




A Sparse Matrix parallel solver for OpenCL

Vishakh Begari ¹

¹ Independent Researcher, Germany

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Matthew Feickert](#) 

Reviewers:

- [@yurlungur](#)
- [@Brookluo](#)

Submitted: 05 June 2025

Published: unpublished

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

Summary

PIRO is an open-source C++/OpenCL software for performing high-performance numerical simulations. It provides modular solvers, supports various sparse formats (CSR, COO), and offers kernel-level customization for advanced users. PIRO is designed for research and production workflows requiring scalable parallel matrix computations.

Statement of Need

Compressed Sparse Row (CSR) is a space efficient way for storing sparse-matrices. It is particularly useful in situations where the matrix has many zero elements. Parallellising CSR matrix operations requires special treatment as not all elements in the matrix are readily available for independent processing. One solution is to process each row separately, making it straightforward to assign different rows to different threads. This approach requires the availability of atomic operations to extract good performance ([eguly & Giles, 2012](#)). For dynamic graphs stored in CSR, parallel algorithms can insert or delete edges concurrently. Lock-free or fine-grained locking mechanisms are employed to allow multiple threads to update the structure without significant contention([Eleni Alevra, 2020](#)). However, this introduces complexity in managing numerous locks, increasing the risk of deadlock or priority inversion if not designed meticulously ([Wheatman et al., 2024](#)). The landscape of high-performance linear algebra and sparse solvers is diverse. Existing libraries often lack GPU support or are tied to proprietary platforms (e.g., CUDA ([NVIDIA, 2022](#))). Libraries such as PETSc ([UChicago Argonne, 2025](#)) supports OpenCL through the ViennaCL([Wien, 2016](#)) backend, however, since ViennaCL primarily targets older OpenCL versions, this approach may lack full compatibility with modern OpenCL features. HYPRE([Laboratory, 2007](#)) is a library of high performance preconditioners and solvers designed for structured and semi-structured grids but GPU acceleration and dynamic matrix updates are limited. Eigen ([EIGEN, 2025](#)) is a highly mature C++ template library for linear algebra which is CPU-focused. Magma ([Innovative Computing Laboratory, 2025a](#)) is highly optimized for dense linear algebra on CUDA, MAGMA-sparse ([Innovative Computing Laboratory, 2025b](#)) is now in Legacy Support Mode (as of December 2024), with no active development.

While hash-based sparse matrix assembly has been established on CPUs ([Aspnäs et al., 2007](#)), adapting it for GPU execution is still a recent challenge ([Du et al., 2022](#)). PIRO addresses this gap with HTLF (Hash table with Load Factor), a complementary approach that scales the hash table using a fitting function. It uses a hash function to compute an index (hash key) into an array of slots, where the corresponding value is stored. Insertion, deletion and lookup can be performed in amortized constant time (on average) independent of the number of non zero elements in the table, assuming a good hash function and well-sized table. It is well known that HT are generally more efficient than search trees or other lookup structures for these operations, especially when fast access is required.

Additionally the software offers:

- **Cross-platform CPU / GPU operation** for dense and CSR based matrix matrices via OpenCL.
- **Modular equation solver** for Partial Differential Equations.
- **Post processing** export function for viewing results (e.g Paraview).
- **Benchmarking** tools and diagnostics.

Algorithm - HTLF

When solving a linear system numerically, especially for large systems, directly computing the inverse is not recommended for numerical stability and efficiency. Iterative solvers are algorithms that find approximate solutions to linear systems, such as $Ax = b$, by starting with an initial guess and progressively improving it through repeated iterations. If approximate solutions are not desired, Direct methods, such as Gaussian elimination, LU decomposition, and Cramer's rule, compute the exact solution to a linear system in a finite number of steps, assuming no rounding errors occur during computation. LU decomposition requires modification of the original matrix, which can be sparse by nature, into an Upper Triangular Matrix (UTM), which can also be sparse, before solving. This step adds, modifies and removes elements from the original matrix through repeated gaussian elimination steps (κ). If sparseness of the matrix is measured by the number of non zero elements (e), then

$$\mu = e/N^2$$

$$\bar{n} = \kappa/N^2$$

$$\lambda = e_{\kappa_0}/e_{\kappa_{N-1}}$$

where, $N = n_x * n_y * n_z$ if n_x, n_y, n_z are the number of grid cells in X, Y and Z direction respectively, the load factor λ is the ratio of nonzero elements in the first row ($\kappa = 0$) to those in the consecutive rows ($\kappa = N - 1$). e for an UTM obtained after gaussian elimination of a Laplacian operator for various cell grid sizes (N) is shown in the figure below.

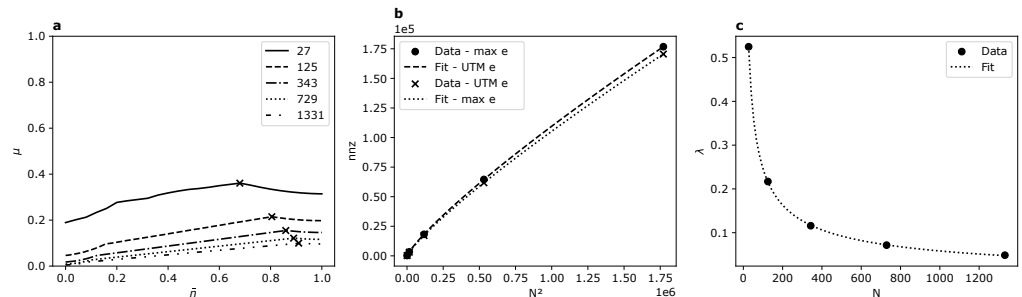


Figure 1: (a) Row filling for different N during gaussian elimination. 'x' marker denoting maximum e . (b) Curve fitting for e . (c) Curve fitting for the load factor.

The sparseness decreases initially, reaching a maximum, before finally increasing according to Figure 1.

Table size estimation

Choosing an appropriate table size for hashing is crucial for the performance of the algorithm. Since the trends are driven by (N), a curve fit for λ is performed as follows :

$$\lambda = a * N^b + c$$

The coefficients are $a = 3.3608$, $b = 0.5552$, $c = -0.014$.

71 Then,

$$TABLE_SIZE = e/(\lambda * \sigma)$$

72 where, σ is the scaling factor, providing more flexibility.

73 Hash function

74 A simple hash function is used :

$$key = index \% TABLE_SIZE$$

75 where,

$$index = row\ number * N + column\ number$$

76 Method

- 77 1. The sparse matrix is initially traversed to identify indices where extra elements are to be
- 78 added. They are assigned a key of -1 in the HT. The HT is then copied to the device.
- 79 2. The gaussian elimination step is performed on every element in parallel by the kernel.
- 80 Resulting zero elements are concurrently set to a key of -2. The HT is copied back to
- 81 the host.
- 82 3. Steps 1. and 2. are repeated $N - 1$ times.

83 For a large enough $TABLE_SIZE$, steps 1. and 2. are $O(1)$ at best. Sometimes traversal

84 performed in step 1 might need extra probing (upon encountering -2 before -1) during insertion

85 due to open addressing.

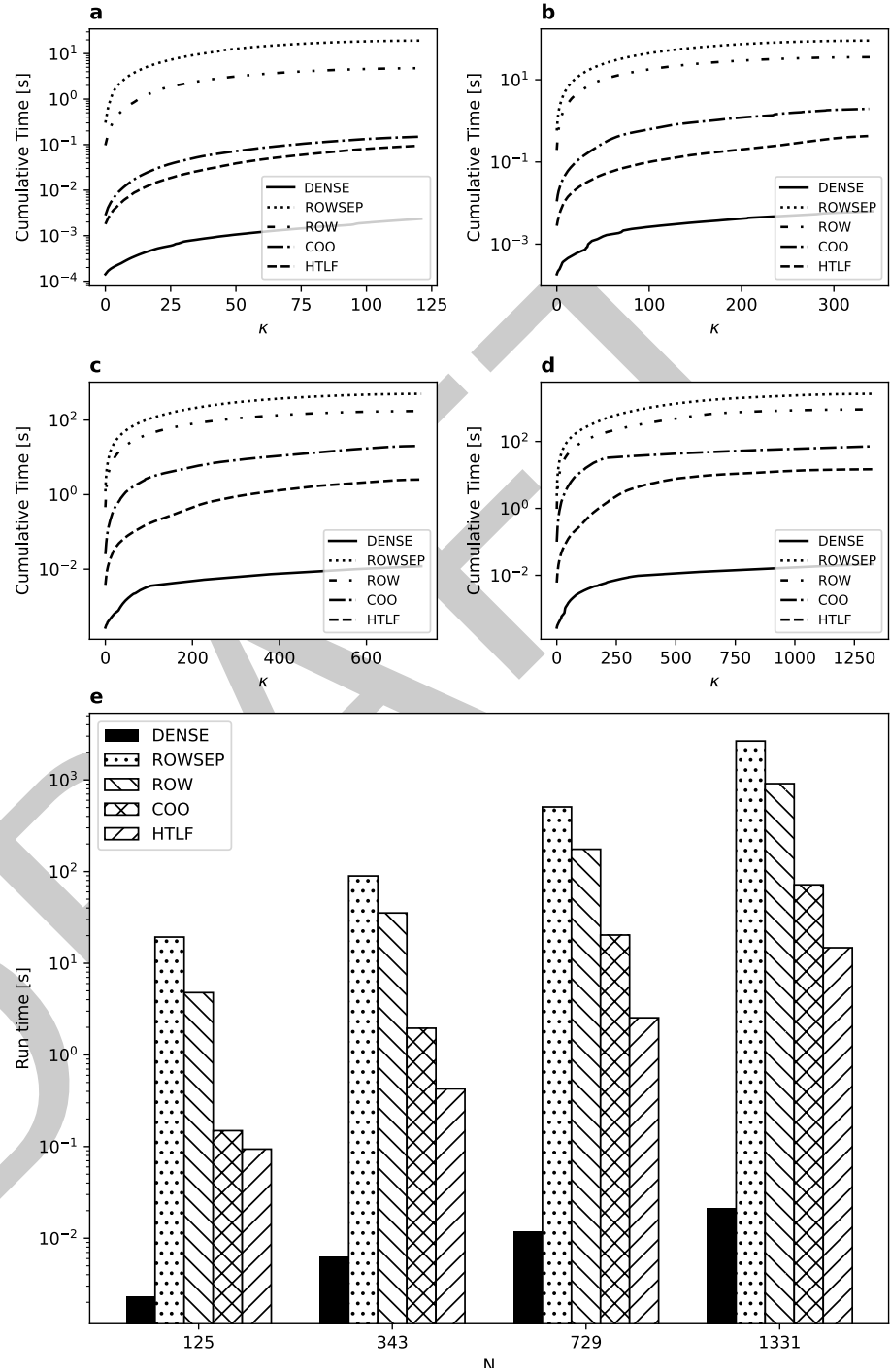


Figure 2: Cumulative run times across gaussian elimination steps for different N . (a) $N = 125$. (b) $N = 343$. (c) $N = 729$. (d) $N = 1331$. (e) Total run times. HTLF uses $\{\sigma = 0.7\}$.

Table 1: Run times and space complexities of various algorithms generating a UTM for a $7 \times 7 \times 7$ grid Laplacian on an AMD Radeon Pro 5300M; $factor =$ percentage of N^2 .

Study	Time [s]	Host memory [int, float]	Device Memory [int, float]
DENSE	0.0062	$N + e + 1, 1, $ $N^2 + e 3 * N^2 $	
ROWSEP	89.6593	$N + e + 1, N + factor + 1, $ $2 * N + e 3 * N + factor + 1 $	
ROW	35.3817	$N + 2 * n + e + 1, 3 * N + factor + 1, $ $N + 2 * n + e 3 * N + factor $	
COO	1.9186	$N + 2 * e + 1, e + N + 1, $ $2 * e e $	
HT	10.7566	TABLE SIZE + 3, TABLE SIZE	TABLE SIZE + 3, TABLE SIZE
HTLF ($\sigma=0.7$)	0.4587	TABLE SIZE + 3, TABLE SIZE	TABLE SIZE + 3, TABLE SIZE

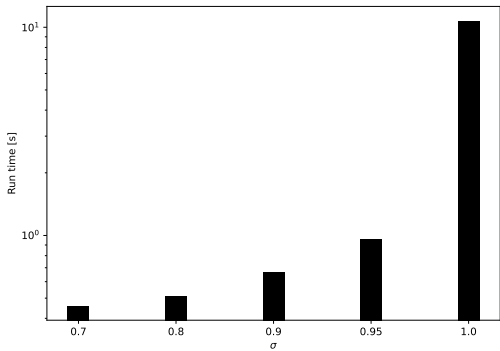


Figure 3: Run times for different σ .

Figure 3 shows performance improvements with reducing σ . The ideal value for σ can be chosen based on use case.

Equations

Partial differential equations (PDEs) can be solved in PIRO by constructing the corresponding matrix equations. The matrix can be represented in Dense, CSR, or HTLF formats. The choice of discretization depends on grid type. The current version implements finite difference methods for structured grids with First-Order Upwind, Downwind, Central-Difference spatial schemes and Forward Euler, Backward Euler temporal schemes. Support for finite volume methods on unstructured grids is currently under development.

Conclusion

Hash table representaton of sparse matrices for executing operations using parallel processing gives flexibility to prioritize balance between time and space efficiency. This is benenifical as the method can be adapted easily based on use case and hardware.

Scope for future work

A simple hash function was used to calculate key values. Alternative hash methods and its effect on various differential operators needs to be explored.

References

- Aspnäs, M., Signell, A., & Westerholm, J. (2007). *Efficient assembly of sparse matrices using hashing*.
- Du, Z., Guan, Y., Guan, T., Niu, D., Huang, L., Zheng, H., & Xie, Y. (2022). *OpSparse: A highly optimized framework for sparse general matrix multiplication on GPUs*. <https://arxiv.org/abs/2206.07244>
- eguly, I. R., & Giles, M. (2012). Efficient sparse matrix-vector multiplication on cache-based GPUs. *IEEE Xplore*, 1–12. <https://doi.org/10.1109/InPar.2012.6339602>
- EIGEN. (2025). <https://github.com/PX4/eigen>
- Eleni Alevra, D. M., Christian Menges. (2020). A parallel packed CSR data structure for large-scale dynamic graphs. In *GitHub repository*. GitHub. <https://github.com/domargan/parallel-packed-csr>
- Innovative computing laboratory. (2025a). <https://icl.utk.edu/magma/>
- Innovative computing laboratory. (2025b). https://icl.utk.edu/projectsfiles/magma/doxygen/_m_a_g_m_a-sparse.html
- Laboratory, L. L. N. (2007). *HYPRE: Scalable linear solvers and multigrid methods*. Lawrence Livermore National Laboratory. <https://computing.llnl.gov/projects/hyre-scalable-linear-solvers-multigrid-methods>
- NVIDIA. (2022). *cuSPARSE library*. NVIDIA Corporation. https://docs.nvidia.com/cuda/pdf/CUSPARSE_Library.pdf
- UChicago Argonne, L. (2025). *PETSc: Portable, extensible toolkit for scientific computation*. UChicago Argonne, LLC. <https://petsc.org/release/>
- Wheatman, B., Burns, R., & Xu, H. (2024). Batch-parallel compressed sparse row: A locality-optimized dynamic-graph representation. *IEEE Xplore*, 1–8. <https://doi.org/10.1109/HPEC62836.2024.10938461>
- Wien, T. (2016). *ViennaCL*. <https://viennacl.sourceforge.net>