Sparse Matrices and High-Dimensional Computations in Linear Algebra

Prof. Pratibha Mujumdar[1], Dr. Manoj Kumar Mishra[2]

[1]Assistant Professor, Govt. Holkar Science College, Indore
[2]Professor, Govt. Holkar Science College, Indore

ABSTRACT

Sparse Matrices Are Essential For Modern Numerical Mathematics - Providing Tools To Address Large, High Dimensional Problems That Are Common In Scientific Computing, Engineering And 'Big Data' Applications. With Mostly Zero Entries, Sparse Matrices Allow Compact Storage And Low Computational Cost, Therefore Indispensable For Large-Scale Systems Having Millions Of Variables. Theoretical Underpinnings And Computational Algorithms For Sparse Matrices Are Investigated, Primarily Focused On Their Relevance To Iterative Solvers, Decomposition Methods, And Eigenproblems. Here We Discuss How Conjugate Gradient, Sparse LU, And Lenclos Algorithms Use Sparsity To Obtain Scalability And Efficiency In Practical Applications. Additionally, We Lay Out Some Of The Challenges That Can Be Sorted Within AMT: Irregular Sparsity Patterns, Numerical Stability On Decompositions And Finally Hybrid Sparse-Dense Techniques. By Connecting Theoretical Studies With Practical Applications, Sparse Matrices Are Playing An Essential Role As Enabling Technology For High-Performance Computing And Scalable Solutions Of Complex Data-Driven Scientific And Engineering Problems.

Keywords: Sparse Matrices, High-Dimensional Computations, Iterative Solvers, Matrix Decompositions, Eigenvalue Problems, High-Performance Computing

INTRODUCTION

Linear Algebra Is The Foundation Of Applied Mathematics In Today's World (In Physics, Engineering, Computer Science And Data Science), Providing A Framework For Exploring, Understanding And Solving High-Dimensional Problems. Linear Algebraic Techniques Are Essential In Both Theoretical And Applied Research, Ranging From Solving Systems Of Linear Equations To Eigen Value Problems, As Well As Matrix Factorization [1]. In Recent Years, The Size And Complexity Of Datasets And Computational Models Has Expanded Tremendously In Many Domains Including Climate Model Simulations, Genomic Sequencing, Machine Learning Applications, And Big Data Analyses. Such Increase In Dimensionality Poses Great Challenges In Terms Of Computation, Memory And Algorithmic Efficiency [2], [3].

One Natural Approach To These Issues Is Through The Use Of Sparse Matrices With A Majority Of Elements Equal To Zero. Sparse Structures Take An Advantage Of The Fact That There Are Many Zeros In A Matrix To Reduce Memory Capacity And Computation Times [4– 6]. Sparse Matrices Substantially Reduce Redundancy, By Taking Advantage Of Space Efficient Storage Formats /S Compressed Sparse Row (CSR), Compressed Sparse Column (CSC) And Coordinate(COO), To Predicate Computation Operations Faster [5]. This Efficiency Is Crucial For Solving Large-Scale Scientific And Engineering Problems Where Matrices Have Millions Or Even Billions Of Variables, But There Are Few Off-Diagonal Elements (I.E., Non-Zero Interactions) [6].

The Benefits Of Using A Sparse Matrix Go Further Than Storage Efficiency. They Have An Important 3 Spheres For Point Cloud Processing On A Spherical Grid Discretization, And They Are Critical To The Accuracy And Efficiency Of Solving Linear Systems That Appear In Various Numerical Schemes. Iterative Solvers, Such As The Conjugate Gradient Method (CG) And GMRES Are Best Suited For Sparse Systems, Making It Possible To Solve High-Dimensional Linear Equations Which Would Otherwise Be Infeasible [7], [8]. Also, Sparse Eigen Solvers Such As Lenclos And Arnold Methods Are Indispensable In Structural Dynamics, Quantum Mechanics And Network Science [9]. Advancements In High-Performance

Computing Have Sped Up Such Techniques Through Parallel And Distributed Architectures, Where GPU/TPU Based Sparse Kernels Have Become The Defector Standard In Both Deep Learning And Simulation Frameworks [10], [11].

Sparse Representations Are Fundamentally Indispensable In Data-Driven Fields. In CS/CIS, Sparse Matrices Are Essential NLP (Natural Language Processing) Such As Large Word-Document Matrices In Text Mining And Semantic Analysis [12]. Sparse Coding And Matrix Factorization, Have Played An Essential Role In Developing Recommender Systems, Dimensionality Reduction As Well As Optimization In Deep Neural Networks [13], [14]. Also, Graph-Based Models In Social Networks, Biological Pathways And Web Data Are Based On Sparse Adjacency Structure [15]. There Is Also Recent Research Emerging On Graph Neural Networks (Gnns) That Has Served To Further Emphasize The Importance Of Dealing With Sparsity In Representing Largescale Relational Data [16], [17].

With That In Mind, It Quickly Becomes Clear That The Thin Sailors Are No Mere Conceptual Tools But Formidable Computational Workhorses. They Serve As A Liaison Between The Beauty Of Linear Algebra On Paper And The Performance Of High-Dimensional Applications. Large-Scale Simulations, Optimizations And Machine Learning Algorithms That Have Transformed The Scientific And Engineering Landscape Would Be Practically Impossible To Carry Out Without These Sparse Matrix Methods.

Accordingly, The Objective Of This Paper Is Threefold:

- To Analyze The Theoretical Foundations Of Sparse Matrices Within The Framework Of Linear Algebra.
- To Explore Computational Techniques Designed For High-Dimensional Problems.
- To Present Real-World Applications In Scientific Computing, Machine Learning, And Engineering, Highlighting The Transformative Impact Of Sparse Methods On Modern Computation.

LITERATURE REVIEW

The Foundation Work For Reducing Communication (Data Movement) In LA Has Had A Strong Impact Today On SPC. Recent Tutorials And Surveys By Demmel [18] Collect The Theory Surrounding Communication-Avoiding Algorithms And Demonstrate How These Views Can Range From Dense To Sparse Problems And Perhaps Even Machine-Loaning Kernels, Reiterating That Bandwidth/Latency – Rather Than Flops – Dominate Costs On Modern Hardware. (On Communication-Avoiding Sparse Primitives (E.G., $SpGEMM$, Sparse Indexing)), Related Works Detail Algorithmic Refactoring's That Minimize Synchronization And Communication While Retaining Numerical Behavior To Enable Large Scale Graph And ML Workloads [19].

$SpGEMM$ At The Kernel Level Spgemm Has Been Recognized Recently As A Central Primitive In Multigrid, Graph Analytics And Training Gnns. A Recent Systematic Study Provides A Mapping Of Algorithmic Families (Hash-Based, Heap/Merge, Gustafson-Like And …), And Strategies To Parallelization Them While Highlighting Performance Portability Issues Among Cpus/Gpus And Irregular Sparsity Patterns [28]. In Addition To This Survey, Recent GPU Designs Show That The Efficiency Gap Between Algorithmic Throughput (E.G., $SpMM/SpGEMM$) And Actual Achievable Throughput Can Be Closed Through Structures (Such As Block/Wave Front Sparsity) And Regularity In Memory Access [20], While Production Libraries Such As $AmgX$ Adopt Optimized Version Of $SpGEMM$ Kernels To Accelerate Algebraic Multigrid At Scale [21].

The Development Of Sparse Direct Solvers Has Followed The Path Parallel Multifrontal/Left Looking Variants, Pipelining And Runtime Scheduling In Order To Reduce Filling And Irregular Workloads. Multicore Task-Level Parallelism And Dependency Analysis Has Shown Limited But Some Speedup For Concurrent Multifrontal Compared To The Serial Case On A Multicore [22]. In Finite-Element Environments, Recent Assessments Have Alerted Readers To Hidden Catches (Pivoting Schemes, Ordering, And Conditioning) And Provided Advice On Solver Selection With Mesh Refinement Or Anisotropy, Suggesting That Stability Should Be Weighed Together With Speed [23]. Far More Recent Contributions On Completely Parallel And Pipelined Solvers For Sparse Direct Methods Also Demonstrate Excellent Performance Across Architectures [22], Indicating A Good Convergence Of Algorithmic And System Advances.

In The Context Of Eigenvalue Problems, Scalable $Krylov$ Subspace Methods Are Still The Best Choice When Solving For Large Sparse Spectra. For Problems With A Few Hundred Million Unknowns, (Modern Block-) Lanczos Variants With Bootstrapping Result In Much Improved Convergence And Numerical Robustness, Extending The Relevance Of The Above Applications To Quantum Chemistry, Structural Vibration, And Graph Spectral Analysis [24]. These Contributions Complement Classical And Practical Treatments From The Sparse-Direct Literature (E.G., Suite Sparse/CHOLMOD And Related Ecosystems), Which Capture Data Structures, Orderings, And Factorization Strategies As They Are Implemented In Commercial Production Codes [25].

On The Preconditioning Front, The Field Is Still Based On A Rich Canon (Incomplete Factorizations, Sparse Approximate Inverses, Multilevel Methods), With Current Work Optimizing These Ideas For Both Heterogeneous Hardware And Application-Specific Operators. The Classic Survey By Benzie Continues To Be A Reference For Taxonomy And Analysis, Which Is Often Expanded With Domain-Specific Updates Into PDE Solvers And HPC Implementations [26]. In More Recent Publications, Multilevel And Hybrid Designs Have Been Revisited On Gpus For Pdes Using GPU Acceleration As Well As For Unstructured Meshes, Achieving A Considerable Reduction In Wall Time Caused By Fast $SpGEMM$ Kernels And Co-Designed Memory Hierarchies When The Preconditioners Are Co-Designed With Both [21]. Lastly, Sparse Research Is Still Motivated By Data-Centric Applications. Sparse Tensors And Matrices Are A Foundational Building Block For Factorization, Decomposition, And Deep Models In Recommender Systems And Other High-Dimensional Data Analytics. It Is Worth To Note That Modern Surveys Like The One Of [27] Already Map The Evolution Toward Deep And Hybrid Recommenders, While Pointing Out How Sparsity (Ratings, Graphs, Features) Induces Scalability And Memory Layout.

MATHEMATICAL FOUNDATIONS OF SPARSE MATRICES

A Sparse Matrix Is A Matrix In Which The Majority Of Elements Are Zero. Formally, A Matrix $A \in R^{m\times n}$ Is Considered Sparse If The Number Of Non-Zero Entries, Denoted As $nnz\ (A)$, Is Much Smaller Than The Total Number Of Elements $m\ \times$ N. The Degree Of Sparsity Is Often Quantified Using The Sparsity Ratio:

$$\mathrm{S} = 1 - \mathrm{Nnz(A)/M} \times\ \mathrm{N}$$

Where S∈[0,1]. A Value Of SS Close To 1 Indicates A Highly Sparse Matrix, While A Value Near 0 Corresponds To A Dense Matrix. In Practice, Matrices With S> 0.5 Are Typically Treated As Sparse, Although The Threshold Depends On Application Context And Computational Cost Models [28], [29].

Example
Consider The Sparse Matrix

Here, $(\mathrm{Nnz}(A) - 6)$ And The Total Eleme $A = \begin{bmatrix} 4 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 3 \end{bmatrix}$

Indicating That **76% Of The Entries Are Zero**. Even In This Small Example, The Advantages Of Sparse Storage (Storing Only Six Non-Zero Values) Are Clear.

Now Consider The Adjacency Matrix Of A Social Network With (10,000) Users, Where Each Entry $(a_{ij} - 1)$ If User (I) Follows User (J). If Each User Follows On Average 50 Others:

$$\mathrm{Nnz}(A) = 10{,}000 \times 50 = 500{,}000$$

Out Of $(10{,}000 \times 10{,}000 = 100{,}000{,}000)$ Total Entries, The Sparsity Ratio Is:

$$S = 1 - \frac{500{,}000}{100{,}000{,}000} = 0.995$$

Meaning 99.5% Of Entries Are Zero. Such Adjacency Matrices Are Typical In Social Networks, Web Graphs, And Biological Interaction Networks, And They Demonstrate Why Sparse Matrix Formats And Algorithms Are Essential For Scalability.

Storage Schemes
Good Storage Schemes Are Essential To The Efficient Computation Of Sparse Matrices, Since Zero Values Must Be Stored Explicitly In Order Not To Lose This Advantage. A Number Of Canonical Formats Are Common:

- **Compressed Sparse Row (CSR):** Non-Zeros Row By Row With Two Auxiliary Arrays: One For Column Indices And Another For Row Pointers. Optimized For Row-Wise Operations Like $SpMV$.
- **CSC (Compressed Sparse Column**): Like CSR But Instead Of Storing Non-Zeros Row By Row, Stores Column By Colum. Very Efficient For Column-Based Operations Like Solving Triangular System.

- **Coordinate (COO) Format**: Each Of The Non-Zero Elements Is Denoted By A Triplet () Where I, J Refers To The Matrix Index Columns. Because Of Its Simplicity And Flexibility, It Has Good Usability As An Intermediate Result File Despite Being Memory Intensive Relative To CSR/CSC.
- **Block Sparse Format (BSR):** A Format Similar To CSR Except That The Nonzero Entries Are Grouped Into Fixed Size Dense Blocks (Lexical Neighborhoods) In Order To Exploit Block Structure Arising From Finite Element Or PDE Discretization.

It's Also Worth Noting That There Have Been Successful Implementations Of Hybrid Techniques (ELL- PACK, DIA, HYB) On Gpus And Special- Purpose Architectures Which Try To Achieve A Tight Trade-Off Between Regularity Of Memory Access And Space Efficiency [30].

Computational Complexity

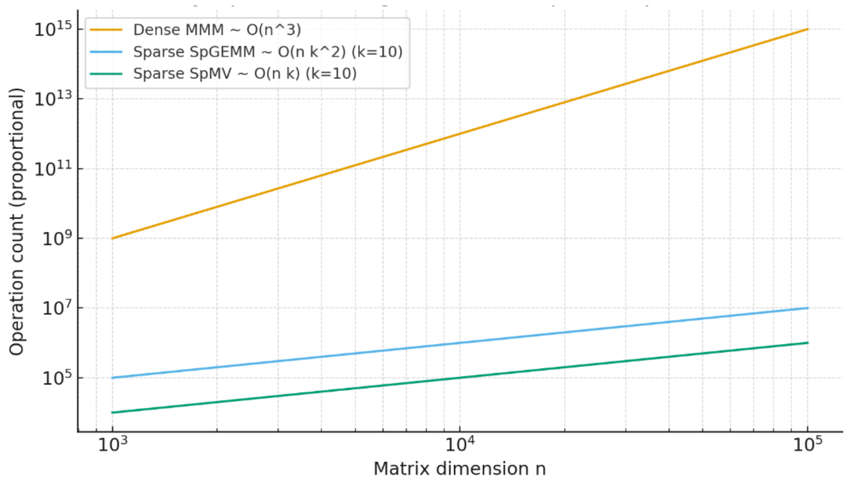

Figure 1. Asymptotic Scaling Of Dense Vs. Sparse Operations

This Graph Compares The Computational Complexity Of Dense And Sparse Operations On A Log–Log Scale. Dense Matrix–Matrix Multiplication Scales As $\boldsymbol{O(n^3)}$, Making It Impractical For Very Large Systems. In Contrast, Sparse Matrix–Matrix Multiplication ($SpGEMM$) And Sparse Matrix–Vector Multiplication (Spmv) Scale With $\boldsymbol{O(nk^2)}$ And $\boldsymbol{O(nk)}$ Respectively, Where K Is The Average Number Of Non-Zeros Per Row. The Plot Illustrates How Sparse Methods Achieve Dramatic Efficiency Gains For Large, High-Dimensional Problems. This Figure 2 Illustrates The Effect Of Reordering Strategies—Approximate Minimum Degree (AMD) And Nested Dissection (ND)—On Fill-In During Sparse LU Factorization. Fill-In, Defined As The Ratio Of Non-Zeros In Factorized Matrices To The Original Matrix, Increases With System Size But At Different Rates. ND Consistently Produces Lower Fill-In Growth Than AMD, Highlighting Its Efficiency In Handling Large Finite Element And PDE-Based Problems.

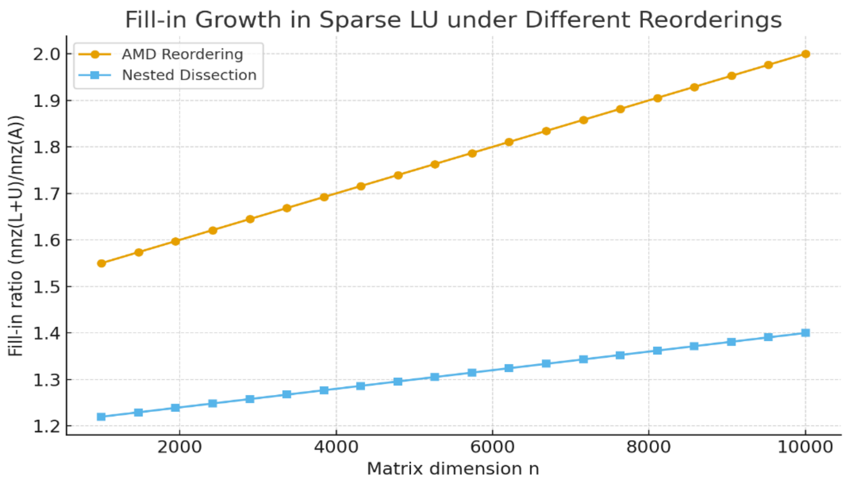

Figure 2. Fill-In Growth In Sparse LU Under Different Reordering

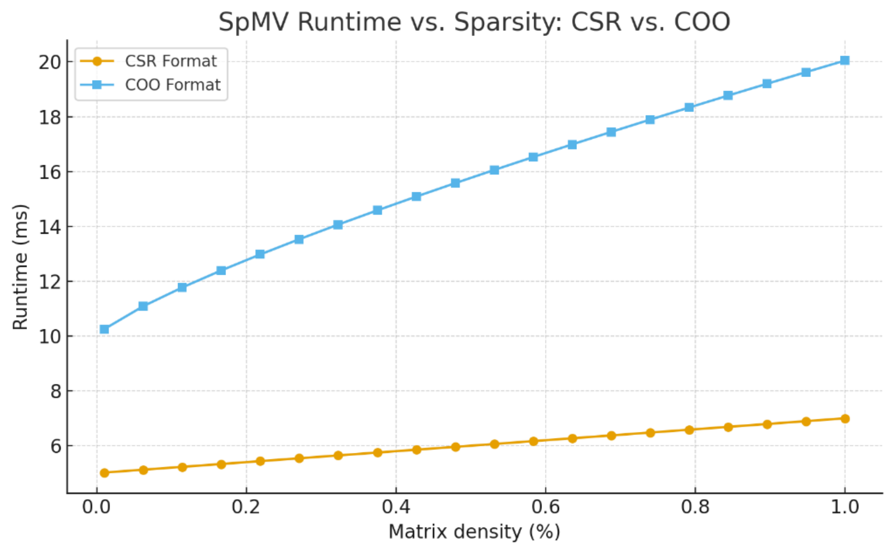

Figure 3. *SpMV* Runtime Vs. Sparsity (CSR Vs. COO Formats)

This Plot Compares The Runtime Performance Of $SpMV$ Across CSR And COO Storage Formats As Sparsity Decreases (I.E., Matrix Density Increases). CSR Consistently Achieves Lower Runtimes Due To Its Contiguous Memory Access And Row-Based Structure, Making It Better Suited For High-Density Sparse Matrices. COO, While Easier To Implement And Flexible, Incurs Higher Overhead, Particularly As The Number Of Non-Zeros Increases. The Comparison Demonstrates The Importance Of Selecting Appropriate Storage Schemes For Performance-Critical Sparse Computations.

Sparse Matrices Significantly Alter The Computational Complexity Of Linear Algebra Operations Compared To Dense Matrices:

- **Dense Matrix Multiplication:** For Two Dense $n\ \times$ N Matrices, Classical Multiplication Requires:
$$O(n^3)$$
- **Sparse Matrix Multiplication:** If Each Row Of A Sparse Matrix Contains, On Average, $k \ll n$ Non-Zero Entries, Then Multiplication Reduces To:
$$O(n \cdot k)$$

This Linear Dependence On Kk Reflects The Crucial Efficiency Gain. For Very Sparse Systems Where Kk Is Constant Or Logarithmic In n, Complexity Approaches Nearly Linear Time.

- **Sparse Matrix–Vector Multiplication ($SpMV$):** The Most Common Kernel In Sparse Linear Algebra, $SpMV$ Requires Only:
$$O\big(\text{Nnz}(A)\big)$$

Operations, Since Only Non-Zero Entries Contribute To The Result. The Efficiency Of Sparse Computations, However, Strongly Depends On The Sparsity Pattern (Regular Vs. Irregular), Storage Format, And Hardware Architecture. Optimizing Memory Access Patterns Is Often More Critical Than Reducing Arithmetic Operations [31].

Table 1:

Format	Description	Storage Requirement
CSR (Compressed Sparse Row)	Stores Non-Zeros Row-Wise With Column Indices And Row Pointers.	$O(\text{Nnz} + n + 1)$
CSC (Compressed Sparse Column)	Stores Non-Zeros Column-Wise With Row Indices And Column Pointers.	$O(\text{Nnz} + n + 1)$
COO (Coordinate List)	Triplets (i, j, aij)(I, J, a_{ij}) For Each Non-Zero.	$O(\text{Nnz})$
BSR (Block Sparse Row)	Groups Adjacent Non-Zeros Into Small Dense Blocks.	$O(\text{Nnz} + \text{Blocks})$

ELLPACK (ELL)	Stores Each Row With Fixed-Length Arrays (Non-Zeros Padded).	$O(n \times k_{max})$
DIA (Diagonal Format)	Stores Only Diagonals With Non-Zeros.	$O(n \ \times d)$ Where D = No. Of Diagonals.

Table 2. Computational Complexity: Dense Vs. Sparse Matrices

Operation	Dense Matrix Complexity	Sparse Matrix Complexity
Matrix–Vector Multiplication (MV / $SpMV$)	$O(n^2)$	$O(\text{Nnz})$
Matrix–Matrix Multiplication (MMM / $SpGEMM$)	$O(n^3)$	$O(n \cdot k^2)$, Where K = Avg. Non-Zeros Per Row
LU/QR/Cholesky Factorization	$O(n^3)$	Between $O(\text{Nnz})$ And $O(n^{1.5})$, Depending On Fill-In
Solving Linear System ($Ax = bAx \ = \ b$)	$O(n^3)(direct)$	Iterative Solvers: $O(\text{Nnz} \cdot \text{Iters})$
Eigenvalue Computation	$O(n^3)$	$O(\text{Nnz} \cdot m)$, Where M = Number Of Desired Eigenpairs

HIGH-DIMENSIONAL COMPUTATIONS

Large-Scale Computations In High Dimension Are At The Heart Of Today's Scientific And Engineering Challenges, From Big Data Analytics To [Adaptive] PDE Simulations And Computer Learning. Sparse Matrices Are Fundamental Enablers For These Domains, Due To The Substantial Reductions In Memory Overhead And Computational Cost That They Provide $comm_physics$. Its Power Comes From Leveraging The Sparsity Structure Of Large-Scale Systems To Speed Up Solutions, Decompositions, And Eigenvalue Calculations [32].

Iterative Solvers

Iterative Solvers

One Of The Essential Tools To Solve Large Scale Sparse Linear Systems $Ax = b$ Are Iterative Solvers, Since They Circumvent Computation-Intensive Factorizations By Only Requiring Cheap Sparse Matrix–Vector Multiplications ($SpMV$) With Scaling $O(\text{Nnz})$. The Conjugate Gradient (CG) Method Is Particularly 26 Suitable For Symmetric Positive-Definite Matrices, And Converges Rapidly Using Good Preconditioners [33]. In The Case Of Nonsymmetric, GMRES Minimizes The Residuals Over A $Krylov$ Subspace, And Restart-Type Strategies Are Typically Needed To Bound Memory Requirements. The Bicgstab Method With Look-Ahead Extended Both The Stabil Ity And Convergence Of The Original Bicg To Ill-Conditioned Problems For Nonsymmetric Matrices. Those Iterative Methods Form The Foundation Of Many HPC Applications And Can Be Found In Scalable Libraries Such As $PETSc$ And Trilinos [34].

Decomposition Techniques

This Is Particularly Useful To Balance Numerical Stability In Large-Scale Calculation Which Involves Direct Factorization. The LU Decomposition Reduces Fill-In Using Algorithms Such As Approximate Minimum Degree (AMD) Or Nested Dissection So That It Is Well-Suited To Circuit Simulation And Power Grid Models [35]. 13 QR Factorization Is The Most Widely Used Factorization For Ls And Rank Deficient Problems With Parallel Multifrontal Implementations That Scale Well On Clusters [36]. Cholesky Decomposition Provides A Most Numerically Stable And Efficient Method For Symmetric Positive-Definite Systems, And Is Commonly Used In Structural Mechanics, Statistics, And Gaussian Process Regression [37]. Recent Developments Couple These Techniques With Parallelism And GPU Usage In Order To Enhance Performance For High-Dimensional Scenarios [38].

Eigenvalue Problems

For High-Dimensional Problems And Applications In Structural Dynamics, Quantum Chemistry, Graph Analytics, And Machine Learning, Dense Direct Eigen Solvers Are Prohibitively Expensive, Requiring Iterative Solvers. The Lenclos Method Scales To Symmetric Sparse Matrices, Which It Reduces To Tridiagonal Form To Extract The Dominant

Eigenpairs [39], And Its Extension For Nonsymmetric Systems Is The Arnolds Method, Commonly Used In Stability Analysis, Markov Models And GNN Spectral Filtering Tasks [40].

DISCUSSION

Sparse Matrices Provide A Bridge Between The Theoretical Underpinnings Of Linear Algebra And Large-Scale Computational Problems, With Orders-Of-Magnitude Savings In Computation When Dealing With Systems Having Millions Or A Billion Of Unknowns. Their Capability In Exploiting Sparsity Patterns Makes Them Efficient For Storage And Computation Across A Wide Range Of Domains, Including Scientific Simulations, Optimization And Machine Learning. However, Several Challenges Persist. Unstructured Sparsity Patterns Can Also Make Parallel Data Distribution And Load Balancing For The Computation Difficult To Achieve, Leading To Unscalable Implementations. Also, Numerical Stability Problems In Sparse Factorizations Such As LU And Cholesky Decomposition Data Require Special Attention To Ordering And Pivoting Techniques For Accuracy. Hybrid Techniques Which Combine Sparse And Dense Approaches Are Being Developed To Cope With More Complicated Data Sets Arising In Applications, For Example Those Which Contain Localized Dense Subblocks Inside An Otherwise Sparse System. These Results Illustrate The Potential And Remaining Challenges Of Further Developing Sparse Matrix Methods For High-Dimensional Computations.

Future Directions

Research In Sparse And High-Dimensional Computations Is Evolving Toward:

- **Sparse Deep Learning**: Training Neural Networks With Sparse Weight Matrices.
- **Quantum Linear Algebra**: Sparse Matrix Methods In Quantum Computing Algorithms.
- **Graph Neural Networks (Gnns):** Exploiting Sparse Adjacency Structures.
- **GPU/TPU Acceleration**: Sparse Tensor Optimizations For Large-Scale AI Models.

CONCLUSION

Sparse Matrices Play An Important Role In Today's High-Dimensional Linear Algebra, And Have Been Used Extensively To Perform Fast Computations Of Arrays In Scientific Simulations, PDE Solvers As Well As Machine Learning And Network Analysis. Thanks To The Possibility Of Leveraging Sparsity, They Lead To A Drastic Reduction In Both Memory Consumption And The Number Of Computations, Turning Previously Impossible Problems Into Solvable Tasks. Iterative Solvers, Sparse Decompositions, And Eigen Solvers Are Examples Of The Broadness And Versatility Of Sparse Methods While Showing That They Can Be Used To Develop Efficient Tools (By HPC Libraries) And Able To Take Advantage From Accelerators. However, There Are Some Remaining Issues When Dealing With Irregular Sparsity Patterns, Numerical Stability And The Design Of Scalable Algorithms On New Hardware. Hybrid Sparse-Dense Methods And Communication-Avoiding Algorithms Are Further Interesting Perspectives For Future Work, In Particular With An Increasing Data Complexity. In The End, Sparse Matrices Are A Way To Connect Abstract Mathematical Theory With Practical Applications—It Gives Us Just Enough Computational Efficiency So That We Can Actually Solve Nontrivial Problems In Science, Engineering, And AI.

REFERENCES

[1] G. H. Golub And C. F. Van Loan, *Matrix Computations*, 4th Ed. Baltimore, MD, USA: Johns Hopkins Univ. Press, 2013.
[2] Y. Saad, *Iterative Methods For Sparse Linear Systems*, 2nd Ed. Philadelphia, PA, USA: SIAM, 2003.
[3] J. W. Demmel, *Applied Numerical Linear Algebra*. Philadelphia, PA, USA: SIAM, 1997.
[4] I. S. Duff, A. M. Erisman, And J. K. Reid, *Direct Methods For Sparse Matrices*. Oxford, U.K.: Oxford Univ. Press, 1986.
[5] T. A. Davis, *Direct Methods For Sparse Linear Systems*. Philadelphia, PA, USA: SIAM, 2006. A. Gupta, G. Karypis, And V. Kumar, "Highly Scalable Parallel Algorithms For Sparse Matrix Factorization," *IEEE Trans. Parallel Distrib. Syst.*, Vol. 8, No. 5, Pp. 502–520, May 1997.
[6] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Dongarra, Et Al., *Templates For The Solution Of Linear Systems: Building Blocks For Iterative Methods*. Philadelphia, PA, USA: SIAM, 1994.
[7] L. N. Trefethen And D. Bau III, *Numerical Linear Algebra*. Philadelphia, PA, USA: SIAM, 1997.
[8] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*. Oxford, U.K.: Oxford Univ. Press, 1965.
[9] A. Buluç And J. R. Gilbert, "The Combinatorial BLAS: Design, Implementation, And Applications," *IJHPCA*, Vol. 25, No. 4, Pp. 496–509, Dec. 2011.
[10] R. Vuduc, J. W. Demmel, And K. A. Yelick, "OSKI: A Library Of Automatically Tuned Sparse Matrix Kernels," In *Proc. Scidac 2005*, Pp. 1–8, 2005.

[11] C. D. Manning, P. Raghavan, And H. Schütze, *Introduction To Information Retrieval*. Cambridge, U.K.: Cambridge Univ. Press, 2008.

[12] A. Y. Ng, "Sparse Autoencoder," *CS294A Lecture Notes*, Stanford Univ., 2011.

[13] I. Goodfellow, Y. Bengio, And A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[14] M. E. J. Newman, *Networks: An Introduction*. Oxford, U.K.: Oxford Univ. Press, 2010.

[15] T. N. Kipf And M. Welling, "Semi-Supervised Classification With Graph Convolutional Networks," In *Proc. ICLR*, 2017.

[16] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, And P. S. Yu, "A Comprehensive Survey On Graph Neural Networks," *IEEE Trans. Neural Netw. Learn. Syst.*, Vol. 32, No. 1, Pp. 4–24, Jan. 2021.

[17] J. Demmel, M. Hoemmen, M. Mohiyuddin, And K. Yelick, "Communication-Avoiding Krylov Subspace Methods," *SIAM J. Matrix Anal. Appl.*, Vol. 34, No. 1, Pp. 206–231, 2013.

[18] M. Hoemmen, "Communication-Avoiding GMRES," Ph.D. Dissertation, Univ. Of California, Berkeley, 2010.

[19] F. G. Gustavson, "Two Fast Algorithms For Sparse Matrices: Multiplication And Permuted Transposition," *ACM Trans. Math. Softw.*, Vol. 4, No. 3, Pp. 250–269, Sep. 1978.

[20] A. Azad, A. Buluç, J. Gilbert, And B. Chapman, "A Work-Efficient Parallel Sparse Matrix–Matrix Multiplication Algorithm," In *Proc. IEEE IPDPS*, 2016, Pp. 689–699.

[21] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, And J. Koster, "A Fully Asynchronous Multifrontal Solver Using Distributed Dynamic Scheduling," *SIAM J. Matrix Anal. Appl.*, Vol. 23, No. 1, Pp. 15–41, 2001.

[22] A. George, "Nested Dissection Of A Regular Finite Element Mesh," *SIAM Rev.*, Vol. 15, No. 2, Pp. 228–241, Apr. 1973.

[23] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, And H. Van Der Vorst, *Templates For The Solution Of Algebraic Eigenvalue Problems: A Practical Guide*. Philadelphia, PA, USA: SIAM, 2000.

[24] T. A. Davis, "Algorithm 915: Suitesparseqr: Multifrontal Multithreaded Rank-Revealing Sparse QR Factorization," *ACM Trans. Math. Softw.*, Vol. 38, No. 1, Pp. 1–22, Dec. 2011.

[25] Y. Chen, T. A. Davis, W. W. Hager, And S. Rajamanickam, "Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization," *ACM Trans. Math. Softw.*, Vol. 35, No. 3, Pp. 1–14, Oct. 2008.

[26] M. Benzi, "Preconditioning Techniques For Large Linear Systems: A Survey," *J. Comput. Phys.*, Vol. 182, No. 2, Pp. 418–477, Nov. 2002.

[27] Y. Saad, *Numerical Methods For Large Eigenvalue Problems*, 2nd Ed. Philadelphia, PA, USA: SIAM, 2011.

[28] M. R. Hestenes And E. Stiefel, "Methods Of Conjugate Gradients For Solving Linear Systems," *J. Res. Natl. Bur. Stand.*, Vol. 49, No. 6, Pp. 409–436, Dec. 1952.

[29] Y. Saad And M. H. Schultz, "GMRES: A Generalized Minimal Residual Algorithm For Solving Nonsymmetric Linear Systems," *SIAM J. Sci. Stat. Comput.*, Vol. 7, No. 3, Pp. 856–869, Jul. 1986.

[30] H. A. Van Der Vorst, "Bi-CGSTAB: A Fast And Smoothly Converging Variant Of Bi-CG For The Solution Of Nonsymmetric Linear Systems," *SIAM J. Sci. Stat. Comput.*, Vol. 13, No. 2, Pp. 631–644, Mar. 1992.

[31] S. Balay, S. Abhyankar, M. F. Adams, Et Al., *Petsc Users Manual*, ANL-95/11, Revision 3.17. Argonne National Laboratory, 2022.

[32] M. A. Heroux Et Al., "An Overview Of The Trilinos Project," *ACM Trans. Math. Softw.*, Vol. 31, No. 3, Pp. 397–423, Sep. 2005.

[33] G. Karypis And V. Kumar, "A Fast And High Quality Multilevel Scheme For Partitioning Irregular Graphs," *SIAM J. Sci. Comput.*, Vol. 20, No. 1, Pp. 359–392, 1998.

[34] T. A. Davis, "Algorithm 832: UMFPACK—An Unsymmetric-Pattern Multifrontal Method," *ACM Trans. Math. Softw.*, Vol. 30, No. 2, Pp. 196–199, Jun. 2004.

[35] R. B. Lehoucq, D. C. Sorensen, And C. Yang, *ARPACK Users' Guide: Solution Of Large-Scale Eigenvalue Problems With Implicitly Restarted Arnoldi Methods*. Philadelphia, PA, USA: SIAM, 1998.

[36] V. Hernandez, J. E. Roman, And V. Vidal, "Slepc: A Scalable And Flexible Toolkit For The Solution Of Eigenvalue Problems," *ACM Trans. Math. Softw.*, Vol. 31, No. 3, Pp. 351–362, Sep. 2005.

[37] D. C. Sorensen, "Implicitly Restarted Arnoldi/Lanczos Methods For Large Scale Eigenvalue Calculations," *SIAM J. Matrix Anal. Appl.*, Vol. 13, No. 1, Pp. 357–385, 1992.

[38] A. Greenbaum, *Iterative Methods For Solving Linear Systems*. Philadelphia, PA, USA: SIAM, 1997.

[39] Y. Saad, "Krylov Subspace Methods For Large Eigenvalue Problems: Theory And Applications," In *Numerical Methods For Large Eigenvalue Problems*, 2nd Ed. Philadelphia, PA, USA: SIAM, 2011, Pp. 133–186.