Deliverable 1: mmult_ps3.log

N	mult_0	mult_1	mult_2	mult_3	mul_t_0	mul_t_1	mul_t_2	mul_t_3
8	2.35758	2.90768	6.23075	3.48922	3.00795	2.72595	6.23075	5.81536
16	2.10194	2.17397	5.58275	5.02046	2.06463	2.01689	5.7673	8.51029
32	1.80139	1.80498	5.96119	5.80834	2.06401	2.06401	6.47215	11.7675
64	1.76463	1.75779	5.3354	5.46397	1.84729	1.75779	5.88973	14.1722
128	1.43629	1.71646	5.84247	5.14528	1.74706	1.6962	5.284	8.58913
256	0.970887	1.0168	3.90581	5.19857	1.46371	1.49231	5.59241	10.5457

Deliverable 2: mmult_0.log

N	GF/s ijk	GF/s ikj	GF/s jik	GF/s jki	GF/s kij	GF/s kji
8	2.42307	2.90768	2.5656	1.93845	2.90768	1.7104
16	1.98251	3.29172	2.12111	2.63337	3.15766	2.38988
32	1.90758	3.02034	1.90758	3.07153	2.28237	2.39077
64	1.59406	2.95446	1.67966	1.98472	3.20501	1.82132
128	1.39268	2.9252	1.19638	0.579069	2.86304	0.605582
256	0.968763	3.38106	0.860036	0.332122	3.10602	0.335392
512	0.808319	3.46911	0.791739	0.243633	3.09139	0.253756

Deliverable 3: mult_1.log

N	GF/s ijk	GF/s ikj	GF/s jik	GF/s jki	GF/s kij	GF/s kji
8	3.79263	3.00795	2.64335	1.98251	2.90768	2.23668
16	7.04893	3.17202	2.07692	2.13408	2.93212	2.53761
32	11.4696	3.14618	1.82314	2.48247	3.27112	3.15715
64	12.257	3.18252	1.79608	1.85864	3.13847	2.18033
128	11.1703	3.44389	1.50655	0.569433	3.32184	0.583299
256	11.7797	3.65444	0.970461	0.342922	3.38623	0.346789
512	10.9837	3.49581	0.802825	0.249933	3.47319	0.244136

Deliverable 4: mult_2.log

N	mult_0	mult_1	mult_2	mult_3	mul_t_0	mul_t_1	mul_t_2	mul_t_3
8	2.29554	2.90768	6.23075	2.64335	2.90768	3.6346	5.81536	5.81536
16	1.64586	1.8173	5.58275	3.81335	2.07692	2.4572	6.28688	9.06291
32	2.04537	2.06872	6.04067	8.54812	2.14716	2.13703	6.51871	13.94
64	1.2811	1.71136	5.3354	8.55678	1.91354	1.88569	4.5809	13.9541
128	1.47159	1.75846	5.25823	6.80086	1.71646	1.79506	5.95545	12.1116
256	1.06573	1.11063	3.98308	5.75219	1.65145	1.5796	5.72246	10.6471
512	0.835774	1.08412	3.83895	5.25719	1.53938	1.50665	5.65849	9.63914
1024	0.809048	1.05396	3.57914	4.96719	1.44833	1.47689	5.66347	8.82033

PS4 Questions
==========

Add your answers to this file in plain text after each question. Leave a blank line between the text of the question and the text of your answer.

CPU
---

1. What level of SIMD/vector support does the CPU your computer provide?

SSE2, SSE3, SSE41, SSE42, AVX, AVX2

2. What is the maximum operand size that your computer will support?

Maximum supported operand size is 256 bits.

3. What is the minimum operand size that your computer will support?

Minimum supported operand size is 32 bits.

4. What is the clock speed of your CPU? You may need to look this up via "About this Mac" on MacOSX. If your Macbook is using Apple M1 Chip, try "sudo powermetrics" in the terminal, looking for "CPU frequency". Look at the "Performance" Tab in Task Manager on a Windows; and try "lscpu" in the terminal on a Linux.

The CPU clock speed is 3.6 GHz.

5. Based on the output from bandwidth.exe on your computer, what do you expect L1 cache and L2 cache sizes to be? What are the corresponding bandwidths? How do the cache sizes compare to what "about this mac" (or equivalent) tells you about your CPU? (There is no "right" answer for this question -- but I do want you to do the experiment.)

The Bandwidth.exe output shows the first large drop in bandwidth after 8k bytes and second large drop after 262k bytes, indicating available L1 & L2 caches to 8KB & 262 KB respectively. However, this need not reflect the true size of the cache, as other programs might be using it. The L1 cache size for data on this computer is shown to be 32 KB. The total L1 cache is indicated as 384 with 6 cores, which equals 6*(32+32), where 32 KB is for L1 instruction cache, and the other 32 KB is for L1 data cache. The L2 cache size inferred from bandwidth.exe (262 KB) is close to the task manager version on this computer (256 KB). The task manager shows 1.5 MB of L2 cache for 6 cores i.e., 256 KB for each core.
The bandwidth for L1 cache & L2 cache are 172 GB/s & 41 GB/s approximately.

Perf
---

6. Based on the output from running this image on your computer, what do you expect L1 cache and L2 cache (if present) sizes to be? What are the corresponding bandwidths? How do the cache sizes compare to what "about this mac" (or equivalent) tells you about your CPU? (There is no "right" answer for this question – but I do want you to do the experiment.)

The numbers for the docker version correspond closely to those given in the task manager i.e., 32 KB & 256 KB for L1 & L2 cache respectively. The bandwidth for L1 cache is 219 GB/s & for L2 cache it is 80 GB/s approximately.

Roofline: Homegrown
---

7. What is the (potential) maximum compute performance of your computer? (The horizontal line.) What are the L1, L2, and RAM bandwidths? How do those bandwidths correspond to what was measured above with the bandwidth program?

The maximum compute performance is 44.3 GFlops/sec. The bandwidth for L1 cache, L2 cache & DRAM are 175.8 GB/s, 67.3 GB/s & 17 GB/s respectively. These number are slightly lower than measured above but still show similar trend.

8. Based on the clock speed of your CPU and its maximum Glop rate, what is the (potential) maximum number of *double precision* floating point operations that can be done per clock cycle? (Hint: Glops / sec :math:`\div` GHz = flops / cycle.) There are several hardware capabilities that can contribute to supporting more than one operation per cycle: fused multiply add (FMA) and AVX registers. Assuming FMA contributes a factor of two, SSE contributes a factor of two, AVX/AVX2 contribute a factor of four, and AVX contributes a factor of eight of eight, what is the expected maximum number of floating point operations your CPU could perform per cycle, based on the capabilities your CPU advertises via cpuinfo (equiv. lscpu)? Would your answer change for single precision (would any of the previous assumptions change)?

Based on maximum flop rste & the clock speed the computer supports 44.3/3.6 = 12.3 flops/cycle.
Based on cpuinfo- FMA & AVX= 2*4 = 8 flops/cycle.
For single precision = 2*8 = 16 flops/cycle.

Roofline: Docker
---

9. What is the maximum compute performance of your computer? (The horizontal line.) What are the L1, L2, and DRAM bandwidths? How do those bandwidths correspond to what was measured above?

The maximum performance is 36.3 GFlops/sec. The bandwidth for L1, L2 & DRAM are 109 GB/s, 67.1 GB/s & 16.5 GB/s respectively. The numbers are lower than those measured above.

mult
---

10. Referring to the figures about how data are stored in memory, what is it about the best performing pair of loops that is so advantageous?

The best order is with 'j' in the innermost loop. With this ordering the only increment that happens in the innermost loop is along the rows of B matrix. For the row elements are stored contiguously on the memory, making the access operations fast.

11. What will the data access pattern be when we are executing ``mult_trans`` in i,j,k order? What data are accessed in each if the matrices at step (i,j,k) and what data are accessed at step (i, j, k+1)? Are these accesses advantageous in any way?

At the k^th step, the input elements are A(i,k) & B(j,k), while at the (k+1)^th step, the input elements are A(i,k+1) & B(j,k+1). Thus, Data is accessed along the rows of both the input matrices, i.e. contiguously. Since, both the input matrices are being accessed contiguously, we expect a huge speedup over executing 'mult' function with 'ijk' order.

12. Referring again to how data are stored in memory, explain why hoisting ``C(i,j)`` out of the inner loop is so beneficial in mult_trans with the "ijk" loop ordering.

The inner loop doesn't iterate on the location of element in the 'C' array. Thus, by hoisting ''C(i,j)'' out of the inner loop, the load C(i,j) & store C(i,j) operations which require memory access can be avoided. The computed value can be stored on the temporary variable 't', which is stored in the register and can be accessed much faster than memory (even L1 cache).

13. (AMATH 583 ONLY) What optimization is applied in going from ``mult_2`` to ``mult_3``?

Going from ''mult_2'' to ''mult_3'', blocking is used. Thus, computations are performed on a small block of the original matrix, that can fit in the cache. Thus, cache is optimally used, and slower memory operation are not required in the innermost loops, for large matrices.

14. How does your maximum achieved performance for ``mult`` (any version) compare to what bandwidth and roofline predicted? Show your analysis.

The max performance is achieved by mult_trans_3. This involves hoisting, unrolling and blocking.
The innermost loop has 8 floating point operations and 8 doubles- C(i,j), C(i+1,j), C(i,j+1), C(i+1,j+1), A(i,k), A(i+1,k),B(k,j),B(k,j+1)=64bytes.
Therefore, numerical intensity = 1/8.
From the roofline plot, assuming L1 cache is available (blocking should allow that), the Gflops/sec should be approximately 12 Gflops/sec. Thus observed performance is close to the predicted performance.

loop ordering
---

15. Which variant of blurring function between struct of arrays and array of structs gives the better performance? Explain what about the data layout and access pattern would result in better performance.

SOA outer and AOS inner both give similar performance. Since for SOA, pixels in a given color plane are contiguous, iterating through the same color plane in the innermost loop leads to best performance. Thus, the index corresponding to different colors 'k' must be in the outermost loop. For AOS, all the color plane elements for a given pixels are stored contiguously. Therefore, 'k' must be iterated through in the innermost loop.
Since, at the memory level, the operations for both SOA outer & AOS inner are similar, both these methods give similar performances.
Speculation- The performance of SOA should probably be slightly better than AOS assuming jump between color planes for SOA and jump between pixels for AOS are slow. The first pixel of a color plane may not be immmediately accessible after the last pixel of previous color plane for SOA. Similarily, the last color of previous pixel need not be immediately followed by first color of the next pixel. Thus, for SOA there will be 2 additional jumps (across colors) apart from 3*Number of pixels-1 contiguous jumps, while for AOS there will be N-1 additional jump (across pixels) apart from the 3*Number of pixels-1 contguous jumps.

16. Which variant of the blurring function has the best performance overall? Explain, taking into account not only the data layout and access pattern but also the accessor function.

Tensor outer gives the best performance of all the methods. Since, the accessor function is defined in 'color plane' or 'slab'-major ordering, k must be in the outermost loop for best performance. Since all the elements are contiguously stored in the accessor function i.e. even the first pixel of a color plane is immediately next to the last pixel of the pervious color plane, there are no additional jumps apart from 3*Number of pixels-1 contiguous jumps.