Research Report: Summer 2019

Paul Scanlon and Matt Murdoch

Colorado State University, Undergraduate Research for Dr. Sanjay Rajopadhye with Nirmal Prajapati

Introduction

Over the course of the summer of 2019, we were tasked with optimizing the problem of parenthesis ordering, MCO, for matrix chain multiplication, OSP. The triply nested loop featured in conventional implementation invites memory traversal optimization for locality and vectorization and suggests the attainability of cubic run-time.

Pursuing this optimization gives us insight about roof-line modeling within the department. Finding the limit of memory boundedness and computational intensity boundedness gives us a direction for optimizing any program on department specific machines. Reaching machine peak is important for advancing general roofline analysis.

The memory structure for this problem given a problem size N can be considered a triangular table with an i and j axis where $i \leq j \leq N$. All permutations of these variables for memory structure traversal are to be considered as they may be vectorized differently. Combining the j and i axes into $j-i$ creates a diagonal direction of traversal, d. Additionally, 'k' reductions must be done in each cell of the structure to build the solution where $i \leq k < j$.

This third loop can be thought of as a third dimension of the table, allowing one to conceptualize three dimensional navigation of the table's cells. There are thus 96 permutations of these loop variables given by:

$$2(i \text{ or } d) \times 2(i \text{ or } d \ +/-) \times 2(j \ +/-) \times 2(k \ +/-)$$
$$\times 3! \ (Permutations\ of\ Ordering\ Three\ Elements) = 96$$

It is clear that a scheme is required to find out which ones will produce the correct answer.

One possible visualization would be a triangular pyramid with the $2D$ table acting as the base and the k dimension rising up to a single point above the top left corner. However, this form of visualizing, while accurate to problem, can be difficult to quickly draw and work with. So, a more compact form of visualizing the $3D$ navigation was created using time stamps on the original $2D$ mapping.

Visualization

The process for proving a mapping's truth value is to track the dependence of each cell in the order of iteration visually. If, at each time step, the required information is complete and available, the mapping should work with the appropriate initialization. First a simple example with $N = 4$ of the mapping d, k, j:

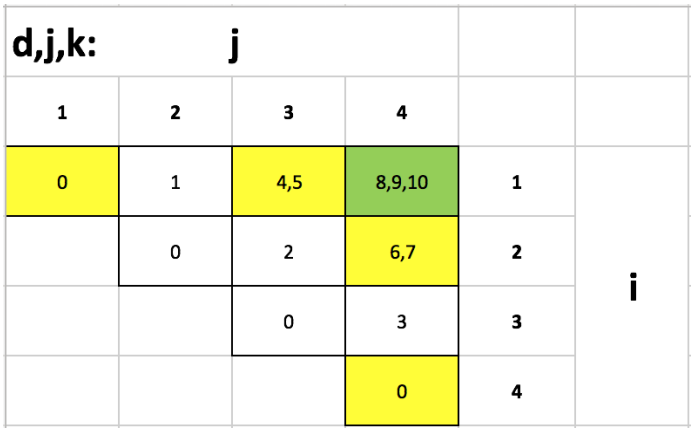

Figure 1: Example of the Visualization constructed for proving Mappings for d, j, k

The time stamps represent the order of building the solution for each cell. For cells with more than one time stamp, the cell was visited multiple times to support the minimum comparison piece of the reduction.

Listing the time steps for this mapping shows that whenever a reduction is encountered, it can be completed fully as all of its dependent cells are complete with information. This visualization also shows that k can be traversed from $i \longrightarrow j$ or from $j \longrightarrow j$. With k as the innermost loop for a diagonal scheme, any direction of traversal is permitted.

A more complicated example of the proof process is shown below for the strict mapping $-i, k, j$ with $N = 5$. This mapping features k was the middle loop meaning that the solution to each cell is not completed in consecutive calculations - part of the solution is built and then came back to later for further evaluation after touching other cells. This mapping relies on the diagonal of zeros being evaluated first. This is accomplished through initialization of the loop variables to promote the diagonal's evaluation first and a $-i, k, j$ traversal order elsewhere.

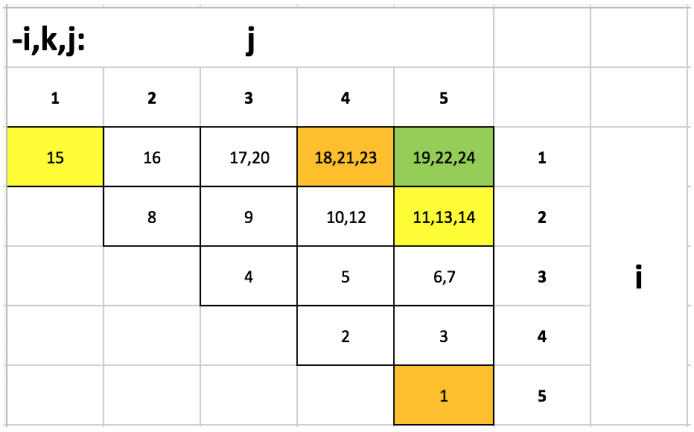

Figure 2: Example of the Visualization Constructed for Proving Mappings for $-i, k, j$

An arbitrarily small problem size is chosen for clarity that still shows all trends of the iteration. The numbers in the triangular structure represent time steps when a reduction is performed and a build step is completed for that cell. In the final result cell $(1, N)$, in

this example $(1, 5)$ highlighted *green*, the direction of iteration of k will determine which cells are considered first for the final reduction. For this specific scheme, it is only legal to traverse k in the positive direction because the two cells that would be considered first with $-k$, highlighted *orange*, have not been completed. Only the *yellow* highlighted cells have been completed and produce the correct answer. This is the strategy for mapping proof - if a direction of iteration can be found to populate the final result cell legally with completed cells below it, the mapping is on track for success.

Illegal Mapping Patterns

There are a few general patterns for observing entire sets of space time mappings to be illegal. One of the most obvious is with k or $-k$ permuted to the outermost loop. In this iteration scheme, an entire pass through the memory structure is attempted without necessarily completing any dependent cells. This guarantees that when the final result, cell (1,N), is reached on the first pass, no comparison can be made to build its solution as neither of its dependent starting neighbors contain complete information.

The next clear illegal mapping is with $-d$ permuted to any loop level. When $-d$ is present, the iteration scheme will start step-wise computation at the final result, cell $(1, N)$. None of this cell's dependent neighbors contain information yet, so failure is inevitable. Some other mappings that begin iteration at the final result are any mappings containing $-j$ and $+i$ together, or mappings where $-j$ comes anywhere before $+d$ in the permutation. These mappings fail due to a lack of information available at the final time step of reduction computation - that is a simple rule of thumb for our visualization scheme, if the dependent cells referring to the starting value of k are incomplete at cell $(1, N)$ in either direction of k's traversal, the mapping is invalid.

Permuting k to the middle loop poses a similar problem of information not being available. Rather than a clear failure of all missing neighbors, these mappings depend on the direction of iteration of k. The mappings $j, -k, d$ and $j, -k, i$ are legal permutations because information has been completed in the dependent cells "behind" the iteration, in the $-k$ direction. If the direction of k is switched, making the mappings j, k, d and j, k, i, information is being pulled from "ahead" of the iteration where it is incomplete. A similar scenario occurs for the mappings $-i, k, j$ and $-i, k, -j$; if k is traversed backwards within these mappings as $-k$, then information is pulled from ahead of where it is completed.

Mappings

Using this information about the nature of relationship between the problem and its memory structure, the following truth table for legal mappings was created.

i,j,k	123	132	213	231	312	321
[+++]	[+i+j+k]	[+i+k+j]	[+j+i+k]	[+j+k+i]	[+k+i+j]	[+k+j+i]
[++-]	[+i+j-k]	[+i-k+j]	[+j+i-k]	[+j-k+i]	[-k+i+j]	[-k+j+i]
[+-+]	[+i-j+k]	[+i+k-j]	[-j+i+k]	[-j+k+i]	[+k+i-j]	[+k-j+i]
[+--]	[+i-j-k]	[+i-k-j]	[-j+i-k]	[-j-k+i]	[-k+i-j]	[-k-j+i]
[-++]	[-i+j+k]	[-i+k+j]	[+j-i+k]	[+j+k-i]	[+k-i+j]	[+k+j-i]
[-+-]	[-i+j-k]	[-i-k+j]	[+j-i-k]	[+j-k-i]	[-k-i+j]	[-k+j-i]
[--+]	[-i-j+k]	[-i+k-j]	[-j-i+k]	[-j+k-i]	[+k-i-j]	[+k-j-i]
[---]	[-i-j-k]	[-i-k-j]	[-j-i-k]	[-j-k-i]	[-k-i-j]	[-k-j-i]
d,j,k						
[+++]	[+d+j+k]	[+d+k+j]	[+j+d+k]	[+j+k+d]	[+k+d+j]	[+k+j+d]
[++-]	[+d+j-k]	[+d-k+j]	[+j+d-k]	[+j-k+d]	[-k+d+j]	[-k+j+d]
[+-+]	[+d-j+k]	[+d+k-j]	[-j+d+k]	[-j+k+d]	[+k+d-j]	[+k-j+d]
[+--]	[+d-j-k]	[+d-k-j]	[-j+d-k]	[-j-k+d]	[-k+d-j]	[-k-j+d]
[-++]	[-d+j+k]	[-d+k+j]	[+j-d+k]	[+j+k-d]	[+k-d+j]	[+k+j-d]
[-+-]	[-d+j-k]	[-d-k+j]	[+j-d-k]	[+j-k-d]	[-k-d+j]	[-k+j-d]
[--+]	[-d-j+k]	[-d+k-j]	[-j-d+k]	[-j+k-d]	[+k-d-j]	[+k-j-d]
[---]	[-d-j-k]	[-d-k-j]	[-j-d-k]	[-j-k-d]	[-k-d-j]	[-k-j-d]

From the OSP.cs auto_gen script

- Impossible (Significant Pattern)
- Fails testing (Lesser Pattern)
- Theroritically works, but fails testing
- Passes testing
- Passes testing, requires diagonal to be evaluated first

Figure 3: Table of Legal Memory Structure Mappings

With the table showing all possible legal traversals that solve the problem correctly finished, the next step is to determine the fastest for an implementation recommendation.

Methodology

To prove which mapping is the fastest, an experiment was constructed. The point of the experiment is to collect a range of tightly grouped repeatable timing data. The data must also take the machine's zero calibration into account. We found the zero calibration time of the department machines, specifically Berlin, to be on the order of tenths of milliseconds. This means problem sizes yielding run times greater than a few milliseconds are required.

Factors of 1000 immediately stuck out as appropriate problem sizes as 1000 produces run times ranging from tenths to hundredths of seconds depending on the mapping. A problem size of 5000 produces run times ranging from about ten seconds to several minutes depending on the mapping. The final data set selected was five trials on each of nine problem sizes between 1000 and 5000 in steps of 500 for each mapping. Confining the trials to Berlin controls peak roof-line variables.

The values of the matrices' dimensions were generated randomly to simulate a complicated problem in which it will be unlikely to find an easy parenthesization. This was facilitated by the random flag in the makefile which was configured to a special executable for this experiment's tests only. Correctness verification was done beforehand. Additionally, to get the most accurate timing data, the original timer in the wrapper script based on time of day was augmented to system time on the order of clock ticks. We are convinced this timer is more accurate and less prone to interruption scenarios.

The modern Intel C++ Compiler, *icc*, was used to generate the executable files for each mapping. This augmentation over *gcc* takes vectorization, fused multiply-add units and may other levels of optimization into account. This also enables the capture of vectorization reports for result analysis.

After the data was collected, outliers were identified and thrown out with a $Z - score$ based protocol. The data of these trials was found to be within 2% standard deviation, so throwing out two of the most extreme outliers from each trial did not significantly change the result of their average used for graphing and slope fitting, but this step guaranteed the data was tightly grouped.

Results

The averaged data was first plotted to determine its shape and order. A cubic order is expected. The exponential shape of the data invites log fitting to see if any of these mapping are of the cubic order, but already, a significant difference between mappings is evident. The raw data is plotted below for all 23 legal mappings.

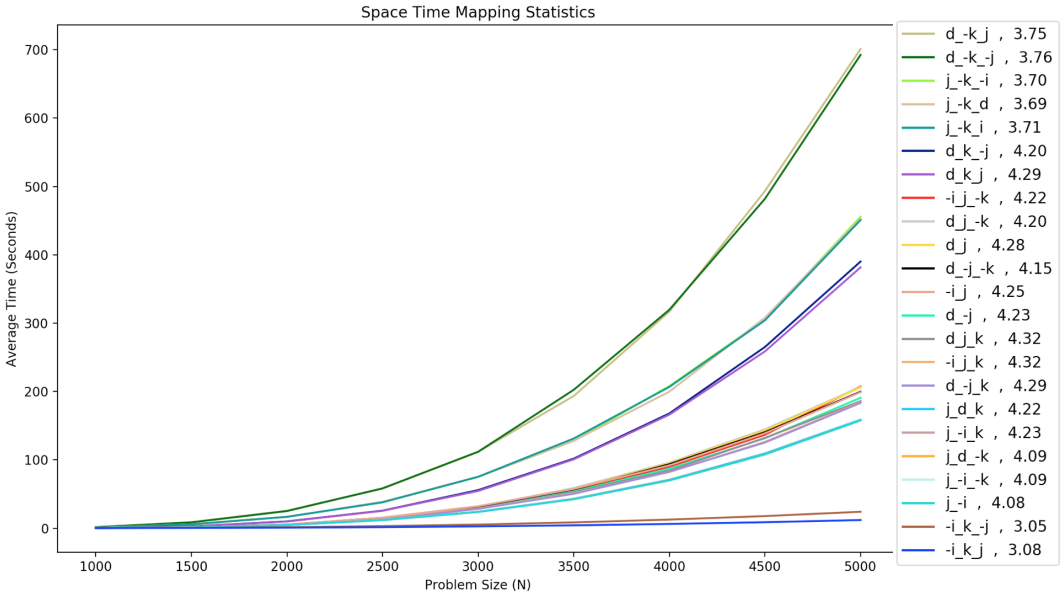

Figure 4: Raw Data Visualization of Run Time for all 23 Legal Mappings

There are about five visible trends or performance. The mappings close to each of these bands share an extent of vectorization and factors that contribute to their performance bound. Five mappings that represent these trends are shown below.

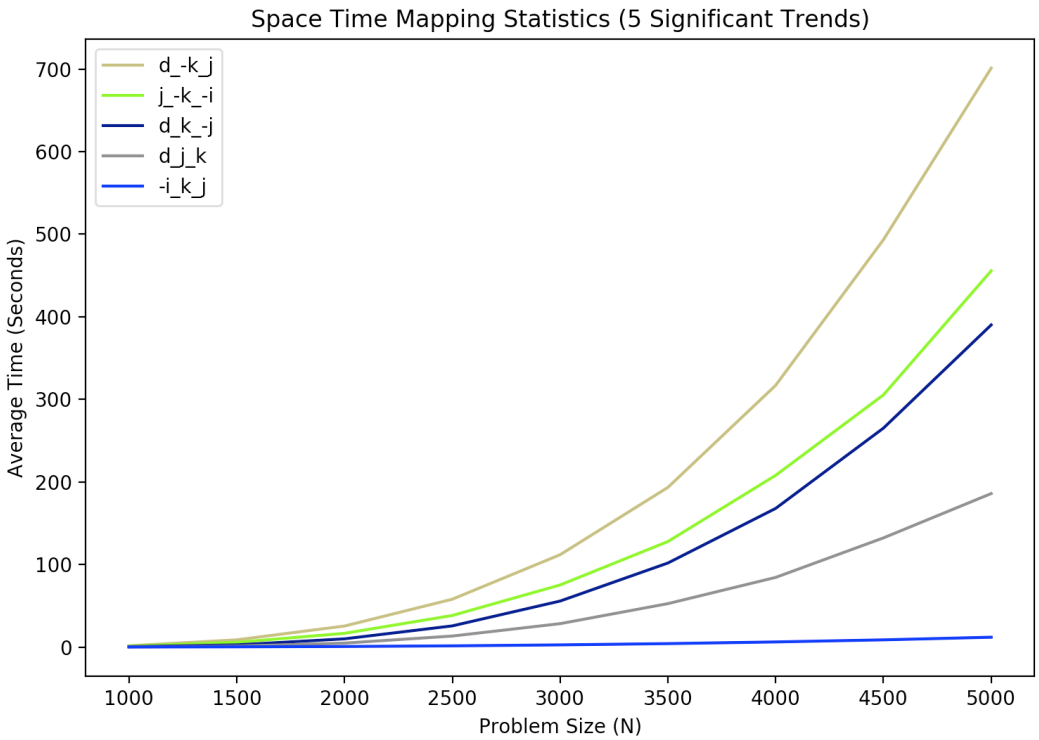

Figure 5: Raw Data Visualization of Run Time for 5 Significant Trends

Logarithmic scaling tells us the exponential order of the raw data via a linearly fitted slope. Steeper slopes and higher intercepts indicate worse performance. The intercept is specific to the problem size considered in the experiment, meaning that a variable is needed to represent initial overhead within the scope of the experiment that will be shadowed by growing problem size. The five trends scaled logarithmically as well as the full set of 23 logarithmically scaled mappings are shown below.

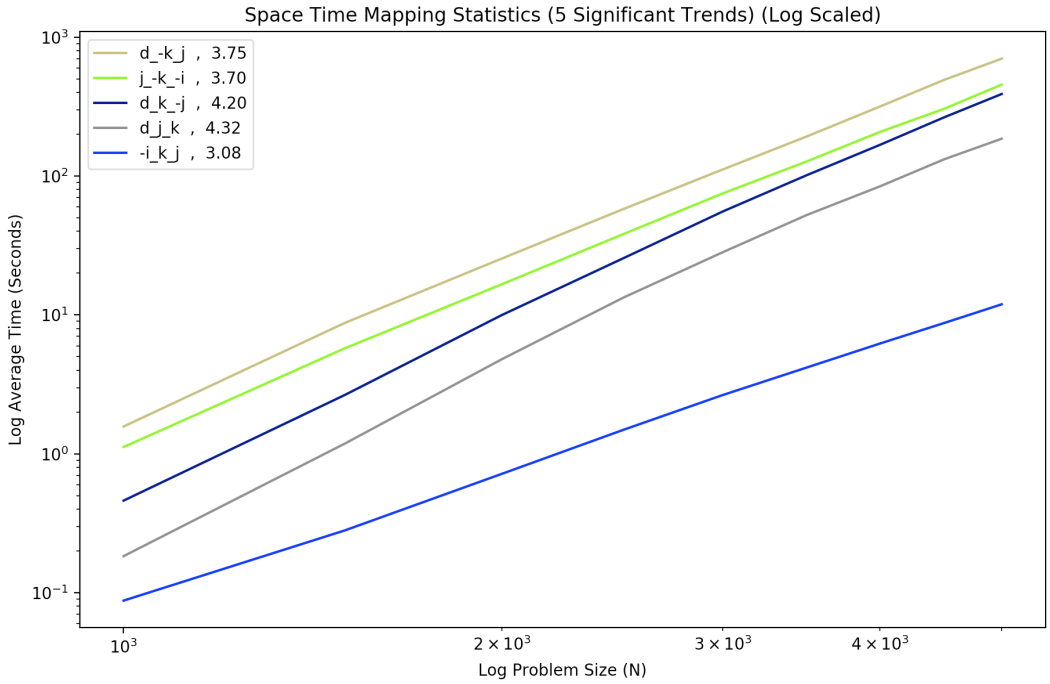

Figure 6: Log Scaled Data Visualization of Run Time for 5 Significant Trends

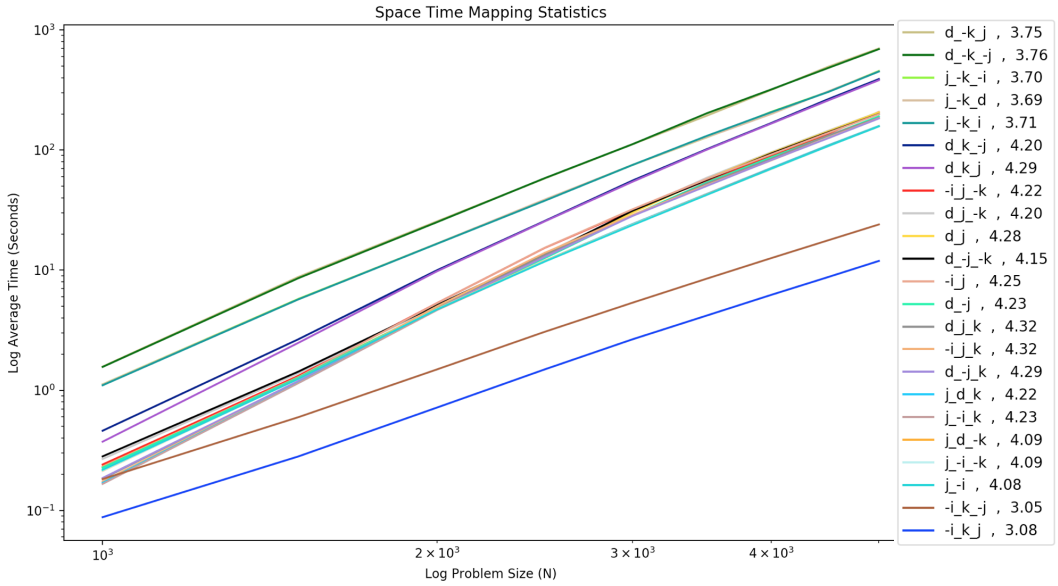

Figure 7: Log Scaled Data Visualization of Run Time for all 23 Legal Mappings

The slope of each logarithmically fitted mapping tells us its exponential order of performance. It is observed that the order of table entries sorted by log slope does not match that of the log scaled graph. This because the table is sorted solely on log slop to portray a result related to arbitrarily large problem size. The intercept has an effect specific to this set of problem sizes - the intercept can be viewed as overhead that has a diminishing effect as problem size increases. A tabular leader-board sorted by log slope is shown below.

Mapping	Log Slope	Log Intercept
-i_k_-j	3.059	-22.847
-i_k_j	3.088	-23.786
j_-k_d	3.699	-25.347
j_-k_-i	3.704	-25.384
j_-k_i	3.713	-25.456
d_-k_j	3.752	-25.346
d_-k_-j	3.761	-25.415
j_-i	4.089	-29.648
j_-i_-k	4.093	-29.682
j_d_-k	4.094	-29.689
d_-j_-k	4.151	-29.92
d_k_-j	4.204	-29.728
d_j_-k	4.206	-30.348
-i_j_-k	4.226	-30.525
j_d_k	4.227	-30.768
j_-i_k	4.232	-30.817
d_-j	4.233	-30.653
-i_j	4.251	-30.728
d_j	4.284	-31.003
d_-j_k	4.295	-31.197
d_k_j	4.298	-30.517
-i_j_k	4.324	-31.421
d_j_k	4.325	-31.417

Figure 8: Table of Exponential Orders for each Mapping

Next, the results were converted to $GigaFlops/Second$. Operational speed is dependent on problem size. A trend is observed that as problem size increases, operational speed decreases indicating an increase in memory boundedness. This trend is not followed by all mappings, though - some have what seems to be constant operational speed despite problem size likely indicating extensive optimization of the memory bound roofline. Also, these factors are heavily influenced by their vectorization report. More vectorization is found to yield higher operational speed.

Mappings	GigaFlops/Second for each Problem Size (N) (Sorted on 1000)									Average
	1000	1500	2000	2500	3000	3500	4000	4500	5000	
d_-k_j	0.53	0.321	0.262	0.225	0.201	0.185	0.168	0.154	0.149	0.244
d_-k_-j	0.533	0.328	0.266	0.224	0.201	0.176	0.167	0.158	0.15	0.245
j_-k_-i	0.744	0.489	0.401	0.34	0.299	0.28	0.256	0.249	0.229	0.365
j_-k_d	0.745	0.493	0.4	0.337	0.3	0.279	0.267	0.247	0.23	0.366
j_-k_i	0.757	0.493	0.402	0.344	0.299	0.272	0.258	0.25	0.231	0.367
d_k_-j	1.81	1.06	0.668	0.508	0.404	0.351	0.318	0.286	0.267	0.63
d_k_j	2.233	1.135	0.682	0.512	0.413	0.355	0.321	0.294	0.273	0.691
d_-j_-k	2.96	1.97	1.304	0.969	0.721	0.632	0.57	0.541	0.522	1.132
d_j_-k	3.108	2.032	1.326	0.948	0.728	0.607	0.556	0.525	0.504	1.148
-i_j_-k	3.46	2.136	1.254	0.851	0.71	0.64	0.592	0.556	0.502	1.189
d_j	3.598	2.195	1.382	0.965	0.746	0.622	0.563	0.531	0.506	1.234
-i_j	3.619	2.189	1.256	0.849	0.7	0.621	0.58	0.547	0.524	1.209
d_-j	3.677	2.21	1.385	1.036	0.788	0.664	0.612	0.577	0.546	1.277
j_-i	3.824	2.262	1.416	1.106	0.949	0.843	0.762	0.703	0.661	1.392
j_-i_-k	3.884	2.263	1.406	1.098	0.942	0.84	0.767	0.708	0.66	1.396
j_d_-k	3.901	2.249	1.397	1.091	0.94	0.838	0.767	0.703	0.66	1.394
d_-j_k	4.48	2.362	1.399	1.013	0.796	0.711	0.646	0.606	0.568	1.398
d_j_k	4.543	2.373	1.389	0.975	0.794	0.679	0.632	0.574	0.56	1.391
-i_k_-j	4.575	4.717	4.472	4.25	4.199	4.202	4.234	4.286	4.337	4.364
-i_j_k	4.785	2.41	1.333	0.936	0.781	0.701	0.65	0.6	0.566	1.418
j_d_k	4.919	2.445	1.426	1.09	0.929	0.828	0.752	0.694	0.656	1.527
j_-i_k	5.018	2.456	1.434	1.097	0.937	0.832	0.756	0.7	0.659	1.543
-i_k_j	9.496	10.011	9.291	8.736	8.472	8.524	8.565	8.673	8.728	8.944

Figure 9: Timing Data Converted to Operational Speed Sorted on Low Problem Size

Mapping	GigaFlops/Second for each Problem Size (N) (Sorted on 5000)									Average
	1000	1500	2000	2500	3000	3500	4000	4500	5000	
d_-k_j	0.53	0.321	0.262	0.225	0.201	0.185	0.168	0.154	0.149	0.244
d_-k_-j	0.533	0.328	0.266	0.224	0.201	0.176	0.167	0.158	0.15	0.245
j_-k_-i	0.744	0.489	0.401	0.34	0.299	0.28	0.256	0.249	0.229	0.365
j_-k_d	0.745	0.493	0.4	0.337	0.3	0.279	0.267	0.247	0.23	0.366
j_-k_i	0.757	0.493	0.402	0.344	0.299	0.272	0.258	0.25	0.231	0.367
d_k_-j	1.81	1.06	0.668	0.508	0.404	0.351	0.318	0.286	0.267	0.63
d_k_j	2.233	1.135	0.682	0.512	0.413	0.355	0.321	0.294	0.273	0.691
-i_j_-k	3.46	2.136	1.254	0.851	0.71	0.64	0.592	0.556	0.502	1.189
d_j_-k	3.108	2.032	1.326	0.948	0.728	0.607	0.556	0.525	0.504	1.148
d_j	3.598	2.195	1.382	0.965	0.746	0.622	0.563	0.531	0.506	1.234
d_-j_-k	2.96	1.97	1.304	0.969	0.721	0.632	0.57	0.541	0.522	1.132
-i_j	3.619	2.189	1.256	0.849	0.7	0.621	0.58	0.547	0.524	1.209
d_-j	3.677	2.21	1.385	1.036	0.788	0.664	0.612	0.577	0.546	1.277
d_j_k	4.543	2.373	1.389	0.975	0.794	0.679	0.632	0.574	0.56	1.391
-i_j_k	4.785	2.41	1.333	0.936	0.781	0.701	0.65	0.6	0.566	1.418
d_-j_k	4.48	2.362	1.399	1.013	0.796	0.711	0.646	0.606	0.568	1.398
j_d_k	4.919	2.445	1.426	1.09	0.929	0.828	0.752	0.694	0.656	1.527
j_-i_k	5.018	2.456	1.434	1.097	0.937	0.832	0.756	0.7	0.659	1.543
j_-i_-k	3.884	2.263	1.406	1.098	0.942	0.84	0.767	0.708	0.66	1.396
j_d_-k	3.901	2.249	1.397	1.091	0.94	0.838	0.767	0.703	0.66	1.394
j_-i	3.824	2.262	1.416	1.106	0.949	0.843	0.762	0.703	0.661	1.392
-i_k_-j	4.575	4.717	4.472	4.25	4.199	4.202	4.234	4.286	4.337	4.364
-i_k_j	9.496	10.011	9.291	8.736	8.472	8.524	8.565	8.673	8.728	8.944

Figure 10: Timing Data Converted to Operational Speed Sorted on High Problem Size

Conclusion

Bibliography

[1] Thomas Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. "Introduction to Algorithms, Third Edition". *2009*
`https://ms.sapientia.ro/~kasa/Algorithms_3rd.pdf`