Gabriel Angelo F. Limlingan
CoE 163 Computing Architectures and Algorithms
April 9, 2024

Software Exercise 02: MMM Loop Ordering

Synopsis: Implement various loop ordering for matrix-matrix multiplication and observe the performance impact of loop ordering on small, medium, and large matrices.

ME Questions

1. *What matrix sizes did you choose?* **10x10, 100x100, 500x500**

I opted to select 500x500 as the matrix size since I noticed that the runtime increases exponentially as the matrix size increases. If I were to have a larger size (like 1000x1000), the runtime would span hours until the output is shown.

2. *How many times did you run the code per matrix size?* **5 times per matrix size**

To have a better value for the runtime averages, I ran the code five times instead of three.

3. *List the runtimes you recorded per matrix size (you can organize your results into a table). What are the average runtimes per matrix size?*

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
10x10 (milliseconds)						
ijk	**0.72100**	**0.56300**	0.86600	**0.56300**	1.01800	0.74620
kji	0.52900	0.55800	0.80300	0.54100	**0.91600**	0.66940
jik	0.52500	0.53800	0.89600	0.55100	0.97600	0.69720
jki	0.55100	**0.50900**	0.88300	**0.52300**	1.86600	0.86640
kij	0.52800	0.54400	**1.00900**	0.54600	**3.06200**	1.13780
ikj	**0.51200**	0.55900	**0.56600**	0.53700	0.98300	0.63140
					Total Average	**0.79140**
100x100 (milliseconds)						
ijk	**455.08**	395.35	406.01	409.31	**421.00**	417.35
kji	408.02	399.25	**387.91**	413.51	417.22	405.18
jik	403.47	406.77	404.75	**401.95**	407.28	404.84
jki	422.85	**426.83**	607.31	422.52	418.40	459.58
kij	397.98	**390.02**	**766.37**	412.20	405.78	474.47
ikj	**394.22**	398.08	746.81	**471.12**	**395.37**	481.12
					Total Average	**440.42**
500x500 (seconds)						
ijk	64.268	63.663	64.681	64.700	64.982	64.459
kji	66.266	61.112	61.702	62.226	64.234	63.108
jik	56.172	57.038	57.697	58.975	56.236	57.224
jki	**68.942**	**67.286**	**70.656**	**74.094**	**68.696**	69.935
kij	**55.105**	56.296	**55.510**	**55.518**	57.431	55.972
ikj	55.724	**54.637**	55.999	56.619	**55.461**	55.688
					Total Average	**61.064**

4. *What can you say about how the runtime is affected by matrix size? What do you think are the reasons behind slowness due to matrix size?*

As mentioned in (1), the runtime increases exponentially as the matrix size increases. This is due to the fact that the performance of the matrix-multiplication algorithm has three nested loops which puts their performance at **O(n³)**.

5. *Are there any loop orderings that consistently run slower than the others?* **Yes!**

When I tested the program for n = 10 and n = 100, the comparison of the runtimes of each loop order did not yield any consistent findings on whichever is faster or slower. However, when running the program with the 'large' matrix size (n = 500), I noticed that the `jki` loop order configuration runs the slowest in all the trials. Conversely, the fastest runtimes for the large matrices are between the `kij` and `ikj` configurations.

6. *Is the slow performance observable for all the matrix sizes? What do you think are the reasons behind the slower performance with changing loop ordering?*

The slow performance is observable even for 'medium' matrix sizes since even a half-second delay in runtime affects the performance of the overall program. For 'large' matrix sizes, it's more apparent that there is slower performance with changing loop ordering (especially with config. `jki`). In a row-wise matrix represented by a 2D array, accessing elements from different rows are not directly accessible in the cache; which opts it to invoke a *cache miss* and instead access the slow memory. In config. `jki`, more cache misses happen since `i` is in the innermost loop. The figure below shows the 2D array `C[i][j]` and the values that reside inside.