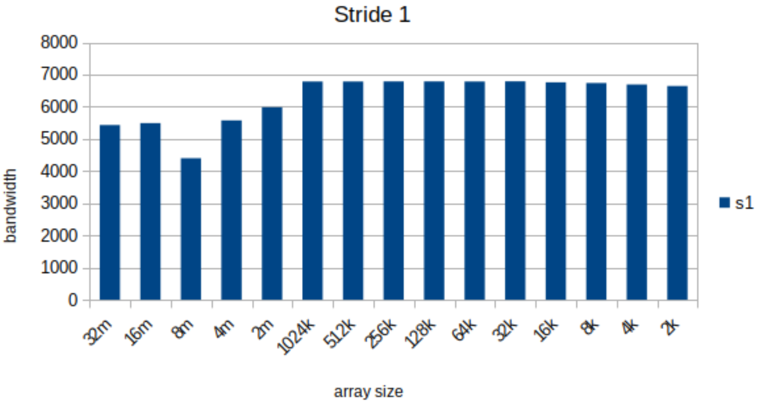
17. Arrays range from 2 KB to 64 MB



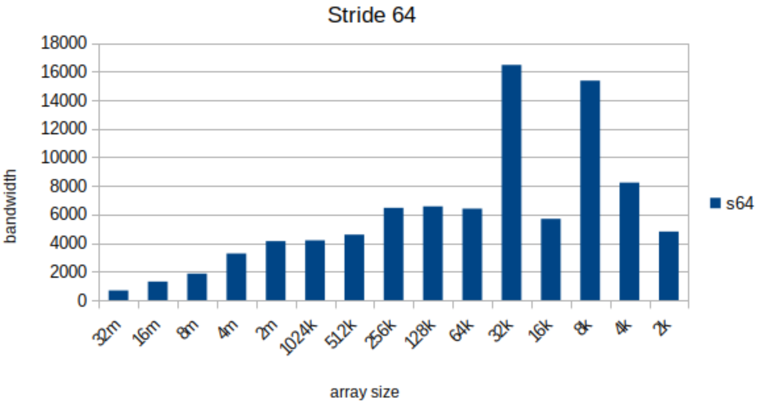
18.

19. The most consistently high performance is achieved at small array sizes. It gets a read bandwidth of 16000-18000 at small array sizes (2k-16k). Shown in red circle and arrow on graph.

20. The most consistently low performance is achieved at large array sizes. it gets a read bandwidth of 1000-2000 at large array sizes (16-32m). shown in green circle and arrow on graph



21.



22. The stride=1 has higher performance than the stride=64 because of prefetching. Prefetching is where the computer retrieves data before it is needed and stores it in the cache for easy access. In this case, the prefetching happens spatially, where the computer will prefetch data close to data it is currently fetching. For stride=1, the loop is going to

be accessing data that is close to previously accesssed data, therefore resulting in many cache hits and therefore higher performance.

23. Temporal locality refers to the amount of time between retrieval of the same data, it is related to the idea that data being fetched now will likely be needed later and so should be stored in the cache. Spatial locality refers to the closeness of data in memory, related to the idea that data being fetched now will likely be close to (in memory) data needed soon, and so data around the currently fetched data should b estored in the cache.

24. Because the array is read twice, the size affects temporal locality because if the array is small enough to fit entirely in the cache then its temporal locality means that the second retrieval will just be reading from cache. Therefore increased array size will decrease temporal locality.

25. The stride affects spatial locality because the computer will load nearby elements into the cache when it retrieves an element, and so if the stride is smaller, it will be more likely to hit an element already in the cache because subsequent elements are closer to previous elements. therefore, increased read stride will decrease spatial locality.

26. Software designers should use try to reaccess the same variables, rather than recalculating values in order to improve temporal locality. Additionally, designers should store data in such a way that data used together stays together (in arrays for example).