Research Proposal: Asynchronous GPU Matrix Inversion

Samuel J. Monson

May 27, 2024

1 Problem Statement

Matrix inversion is a fundamental component of Linear Algebra that has wide practical application in nearly all math-adjacent fields. Most notably, it has proven to be the best algebraic method for solving linear equations on a computer [1]. This has made matrix inversion a critical component of everything from Computer Graphics to Machine Learning [2].

However, a significant drawback of matrix inversion lies in its computationally intensive nature. Traditional methods such as Gauss-Jordan [3] and LU-Decomposition [1] possess a multiplicative asymptotic time complexity of $O(n^3)$ for any given $n \times n$ matrix. In 1969 Strassen [4] introduced a new method that brought the complexity down to $O(n^{2.808})$. Following this discovery, various groups competed to improve on Strassen's results culminating in the Coppersmith and Winograd method [5] which achieved a time complexity of $O(n^{2.376})$. Theoretically, the best sequential cost achievable for matrix inversion is $O(n^2)$ [6]; however, no general algorithm has been found that achieves this cost.

While no general $O(n^2)$ inversion method has been found, there exist various methods that optimize for specific matrix types. Triangular matrices can be inverted using Gaussian Elimination in $O(n^2)$ time [1]. The inverse of an orthogonal matrix is its transpose [2], thus the time complexity to invert an orthogonal matrix corresponds to the number of swap operations required to flip rows and columns, specifically $\frac{n^2}{2} - n$. Alternatively, if the matrix is simply read from memory in the new order, the time complexity is O(1). Cholesky Decomposition [1] improves on LU-Decomposition for symmetric, positive-definite matrices and has a time complexity of $O\left(\frac{3}{4}n^3 + \frac{3}{2}n^2\right)$. QR-decomposition [1] is slower then LU in most cases excluding where the inverses of multiple similar matrices are needed. For example, when we have a series of matrices where $A_{i+1} = A_i + B$ intermediate steps of the QR process of A_i can be reused for A_{i+1}.

The Graphics Processing Unit (GPU) is a specialized co-processor originally created for the task of translating data representations into computer graphics. As modern display technology consists of a grid of independent points, GPUs have evolved to excel at embarrassingly parallel workloads. Due to this specialization, GPUs have found significant success outside of their originally designed function, particularly in scientific fields where extensive independent computation is crucial [7]. Since a grid of points is essentially a physical example of a matrix, GPUs are especially well-suited for operations involving matrices.

2 Related Work

2.1 Strassen Matrix Inversion

In 1969 **Strassen [4]** published a simple 3 page paper that, for the first time, showed algorithms for matrix multiplication and inversion that were sub-$O(n^3)$. Though not fully recognized until later, this was a huge milestone for computing. While the matrix multiplication algorithm is an achievement with arguably greater impact we will focus on inversion. Strassen's matrix inversion operates by splitting the matrix into quadrants and running recursively on two of them. The algorithm is a prime example of a divide and conquer approach and is thus well suited for a parallel implementation.

However, there are some problems that prevent Strassen matrix inversion from being the ideal algorithm. Due to the recursive inversion only taking place on two of the quadrants, certain matrices can end up with recursive quadrants that are very close to singular, leading to high numerical error. A solution is to pivot the matrix at each level to whichever of the top-most quadrants has a higher determinant [8], but this adds extra complexity to the algorithm. Another issue is that each level of recursion contains many intermittent matrix multiplications. All of these calculations must be stored, which more then doubles the space requirements of a typical inversion.

2.2 Parallel Gauss-Jordan

While Gauss-Jordan is no longer the most efficient matrix inversion method, **Sharma et al [9]** were able to redesign it into a highly parallel GPU based Gauss-Jordan algorithm. Their implementation showed impressive results; achieving $O(n)$ latency, though with some major caveats.

Sharma et al hits many of the limitations of GPU hardware. In order to avoid the extra I/O penalty associated with global GPU memory, the algorithm must be able to store one row and one column of the matrix in shared memory. Additionally, the GPU algorithm improves on the parallel $O(n^3)$ by doing n^2 of the work in parallel, thus if the GPU does not have at least n^2 parallel threads, than the latency will increase exponentially.

2.3 In-Place Approach

Typically, the Gauss-Jordan algorithm requires appending a $n \times n$ unit matrix to the original matrix. However, in 2013 **DasGupta [10]** introduced a modified Gauss-Jordan algorithm that handles the inversion in-place. While this algorithm improves the space efficiency of Gauss-Jordan, it retains the time complexity of $O(n^3)$. In 2023, **Xuebin et al [11]** created a parallel modification of DasGupta's algorithm that is optimized for inverting many small matrices at a time on GPU. This algorithm is bound by similar limitations to the one in Sharma et al [9] (see 2.2). Assuming $n \times \textit{number of matrices}$ is small enough to fit into the total number of parallel threads, then the algorithm runs in $O(n^2)$ time.

2.4 GPU Thread-Data Remapping

Due to the nature of GPU architecture, threads within the same warp are not able to execute different paths in parallel. This limits the performance of workloads that contain conditional branching or uneven allocation of work as branches are serialized. Largely divergent workloads can try to avoid this overhead by periodically reshuffling data to reduce the divergence inside warps; this technique is called Thread-Data Remapping (TDR). The most common form of TDR involves stopping all work at set intervals and performing synchronization. This approach is less than ideal since full workload synchronization requires the CPU to step in and handle workload discrepancies between runs. Communication between the CPU and GPU is expensive and should be avoided if possible.

A better approach, introduced by **Cuneo and Bailey [12]**, handles TDR entirely on-GPU by implementing a work scheduling mechanism that is reminiscent of the promise and future concurrency model. While not the first on-GPU TDR, Cuneo and Bailey's method is the first to allow remapping across blocks without synchronization.

3 Justification

While there are many attempts at GPU matrix inversion algorithms, none thus far have used on-GPU TDR to handle the optimization of work. Cases of large recursion (see 2.1) or work sizes (see 2.2) exceeding the number of available threads can benefit immensely from the ability to remap and defer work on demand. Therefore, it is of interest to develop an algorithm that takes advantage of the highly parallel asynchronous compute offered by Cuneo and Bailey [12].

4 Evaluation

As the goal of this research is a practical implementation, the results will be evaluated through experimental comparison to existing matrix inversion implementations and by asymptotic complexity analysis. All necessary software is available for free. The development timeline of implementation can be found in section 5. Access to a capable GPU will be necessary for the purposes of benchmarking.

5 Research Plan

The following is a list of major project milestones and completion dates. Dates are only intended to be rough estimates and are thus subject to change.

#	Name	Due
5.1	Preliminary Setup	02/28
5.2	Winter Progress Report	03/15
5.3	Naive Gauss-Jordan	04/10
5.4	Optimized Gauss-Jordan	04/24
5.5	Hybrid Approach	05/15
5.6	Final Report	06/07

5.1 Preliminary Setup

Prepare a project repository with all necessary dependencies.

5.2 Winter Progress Report

A report containing all progress of winter quarter.

5.3 Naive Gauss-Jordan

Implement parallel Gauss-Jordan based upon the algorithm developed in Sharma et al [9] (section 2.2) that adds basic support for on-GPU TDR though the use of asynchronous primitives provided by Harmonize [12] (section 2.4).

5.4 Optimized Gauss-Jordan

Attempt optimization of parallel Gauss-Jordan algorithm by removing as much synchronization as possible. If applicable, implement in-place optimizations provided by Xuebin et al [11] (section 2.3).

5.5 Hybrid Approach

Implement a divergent path approach that chooses the optimal algorithm for any given matrix.

5.6 Final Report

A final report containing the following sections:

- Abstract. Project's goals, methodologies, and contributions.

- Introduction. Context of project, project goals and contributions.
- Related work. Existing work relevant to project and what distinguishes project from existing work.
- Description of Research. Description of models and assumptions (if any). Details of project goals and design. How the design of the project fulfills the goals.
- Evaluation / Results. Evaluation methodology and present results and analyses.
- Conclusions. Lessons learned and goals achieved by project.
- Future Work. Possible future work and open questions.
- References.

References

[1] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, "Solution of linear algebraic equations," in *Numerical Recipes: The Art of Scientific Computing*, 3rd ed. 32 Avenue of the Americas, New York, NY 10013-2473, USA: Cambridge University Press, 2007, ch. 2, pp. 37–109, ISBN: 978-0-521-88068-8 (cit. on p. 1).

[2] H. Anton and C. Rosses, "Applications of linear algebra," in *Elementary Linear Algebra*, 11th ed. 111 River Street, Hoboken, NJ 07030-5774, USA: John Wiley & Sons, Inc., 2014, ch. 10, pp. 527–713, ISBN: 978-1-118-43441-3 (cit. on p. 1).

[3] S. C. Althoen and R. McLaughlin, "Gauss-jordan reduction: A brief history," *American Mathematical Monthly*, vol. 94, no. 2, pp. 130–142, Feb. 1987, ISSN: 0002-9890. DOI: `10.2307/2322413`. [Online]. Available: `https://www.jstor.org/stable/2322413` (cit. on p. 1).

[4] V. Strassen, "Gaussian elimination is not optimal," *Numerische Mathematik*, vol. 13, pp. 354–356, Aug. 1969. DOI: `10.1007/BF02165411` (cit. on p. 1).

[5] D. Coppersmith and S. Winograd, "On the asymptotic complexity of matrix multiplication," in *22nd Annual Symposium on Foundations of Computer Science (sfcs 1981)*, 1981, pp. 82–90. DOI: `10.1109/SFCS.1981.27` (cit. on p. 1).

[6] H. Cohn, R. Kleinberg, B. Szegedy, and C. Umans, "Group-theoretic algorithms for matrix multiplication," in *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, 2005, pp. 379–388. DOI: `10.1109/SFCS.2005.39` (cit. on p. 1).

[7] H. Nguyen, *Gpu gems 3*, First. Addison-Wesley Professional, 2007, ISBN: 9780321545428 (cit. on p. 1).

[8] D. Bailey and H. Ferguson, "A strassen-newton algorithm for high-speed parallelizable matrix inversion," in *Supercomputing '88:Proceedings of the 1988 ACM/IEEE Conference on Supercomputing, Vol. I*, 1988, pp. 419–424. DOI: `10.1109/SUPERC.1988.44680` (cit. on p. 2).

[9] G. Sharma, A. Agarwala, and B. Bhattacharya, "A fast parallel gauss jordan algorithm for matrix inversion using cuda," *Computers & Structures*, vol. 128, pp. 31–37, 2013, ISSN: 0045-7949. DOI: `10.1016/j.compstruc.2013.06.015`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0045794913002095` (cit. on pp. 2, 3).

[10] D. DasGupta, "In-place matrix inversion by modified gauss-jordan algorithm," *Applied Mathematics*, vol. 4, no. 10, pp. 1392–1396, Sep. 2013. DOI: `10.4236/am.2013.410188` (cit. on p. 2).

[11] J. Xuebin, C. Yewang, F. Wentao, Z. Yong, and D. Jixiang, "Fast algorithm for parallel solving inversion of large scale small matrices based on gpu," *The Journal of Supercomputing*, vol. 79, no. 16, pp. 18 313–18 339, May 2023, ISSN: 1573-0484. DOI: `10.1007/s11227-023-05336-7`. [Online]. Available: `https://link.springer.com/article/10.1007/s11227-023-05336-7` (cit. on pp. 2, 3).

[12] B. Cuneo and M. Bailey, "Divergence reduction in monte carlo neutron transport with on-gpu asynchronous scheduling," *ACM Trans. Model. Comput. Simul.*, vol. 34, no. 1, Jan. 2024, ISSN:

1049-3301. DOI: 10.1145/3626957. [Online]. Available: https://dl.acm.org/doi/10.1145/3626957 (cit. on pp. 2, 3).

[13] J. R. Gardner, G. Pleiss, D. Bindel, K. Q. Weinberger, and A. G. Wilson, "Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration," in *Advances in Neural Information Processing Systems*, 2018. DOI: 10.48550/arXiv.1809.11165. [Online]. Available: https://arxiv.org/abs/1809.11165.

[14] I. Dimov, T. Dimov, and T. Gurov, "A new iterative monte carlo approach for inverse matrix problem," *Journal of Computational and Applied Mathematics*, vol. 92, no. 1, pp. 15–35, 1998, ISSN: 0377-0427. DOI: 10.1016/S0377-0427(98)00043-0. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377042798000430.

[15] C. Misra, S. Haldar, S. Bhattacharya, and S. K. Ghosh, "Spin: A fast and scalable matrix inversion method in apache spark," in *Proceedings of the 19th International Conference on Distributed Computing and Networking*, ser. ICDCN '18, Varanasi, India: Association for Computing Machinery, 2018, ISBN: 9781450363723. DOI: 10.1145/3154273.3154300.

[16] E. Agullo, H. Bouwmeester, J. Dongarra, J. Kurzak, J. Langou, and L. Rosenberg, "Towards an efficient tile matrix inversion of symmetric positive definite matrices on multicore architectures," in *High Performance Computing for Computational Science – VECPAR 2010*, J. M. L. M. Palma, M. Daydé, O. Marques, and J. C. Lopes, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 129–138, ISBN: 978-3-642-19328-6. DOI: 10.1007/978-3-642-19328-6_14.