

The Effect of Sparse Matrix Multiplication on Analytical Ultracentrifugation Analysis Using Ultrascan Software

Zachary Adam Ozer

Abstract

Analytical ultracentrifugation (AUC) is a powerful technique for determining self-association and interaction properties of macromolecules within solution. In order to characterize the interaction and association of a sample, AUC scan data must be gathered from a wide array of experiments and then inserted into a matrix, whereupon a nonlinear least squares (NLS) algorithm is used to fit the parameters globally. However, not all datasets contain data for all parameters. In experiments for which the value of the variable is undetermined, it is assigned a value of zero, thus ensuring that it does not contribute to the fitted value. In most cases, this results in a sparse data matrix, that is, one that is populated mainly by the zeros. Yet, the NLS algorithm is an extremely time consuming process. Each improvement in the fit requires that the entirety of the data matrix be multiplied by its transpose, an operation which entails the application of a series of mathematical operations to every entry within the matrix, including the vast numbers of zeros. The purpose of this project was to optimize the efficiency of analysis for analytical ultracentrifugation data, specifically as it is utilized in the Ultrascan software. The hypothesis of this experiment stated that substituting a highly repetitive, but computationally simple matrix multiplication algorithm with a more complex process, such as a sparse matrix multiplication algorithm, would provide a notable improvement in efficiency, as only entries which contribute to the NLS algorithms fit will be processed.

The resulting improvement in overall efficiency was staggering. For the largest dataset, one involving 38,121 data points, the original algorithm required an average of 87,797.9 ms per iteration with a standard deviation of 57,388.5 ms per iteration. The sparse matrix algorithm, alternatively, required only 737.56 ms per iteration with a standard deviation of 309.04. Furthermore, the time per operation of the sparse matrix algorithm was one third of that used by the original algorithm. Although the algorithmic improvements appear quite impressive, the conversion to a sparse matrix data structure yielded results that are even more phenomenal. For the largest dataset, 38,121 data points, the compressed data required two orders of magnitude less space than the uncompressed data, dropping from 46.05 megabytes to 761.62 kilobytes. Although it is difficult to attribute the improvement in efficiency to either the data structure or the improved multiplication algorithm, since both are required for either to perform their function, it seems that the benefit of the new implementation is a result of the improved data structure.

I. INTRODUCTION

Analytical ultracentrifugation (AUC) is a powerful technique for determining self-association and interaction properties of macromolecules in solution.¹ AUC allows for the observation of a sample in its native state, thus removing the potential effects of preparation and surface interactions often associated with the use of investigatory tools.² These complications are overcome in the very design of the AUC apparatus, which is little more than a preparative ultracentrifuge used in combination with an optical detection system (Fig. 1).³ The samples and a reference buffer are loaded into the centerpiece of the centrifuge cell, which is then sandwiched in between two viewing windows (Fig. 2). This stack is then secured within the cell housing (Fig. 3), loaded into a rotor, and spun at speeds between 3,000 and 60,000 rpm. The optical detection system then records the interference or absorbance of the samples at systematic radial distances from the center of the cell at a specific wavelength, all the while, the sample remains spinning (Fig. 1). These scans provide data about the thermodynamic properties of the sample, which can then be analyzed in order to determine other characteristics.²

Two types of experiments can be performed using an AUC: sedimentation velocity and sedimentation equilibrium.⁴ In sedimentation velocity experiments, the rotor of the AUC is set to very high speeds, thus causing the solute to sediment towards the bottom of the cell. A solute concentration gradient, the boundary, is created between the meniscus and the region of uniform concentration, above which there is no material (Fig. 4). The shape and evolution of the boundary provides information about the molecular weight and shape of the components in the solution, as the boundary moves towards the bottom of the cell at a rate proportional to the sedimentation coefficients of the sample components. Sedimentation equilibrium experiments, alternatively, are able to distinguish between molecules of differing masses, but do not differentiate based on shape. The apparatus is spun at relatively low speeds in order to balance the effect of diffusion with that of sedimentation, thus establishing a stable concentration gradient (Fig. 5). Observing this gradient at multiple speeds and load concentrations allows the determination of the molecular mass of the sample, both natively and as an associative system.

When trying to characterize the interaction and self-association of a sample, both types of scans are necessary, along with a great deal of sophisticated analysis. Equilibrium data can provide information about the average molecular weight, homogeneity, and thermodynamic non-ideality while velocity experiments are able to determine the molecular shape as well as whether conformational changes occurred.⁵ However, in order to characterize the interaction and association of a sample, this information must be gathered from a wide array of experiments performed at multiple load concentrations and rotor speeds. This data is then inserted into a matrix, whereupon a nonlinear least squares (NLS) algorithm is used to fit the parameters globally, meaning that the best variable values are found by fitting the data from all datasets.⁶ However, not all datasets contain data for all parameters, resulting in the creation of local variables. Thus, in experiments for which the value of the local variable is undetermined, they are assigned a value of zero, thus ensuring that they do not contribute to the fitted value. In most cases, there are very few global and many local variables. This results in a sparse data matrix, that is, one that is populated mainly by the zeros.

The NLS algorithm is an extremely time consuming process, as each improvement in the fit requires that the entirety of the data matrix be multiplied by its transpose. The multiplication of a matrix by its transpose entails the application of a series of mathematical operations to every entry within the matrix.⁷ In order for the processor to perform these operations, the data must first be available in low-level, high-speed memory.⁸ Unfortunately, low-level memory has very limited storage capabilities, and once it fills up, it moves the least accessed data back into high-level memory. Thus, for processes wherein no single piece of data is accessed very often, there are frequent exchanges of data from low-level memory to high-level memory, a process which is very time intensive. In addition, the processor becomes underutilized during these exchanges periods, as it must wait for the incoming data before it is able to begin processing it. Furthermore, the processing time increases based on what type of operations are being performed. Notably, multiplication operations are a great deal more complex than addition, subtraction, or comparison operations, and thus more time consuming.

I. PURPOSE

The purpose of this project is to optimize the efficiency of analysis for analytical ultracentrifugation data, specifically as it is utilized in the Ultrascan software. I will observe the effect of substituting a highly repetitive, but computationally simple matrix multiplication algorithm with a more complex process, designed to perform fewer operations by taking advantage of known characteristics of the data.

I. HYPOTHESIS

The hypothesis of this experiment states that the use of a sparse matrix multiplication algorithm will provide a notable improvement in efficiency, as only entries which contribute to the NLS algorithms fit will be processed. In addition, the vast majority of the operations will be fast comparison operations, thus minimizing the number of slower multiplication operations and further improving efficiency.

I. RESULTS

The resulting improvement in overall efficiency was staggering. For the smallest dataset, which contained 3,836 data points, the sparse matrix algorithm required only 12.5 milliseconds per iteration, compared with the 141.5 ms per iteration required for the original algorithm (Fig.s 6, 7, 8; Table 1). In the same 141.5 ms, the sparse matrix algorithm was able to process 11,305 data points, an operation which required 18017.5 ms using the original algorithm, with a notable 1726.05 ms standard deviation. For the largest dataset, one involving 38,121 data points, the original algorithm required an average of 87,797.9 ms per iteration with a standard deviation of 57,388.5 ms per iteration. The sparse matrix algorithm, alternatively, required only 737.56 ms per iteration with a standard deviation of 309.04. Furthermore, the time per operation of the sparse matrix algorithm was one third of that used by the original algorithm. This is somewhat misleading, however, as the original algorithm operated on much larger matrices. Operating on matrices of the same size, the sparse matrix algorithm proved far

less efficient than the original algorithm, requiring more than five times more time per entry. Interestingly, both algorithms seem to show a plateau in their time per operation curves (Figures 10, 11).

Although the algorithmic improvements appear quite impressive, the conversion to a sparse matrix data structure yielded results that are even more phenomenal. Expectedly, the conversion significantly increased the program's overhead, accounting for approximately one quarter of the total operation time, on average (Table 2). Excluding the smallest dataset, the data matrices being analyzed were 97.45% sparse, on average, and became increasingly sparse with increasingly large datasets. This meant that data for the largest dataset, 38,121 data points, which initially represented a matrix of approximately 11,512,500 entries, the sparse matrix representation of the data required only 190,405 entries. As a result, the compressed data required two orders of magnitude less space than the uncompressed data, dropping from 46.05 megabytes to 761.62 kilobytes. Quite surprisingly, besides requiring orders of magnitude more storage space overall, the standard matrix structure did not exhibit linear growths in size with linear increases in dataset size, while the sparse matrix structure did (Figures 8, 9).

I. DISCUSSION

While it is clear that the switch to a sparse matrix structure provided a huge increase in efficiency, it is difficult to attribute this improvement solely to the new data structure or the improved multiplication algorithm, since both are required for either to perform their function. However, it seems clear that this improvement is not a result of increased algorithmic efficiency. While initial tests seemed to indicate that the enhanced multiplication algorithm exhibited a complexity of $O(\ln(n))$ and thus had a plateau, in fact, it is far less efficient than the original algorithm, even though its complexity is of the same magnitude, $O(n^3)$. Overall, the enhanced algorithm is over 5 times less efficient than the original algorithm, both in terms of overall time required to process a similarly sized matrix, and in terms of time per operation. However, it seems that the enhanced algorithm's utilization of comparisons operations over multiplication operations provides an increase in efficiency when the size of the matrix grows larger, as the plateau for the sparse algorithm's time per operation was one third of that of the standard algorithm.

Thus, it seems that the benefit of the new implementation comes from the use of the sparse matrix data structure. Despite the increase in inefficiency of the multiplication algorithm, the massive reduction in the size of the data to operate on is the primary cause of the decreased processing time. However, the benefit derived from compressing the data structure seems to be the result of more than a simple reduction in the number of operations and the use of faster operations. The significant increase in the standard deviation of the time per iteration of the sparse matrix algorithm for data sets involving more than about 20,000 AUC data points seems to indicate that the size of the sparse matrix is exceeding the amount of low-level memory available. This assessment is corroborated by the non-linear increases in the standard matrix structure, as there is some memory overhead associated with utilizing high-level memory. Thus, the sparse matrix algorithm ensures that the processor remains fully loaded and that no time is lost as data is moved in between low-level and high-level memory.

I. FUTURE WORK

It appears that a great deal of improvement stands to be made in the sparse matrix multiplication algorithm, as both the new and original algorithm show a complexity of $O(n^3)$. The most efficient known process for matrix multiplication is the Coppersmith-Winograd algorithm, which has a complexity of $O(n^{2.376})$, and rigorous proof has demonstrated that the lower limit of matrix multiplication complexity is $O(n^2)$. Thus, while implementing a Coppersmith-Winograd algorithm would achieve some boost in multiplication efficiency, this improvement is finite. In fact, the time spent performing matrix multiplication no longer represents a majority of the time spent performing the NLS analysis. Other processes such as display updates dominate the processor utilization. Before more work is done to the matrix multiplication procedure, these processes should be identified and their algorithms streamlined.

Although the updated implementation of a sparse matrix algorithm and corresponding data structure provided an increase in efficiency of several orders of magnitude, there is a significant latency associated with converting the data into this format, accounting for approximately one quarter of the total sparse matrix algorithm time. The conversion process, however, is perfectly suited towards threading, that is, on a computer which utilizes multiple processors, the workload may be distributed, thus minimizing the conversion time. However, this only benefits those using the most sophisticated computer equipment. If the data were stored in a sparse matrix format initially, it would be possible to eliminate the need for conversion entirely. This shift would also require that all other operations done to the matrix be made to utilize a sparse matrix format. Complications arise, as there is no guarantee that the data is generated sequentially. Since no matrix multiplication algorithm acts efficiently in the absence of a sequential matrix, a time-consuming sort operation would have to be performed first, most likely offsetting the gain of the initial conversion bypass.

I. ACKNOWLEDGMENTS

First, I would like to thank the entire UTHSCSA Department of Biochemistry, its faculty and staff, for their time, the use of their labs and equipment, and their support of the B-SURE program. The organizers of the program, Dr. John Hart, Dr. Bruce Nicholson, and Anna Uriegas, deserve special recognition, as they were not only the ones who provided us with food, but also their dedication and their encouragement of our endeavors as scientists. I would especially like to thank my PI, Dr. Borries Demeler, Dr. Stephen Hardies, Jeremy Mann, Yu Ning, and Virgil Schirf, for their assistance and their wisdom. This project would have been impossible without their constant guidance. I would also like to express my gratitude to NIH for making this experience possible by funding the program. One of the most overlooked groups is the teachers, at MIT and Keystone School, who provided me with the education necessary to participate in such a program. Finally, I would like to thank my family and friends, who always have encouraged me to endeavor to achieve.

I. FIGURES (SEE NEXT PAGE)

I. BIBLIOGRAPHY

- ¹ Schuck, Peter. "On the analysis of protein self-association by sedimentation velocity analytical ultracentrifugation." Analytical Biochemistry September 2003: 104-124.
- ² Cole, James L., and Jeffrey C. Hansen. "Analytical Ultracentrifugation as a Contemporary Biomolecular Research Tool." Journal of Biomolecular Techniques 10.4 (1999): 163-176.
- ³ Schirf, Virgil. "Analytical Ultracentrifuge." Analytical Ultracentrifugation Workshop. San Antonio, Texas. 02 August 2005.
- ⁴ Demeler, Borries. "Focus on Problem Solving - From Experimental Design to Data Analysis." Analytical Ultracentrifugation Workshop. San Antonio, Texas. 02 August 2005.
- ⁵ Laue, Thomas M. *Short Column Sedimentation Equilibrium Analysis for Rapid Characterization of Macromolecules in Solution*. Beckman Coulter Application Information DS-835, 1992.
- ⁶ Johnson, M.L., Correia, J.J., Yphantis, D.A., and Halvorson, H.R. "Analysis of data from the analytical ultracentrifuge by nonlinear least squares techniques." Biophys J 36 (1981): 575-588.
- ⁷ Ellard, Dan. "Operations on Sparse Matrices." 21 July 1997. Harvard Electrical Engineering and Computer Science. 2 Aug. 2005
<<http://www.eecs.harvard.edu/~ellard/Q-97/HTML/root/node20.html#mosparse>>.
- ⁸ Terman, Chistopher. "Virtual Memory." Computational Structures. Cambridge, Massachusetts. 07 April 2005.

http://bioc.rice.edu/bios576/AU/AU%20Page_files/image022.jpg

Fig. 1 – Analytical Ultracentrifuge Diagram..

http://www.embl-hamburg.de/~geerlof/webPP/protocoldb/Characterization/AUC_protocol/window_ass_12mm.jpg

Fig. 2 – Centerpiece Sandwiched Between Window Assemblies.

From

http://www.embl-hamburg.de/display?file=~geerlof/webPP/protocoldb/Characterization/AUC_protocol/auc_cell_assembly.html

http://www.embl-hamburg.de/display?file=~geerlof/webPP/protocoldb/Characterization/AUC_protocol/auc_cell_assembly.html

Fig. 3 – Cell Housing.

<http://www.cauma.uthscsa.edu/velocity.png>.

Fig. 4 – Sedimentation Velocity Experiment: a moving boundary is generated because the rotor speed is large enough to prevent back diffusion from the cell bottom to influence the absorbance near the meniscus, which will deplete over time.

<http://www.cauma.uthscsa.edu/equilibrium.png>.

Fig. 5 – Sedimentation Equilibrium Experiment: the rotor speed is slow enough such that the back diffusion at the bottom of the cell will balance out the sedimentation and form an equilibrium gradient.

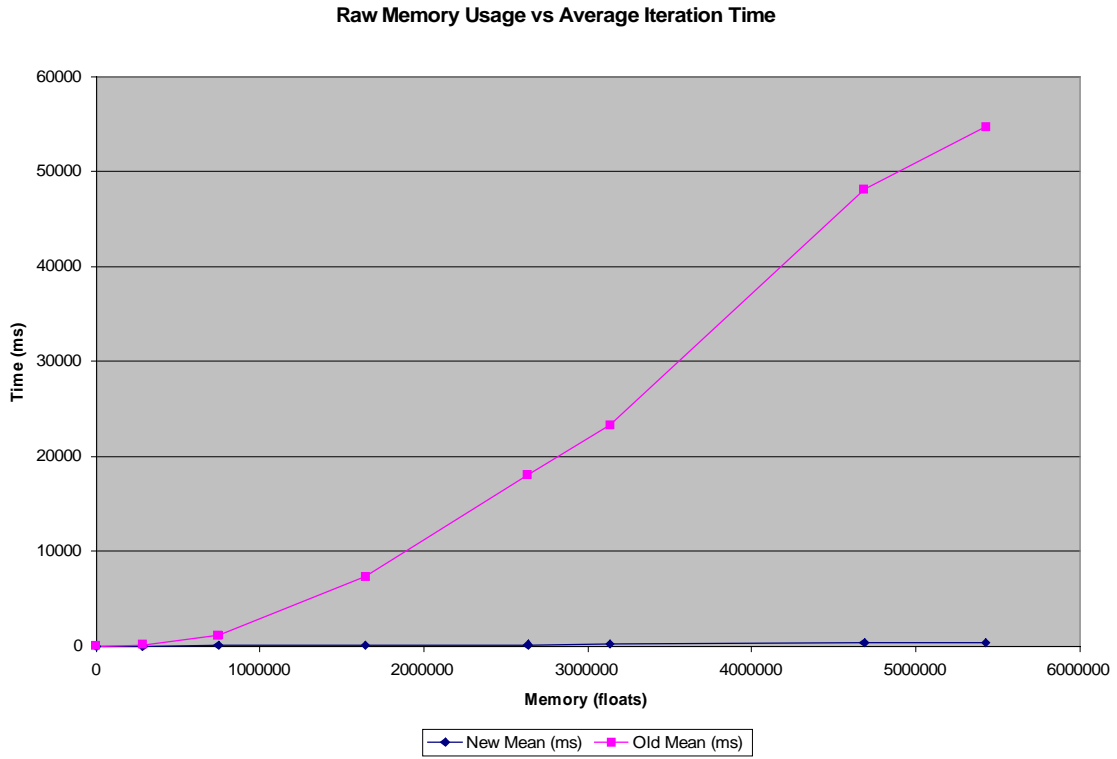


Fig. 6 – Raw Memory Usage. The time required by the new algorithm is significantly less.

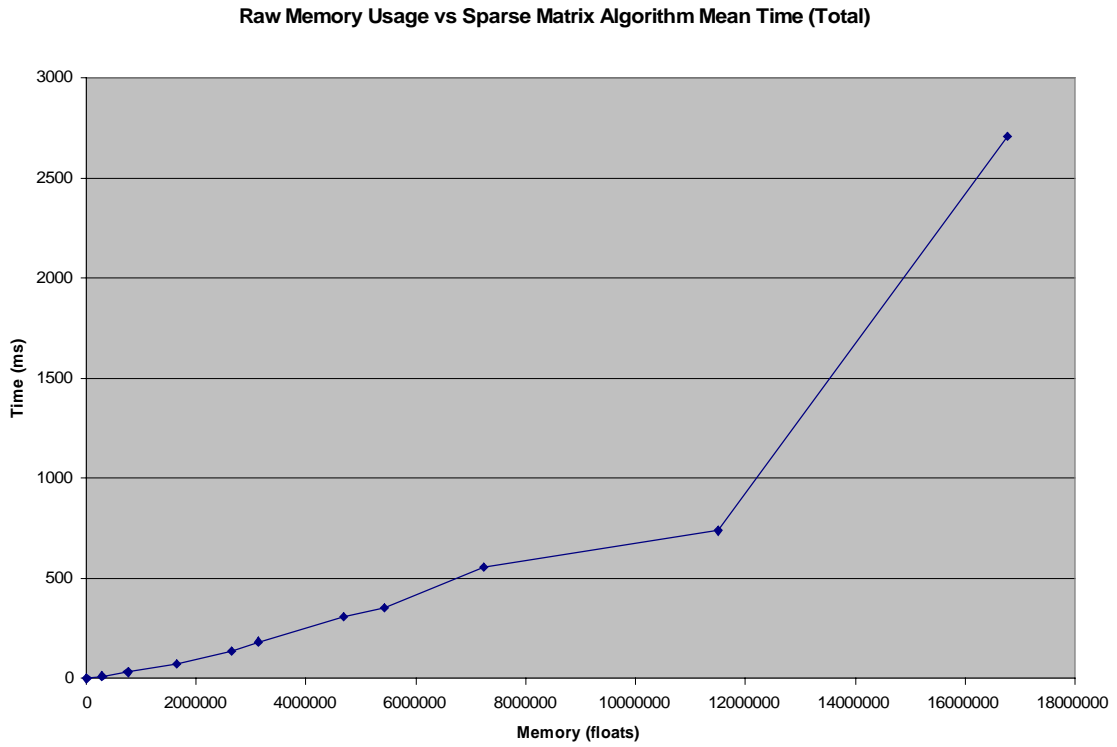


Fig. 7 – Raw Memory Usage vs Sparse Matrix Algorithm Time. Complexity is $O(n^3)$.

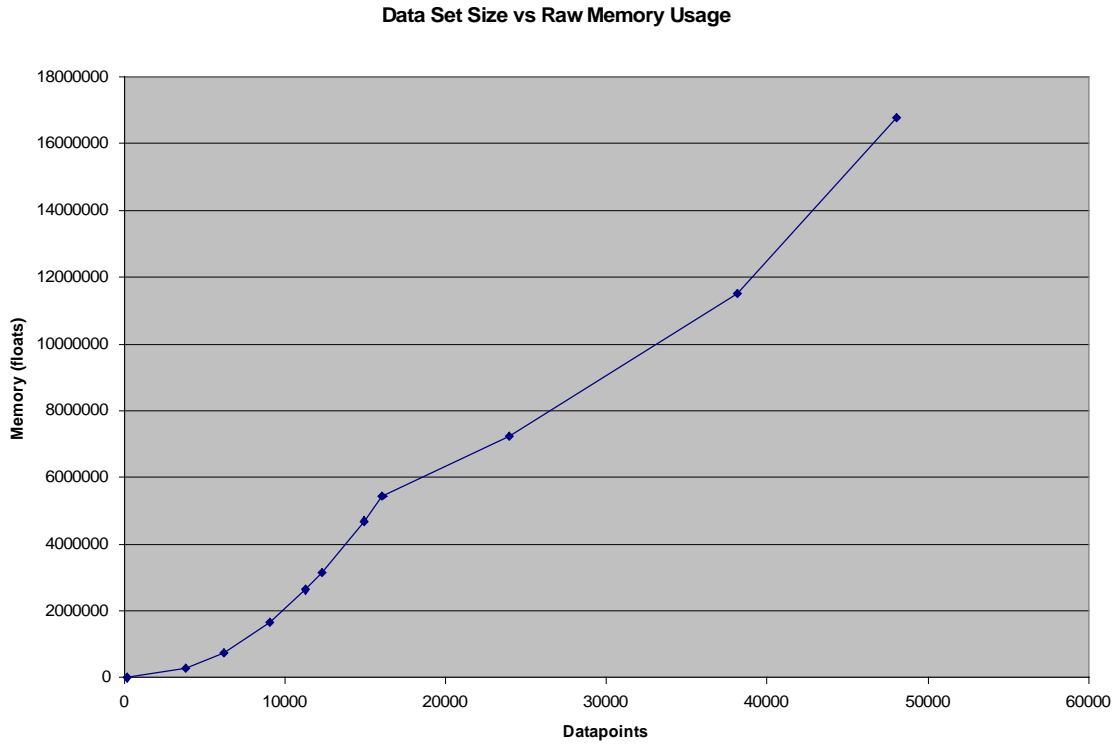


Fig. 8 – Data Set Size vs. Raw Memory Usage. Strangely, memory requirements do not grow linearly.



Fig. 9 – Data Set Size vs. Raw Memory Usage. As expected, memory requirements do grow linearly.

Raw Matrix Size vs Multiplication Algorithm Time per Entry

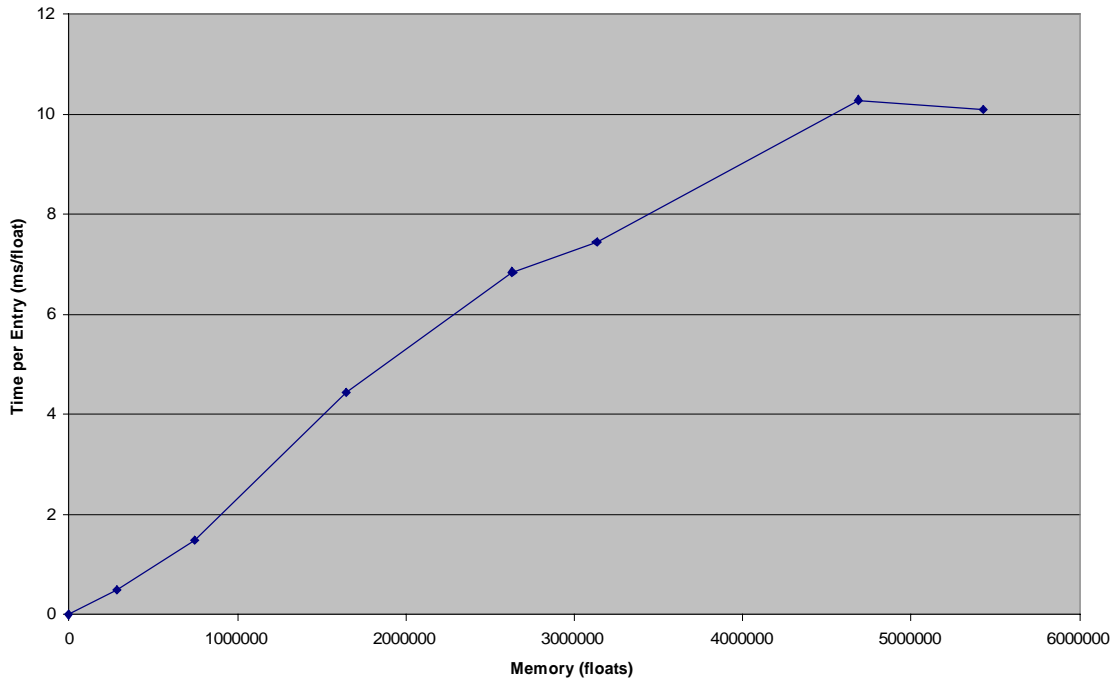


Fig. 10 – Raw Matrix Size vs Multiplication Time per Entry. Interestingly, there appears to be a limit.

Sparse Matrix Size vs Multiplication Algorithm Time per Entry

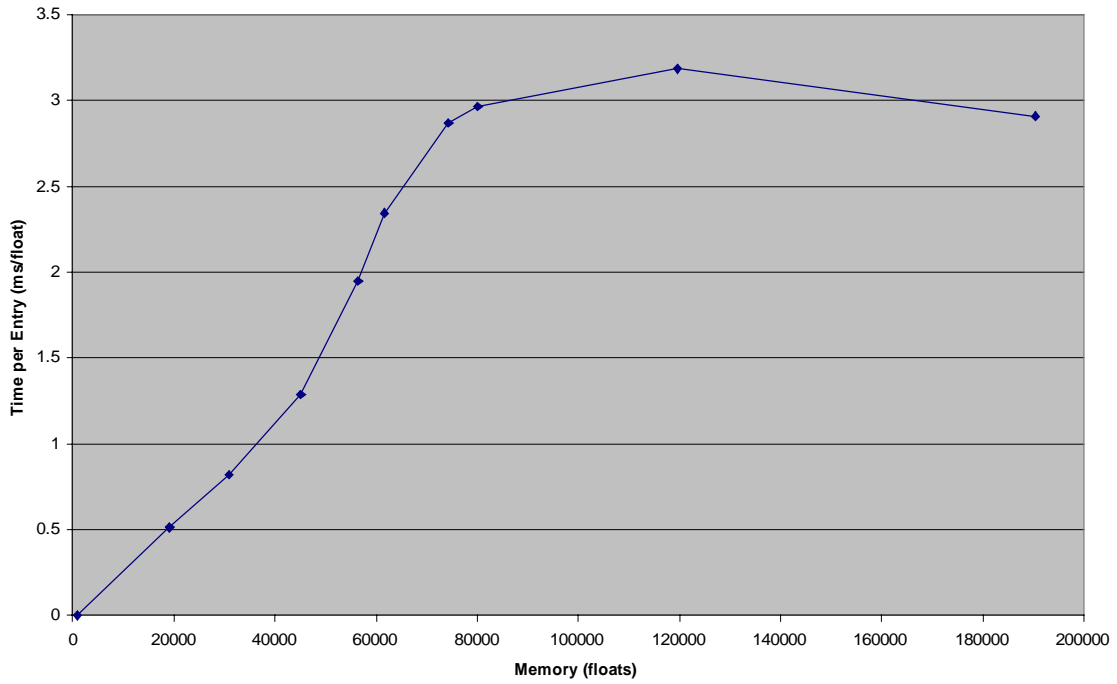


Fig. 11– Sparse Matrix Size vs Multiplication Time per Entry. Again, there appears to be a limit.

Table 1 – Time Utilization

Data Points	New Mean (ms)	Old Mean (ms)	New Std Dev	Old Std Dev	New Time per Entry (ms)	Old Time per Entry (ms)
179	0	0	0	0	0	0
3836	12.5	141.5	0.707107	0.707107	0.513350856	0.498478144
6184	32	1106	0	0	0.819168544	1.483671564
9032	74	7283	0	0	1.287004144	4.4304529
11305	135.5	18017.5	2.12132	1726.05	1.945867134	6.840201058
12336	184	23313.2	5	705.75	2.344294564	7.440367148
14919	308	48151.5	9.8995	2126.27	2.866612106	10.27626609
16060	351	54754	1.4142	8.48528	2.962999104	10.08695332
38121	737.56	87797.9	309.037	57388.5	2.908560777	7.626310532

Table 2 – Memory Utilization

Data Points	New Memory (floats)	Old Memory (floats)	New Size (kb)	Old Size (kb)	Size Ratio (Old / New)	% Sparse
179	893	895	3.572	3.58	1.002239642	0.223463687
3836	19132	283864	76.528	1135.456	14.83713151	93.26015275
6184	30840	745448	123.36	2981.792	24.17146563	95.8628905
9032	45040	1643850	180.16	6575.4	36.49755773	97.26009064
11305	56371	2634060	225.484	10536.24	46.7272179	97.85991967
12336	61512	3133340	246.048	12533.36	50.93867863	98.03685524
14919	74307	4685700	297.228	18742.8	63.05866204	98.41417504
16060	80076	5428200	320.304	21712.8	67.7881013	98.52481486
23961	119605	7236200	478.42	28944.8	60.50081518	98.34712971
38121	190405	11512500	761.62	46050	60.46322313	98.34610206