Cover Letter

I am a postdoctoral scholar at Stanford University. I received my PhD from University of Illinois in July 2022. My research has been in fast algorithms, inverse problems, parallel computing, and programming models.

Research Statement

Mert Hidayetoglu (him/his)

My research interest is high-performance communication in high-end GPU systems in the realm of clusters, data centers, and supercomputers. Specifically, I envision automatic optimization of nonuniform data movement across hierarchical interconnect architectures for large-scale applications that can impact the future course of science and industry.

My dissertation work was on optimizing communication in large-scale applications with sparse data. I proposed hierarchical communications to take advantage of multi-GPU node architectures. I optimized data movement to be tailored to a specific sparsity pattern *and* the underlying system. As supercomputers and data center server systems increasingly adopted multi-GPU architectures, due to their superior compute throughput and power efficiency, I proposed techniques for several high-impact applications to make effective use of these complex designs. My postdoc research is aimed at identifying abstractions that allow developers to reason about the systems, adapt to different systems, and get performance out of them without making every program a one-off exercise. In the rest of this statement, I will talk about how I got here, my postdoc research, and future directions.

How did I get here?

My background is in electrical engineering, where I focused on solving large-scale computational electromagnetics problems. These problems have a large memory footprint, but we had a small computer cluster (16 nodes). I employed an SSD in each node and developed parallel out-of-core techniques to solve integral equations with billions of unknowns (in 2012) for computing radar-cross-sections of aircraft. During my PhD, my specific focus was to solve large-scale inverse problems in imaging. I solved 3D image reconstruction problems with fast algorithms running on the top supercomputers in the world (e.g., Blue Waters, Summit).

The common theme of the applications that I worked on is two-fold. First, they involve large enough data transfers to saturate the data-movement rate (bandwidth) of the system. Second, they involve the repetition of an iterative pattern. My research takes advantage of the hierarchical interconnect architecture by analyzing and memoizing the optimal sparse data movement patterns over high-bandwidth (on-chip) memory so that we can reduce the amount of data moved through "slow" memory, and reuses the optimal pattern in each iteration.

I was fortunate enough to spend summer 2018 at Argonne National Laboratory. There I applied hierarchical communications across GPUs specifically for solving a large-scale X-ray imaging problem [SC19, SC20]. The measurement data from a 3D mouse brain were collected using a particle accelerator (the Advanced Photon Source). This problem required multiplying a large (11 TB) sparse matrix and its transpose in each iteration, and it takes a supercomputer (with at least 768 GPUs) to even hold the data for the problem in memory. We used 4,096 nodes of the Summit supercomputer (24,576 GPUs) and applied hierarchical communications to solve the problem in under four minutes; with standard techniques the problem would have taken hours or even days to solve. The peak performance we achieved of 64 PFLOPS (32% of the theoretical limit) is unprecedented in sparse applications and was primarily due to efficient hierarchical communication.

Postdoc Research

In my PhD work I had great difficulty implementing and optimizing communications tailored to specific applications and systems, and I thought that "this should be easier." My postdoc research is about making developing hierarchical, optimized data movement across GPUs easier with a 1) composable, 2) portable, and 3) performant communication library.

Data movement on supercomputers can be very complex, and the lower-level optimizations vary drastically across systems. The differences between vendors are so significant that using communication software not carefully matched to the hardware can severely affect performance, resulting in wasted time and energy. However, optimizing communication algorithms for specific systems is very tedious with currently available programming models. Thus, my research explores intuitive programming models for communication and automatic optimization techniques that work across GPU systems of diverse architectures and scale.

I (with my collaborators) have already reached a few important milestones.

CommBench: I developed a configurable benchmarking tool to gain performance insights on current systems [CommBench]. We extensively benchmark different communication interconnects and implementation libraries across six different supercomputers (Summit, Delta, Perlmutter, DGX-A100, Frontier, and Aurora), and find optimization opportunities that existing mainstream communication library implementations (MPI, NCCL) do not exploit. For example, we take advantage of multiple network interface cards (NICs) on each node by "striping" the communication data to maximize the communication bandwidth across nodes. CommBench has already been used for acceptance tests for advanced (exascale) system procurement, specifically for testing the implementation libraries and their tuning for communications at each level of the network hierarchy of a specific system.

HiCCL: I used CommBench as a tool to develop a hierarchical communication library (HiCCL) for optimizing any user-defined communication pattern across systems with various architectures and different vendors, currently including Nvidia, AMD, and Intel. Two fundamental data-dependency schemes (one-to-many and many-to-one) across GPUs are used as primitives to express all standard collective functions in a few lines of code. HiCCL then synthesizes optimized communication code automatically for an abstract machine represented with a few (five) parameters. HiCCL uses a level of abstraction that is general enough to work across contemporary node architectures and sufficiently descriptive to take advantage of a specific architecture. I tested HiCCL across the eight most common collective communication patterns on five supercomputers (40 cases) and found that HiCCL utilized approximately 95% of the peak communication throughput. In contrast, the corresponding MPI implementations utilize 10% of all systems, and NCCL functions utilize 80%, where available (12 cases).

Future Directions

I would like to form a group focused on high performance data movement in large-scale applications on both current and future architectures. I will work with students and collaborators with varied backgrounds and expertise to explore a few potential research directions:

1. Foundational machine learning (ML) models are large, static, and repetitive. My research is a perfect candidate to reduce the cost of such demanding applications, such as the gradient exchange communication while training large language models. Furthermore, the training of these models is data-hungry, and once the main operations are optimized, storage and I/O may become the bottleneck. I will extend the hierarchical communication pipeline to distributed disk I/O on supercomputers for streamlining the data flow into the training of foundational models.

2. I am interested in sustainable computing from a systems point of view. Optimizing data movement already saves a lot of energy since it is costlier than computations. Nevertheless, it still takes a supercomputer to solve large problems because of the memory requirements. An under-explored alternative is to use storage systems that have slower bandwidth yet allow solving large problems with smaller compute resources. The challenge is efficient (sparse) memory access to the disk, which I investigated in different contexts. My future research will investigate storage systems for fitting larger problems in smaller systems with a minimal penalty.

3. Dynamic workloads with small messages are the opposite extreme of my focus so far on large and repetitive workloads. This direction is especially challenging because they are typically latency-critical, and we cannot make optimization decisions in advance. Moreover, the sparse data dependencies must be analyzed on-the-fly to enable packing and routing of data in an optimized way to take full advantage of the communication interconnect of the system.

4. In my research, the machine abstractions are parameterized for fitting on systems with different shapes and sizes. Nevertheless, choosing good parameters for a specific system and workload still requires expertise. I will investigate the auto-tuning mechanism, where the proposed tuner queries the system and performs measurements for choosing the optimal parameters for a specific system automatically. We would like to converge to the desired parameters within a few trials, and therefore I will investigate nonlinear optimization schemes and control theory, informed by intuition, for fast convergence.

5. We have already demonstrated sparse hierarchical communication on X-ray image reconstruction at an unprecedented scale. The new generation of light sources, such as LCLS-II at SLAC or APS (Gen-5) at Argonne will be brighter, which means (i) larger amounts of data will be produced and (ii) new imaging techniques will be unveiled. We will use the proposed frameworks to enable novel X-ray imaging techniques on new exascale systems. I will use my collaborations to demonstrate the work in practice.

Long Term

Machines are increasing in complexity, and communication in future data centers will increasingly dominate time and energy costs. Thus there will be challenging and important research problems to address in making communications performant, portable, and programmable for many years to come. Enabling application developers to reduce the costs of data movement without introducing excessive complexity into the applications will be critical for continued progress in computing.

[SC19] MemXCT: Memory-centric X-ray image reconstruction with massive parallelization.

[HPEC20] At-scale sparse deep neural network inference with efficient GPU implementation.

[SC20] Petascale XCT: 3D Image reconstruction with hierarchical communications on multi-GPU nodes. **(best paper)**

[CommBench] https://github.com/merthidayetoglu/CommBench.

Teaching Statement

Mert Hidayetoglu (him/his)

The most important aspect of teaching for me is conveying the insights that are useful to build systems. The best way I have found to do that is to use an algebra to describe computational systems succinctly. For example, we can represent collective communication as a sparse matrix with a sparsity pattern specific to the data movement across GPUs. I show optimization as an algebraic operation, like factorization, which is generalizable to any communication pattern and system architecture. My teaching style connects the dots between special cases in applications and intuitive, generalized principles through mathematical thinking.

Giving lectures and seminars has always been a part of my studies. I was fortunate enough to contribute to my PhD advisor's teaching kit in GPU programming. I prepared lectures that not only have been taught at University of Illinois, but also globally through a MOOC which reached tens of thousands of students. Moreover, I contributed to later editions of "Programming massively parallel processors: A hands-on approach" by Kirk & Hwu, which has become the standard textbook in senior / graduate classes in GPU computing and parallel programming. I traveled every year (2016–2019) to teach at a summer school on GPU programming at the Barcelona Supercomputing Center. I updated the hands-on labs according to the relevant problems each year and gave lectures. I also led tutorials for the dissemination of the frameworks that I have developed over the years. My latest experience was at MLSys 2022 conference, where I curated a tutorial on "Sparsity in ML: Understanding sparsity in neural networks on heterogeneous systems."

I have mentored several undergraduate students in their junior and senior years in CS and CE programs with various research interests related to GPU systems. I involved mentees by carefully guiding them through the learning steps from the classroom to the frontiers of research in computing. I found it very rewarding to contribute to someone's intellectual development. I focused on creative thinking, and finding ways to communicate complex ideas through sketches on a whiteboard, pair programming, and designing example programs. In short, I tried a variety of methods to try to communicate with students in whatever way worked for them.

I would like to teach two graduate-level courses. The first covers fast parallel algorithms for solving large-scale inverse problems with low computational complexity. I would focus on developing numerical analysis and computational thinking by learning how to map physical data structures to supercomputers. The second and related course would involve programming models that make programming supercomputers easier. This course will involve i) standard APIs and language extensions for parallel programming, such as CUDA, MPI, ii) higher-level programming systems such as Legion, Charm++, TACO, and iii) domain-specific languages such as Halide and those I propose for large-scale applications in my own research.

For undergraduate programs, I can teach typical introductory classes in CS, EECS, and ECE curricula. For junior and senior classes, I would especially enjoy teaching parallel programming, vector space optimization, and GPU computing.

Diversity Statement

Mert Hidayetoglu (him/his)

During my graduate studies as an international student, I could relate the importance of inclusion and belonging for well-being and success of a research group. My best example is my doctoral research group, which attracted the best talent in the department to work with us due to our inclusive group culture and the comfort of belonging that kept the students around. My advisor was able to gather a diverse group of people with different talents, whereas monolithic groups lost people. I saw first-hand that diverse groups flourish because each person can contribute with a unique perspective and that diversity allowed us to approach research problems from every possible direction, making our research stronger.

I recognize that we cannot simply have diversity without recognizing that people come from different backgrounds and have different needs. We also need to value equity by being responsive to these different needs in our students and colleagues as much as possible, which can be something as simple as giving an extension for a student who is having a difficult time at home.

Diversity and equity also promotes interesting research. Stanford is one of the most diverse environments I have ever been. I enjoy meeting with interesting people from drastically different walks of life, and therefore with unique perspectives. This environment allows cross-pollination of ideas that is only possible by the inclusive culture, and focusing on productive, interesting research. As a faculty member I would be committed to set the environment so that diversity is celebrated, and everybody is included and welcomed to belong in our working environment.

Mert Hidayetoglu

Postdoctoral Scholar

merth@stanford.edu

merthidayetoglu.github.io

Research Interests Parallel computing, fast algorithms, inverse problems, programming models.

Education

University of Illinois at Urbana-Champaign
Electrical and Computer Engineering
Doctor of Philosophy, July 2022
w/ computational science and engineering concentration.
Advisors: Weng Cho Chew and Wen-mei Hwu

Bilkent University
Electrical and Electronics Engineering
Master of science, April 2015
Advisors: Levent Gürel and Ömer İlday
Bachelor of science, January 2013

Work Experience

		Location
08/2022 – Present	*Postdoctoral Scholar*, Stanford University, Computer Science Department / SLAC National Accelerator Laboratory	Stanford, CA

- Developed a hierarchical communication library for GPU systems.
- Communication benchmarking of exascale computers.
- Adaptive mesh refinement implementation with Legion programming model.

08/2015 – 08/2022	*Research* Assistant, Coordinated Science Laboratory, Computer Systems and Architecture Group	Urbana, IL

- Optimized GPU throughput for large-scale sparse/irregular workloads.
- Designed and implemented a memory-centric algorithm (MemXCT) with hierarchical communications (Petascale XCT) for 3D X-ray image reconstruction.
- Scaled MemXCT up to 4,096 KNLs – 256k cores of ThetaGPU.
- Solved a TB-scale imaging problem on 24,576 GPUs on Summit.
- Won the 2020 MIT/Amazon/IEEE Graph Challenge in sparse DNN inference [Link].
- Optimization of GPU memory accesses for distributed sparse DNN inference.
- Heterogeneous and out-of-core algorithms for spare DNNs with large models.
- Porting performance petascale applications, i.e., HPCG, SETSM, and ChaNGa.
- Developed sparse, hierarchical communications (improves throughput by 60%).

08/2015 – 05/2019	*Research Assistant*, University of Illinois at Urbana-Champaign, Electrical and Computer Engineering Department	Urbana, IL

- Worked on fast and parallel algorithms for scattering problems.
- Designed and implemented a massively parallel inverse multiple-scattering imaging (scaling up to 4,096 GPUs).
- Deployed of large-scale distributed linear and nonlinear optimization methods.
- Developed fast spectral techniques & parallelization for 2.5-dimensional modeling.

05/2019 – 08/2019	*Research Intern*, IBM T. J. Watson Research Center, Data centric systems and high-performance computing group	Cambridge, MA

- Optimizing HPCG benchmark on Summit supercomputer by graph coloring.

05/2018 – 08/2018	*Givens Associate*, Argonne National Laboratory, Data science and learning (DSL) and X-ray science (XSD) divisions	Lemont, IL

- Worked on supercomputing solutions for image reconstruction.

07/2016 – 08/2016	*Research Assistant*, The University of Hong Kong, Department of Electrical and Electronic Engineering	Hong Kong S.A.R., China

- On-site collaboration with computational electromagnetics group.

08/2012 – 08/2015	*Co-Founder & Staff*, ABAKUS Computing Technologies	Ankara, Turkey

- Conducted industry- and government-funded research projects.
- Organized of CEM'15 Computational Electromagnetics Workshop. İzmir, Turkey
- *See other duties under BiLCEM*.

12/2010 – 10/2014	*Research Assistant*, Bilkent University, Department of Electrical and Electronics Engineering, Computational Electromagnetics Research Center (BiLCEM)	Ankara, Turkey

- Development of novel and parallel out-of-core electromagnetics solvers.
- Accurate and fast solutions of large-scale electromagnetics problems.

- Implementation of iterative solvers and preconditioners for solutions of extremely large dense linear systems (up to $N = 2.1$ billion unknowns!).
- Assistant in CEM'13 Computational Electromagnetics Workshop. İzmir, Turkey

06/2011 – 09/2011 *Summer Intern*, BiLCEM Ankara, Turkey
- Developed of a parallel mesh refinement code for large-scale geometries.
- Assistant in CEM'11 Computational Electromagnetics Workshop. İzmir, Turkey

06/2010 – 07/2010 *Summer Intern*, ETA Electronic Design Inc. Ankara, Turkey
- Implementation and documentation of a testing software for a power distribution system (of MILGEM Turkish cruiser).

Teaching Experience

Co-developer, NVIDIA Accelerated Computing Teaching Kit – Multi-GPU Systems [Link] Urbana, IL
- Made global impact by reaching to tens of thousands of students through MOOC.

Tutorial Organizer, Conference on Machine Learning and Systems (MLSys 2022). Santa Clara, CA Aug. 2022
- Sparsity in ML: Understanding and optimizing sparsity in neural networks running on heterogeneous Systems [Link]

Summer School Organizer, Barcelona Supercomputing Center, Programming and Tuning Massively Parallel Systems + Artificial Intelligence (PUMPS+AI) Barcelona, Spain 2016—present
- Lecture: Scalable Algorithms and Supercomputing Applications.
- Mentor of a clinic case study: Lattice Boltzman method for multi-GPU clusters.
- Lecture: Massively parallel heterogeneous computing.
- Mentor of a clinic case study: Runge-Kutta type integrator scheme for a stochastic Schrödinger equation **(mentee won the best poster award and a GPU).**

Mentor, University of Illinois at Urbana-Champaign, Coordinated Science Laboratory Urbana, IL
- Mentored undergraduate students on the IBM's C3SR Undergraduate Research in AI (URAI) and Discovery Accelerator Institute (IIDAI) programs.

Teaching Assistant, University of Illinois at Urbana-Champaign Urbana, IL
Department of Electrical and Computer Engineering
- ECE 408 Applied Parallel Programming guest lecture: Programming GPU Cluster Dec. 2019
- ECE 508 Manycore Parallel Algorithms Spring 2019
- ECE 408 Applied Parallel Programming guest lecture: Programming GPU Cluster Nov. 2018
- ECE 350 Fields and Waves II Spring 2017

Instructor, National Center for Supercomputing Applications (NCSA) Urbana, IL
- Thought a crash course on CUDA to National Geospatial-Intelligence Agency Nov. 2018
- Mentor to Lawrence Berkeley Lab for NCSA GPU Hackaton Sept. 2018

Teaching Assistant, Bilkent University Ankara, Turkey
Department of Electrical and Electronics Engineering
- EEE 212 Microprocessors (3 Semesters) 01/2013 –
- EEE 202 Circuit Theory 04/2015
- EEE 491 Electrical and Electronics Engineering Design

Undergraduate Tutor, Bilkent University Summer 2012
Academic Student Coordination Unit
- CS 114 Introduction to Programming for Engineers

Coordinator and Lecturer, IEEE Student Branch 11/2011 –
and Computational Electromagnetics research Center (BiLCEM) 05/2012
- Introduction to Unix/Linux, FORTRAN, parallel computing, parallel programming, and MATLAB classes

Honors, Awards, and Recognitions

ACM/IEEE-CS George Michael Memorial HPC Fellowship 2021
IBM-Illinois C3SR Best Research Recognition
ECE Illinois Yi-Min Wang and Pi-Yu Chung Research Award 2020
SC20 Best Paper – Winner (Lead author)
ACM Student Research Competition – SC20 1st Place
ACM SIGHPC Certificate of Appreciation, 2020
MIT/Amazon/IEEE Sparse DNN Graph Challenge Champion 2020 (Lead author)
HPCC Best Paper Award 2019
IEEE TCPP / NSF Travel Grant for IPDPS 2019
ECE Illinois Paul D. Coleman Outstanding Research Award 2018
Argonne National Laboratory Givens Fellowship, Class of 2018
IPDPS PhD Forum Outstanding Poster Presentation 2018 (by public voting, presenter)

Grainger College of Engineering Computational Science and Engineering Fellow, Class of 2018
National Academies Travel Grant for USNC-URSI 2017, 2018
ECE Illinois Dan Vivoli Endowed Fellowship 2017
ECE Illinois Professor Kung Chie Yeh Endowed Fellowship 2016
Turkcell Technology Leaders Graduate Scholarship Program, Class of 2014
TÜBİTAK Graduate Research Scholarship (2013–2014)
Bilkent University EEE Department Research Excellence Award 2013
BiLCEM undergraduate research fellowship (2011–2013)

Professional Service

TPC Member, IEEE IPDPS Programming Models, Compilers and Runtime Systems [Link]
SC20 SCC Reproducibility Challenge Benchmark Lead Author [Link]
ICCEM 2020 Organizer and Co-Chair of Special Session on Complex Inverse Problems [Link]
IPDPS 2020 Proceedings Vice-Chair for PhD Forum (Cancelled due to COVID-19 Pandemic)
Session Assistant at 2019 (57th) Allerton Conference
Volunteer Student Assistant IPDPS 2018
Co-organizing CEM'17 Int. Computing and Electromagnetics Workshop
Volunteer Student Assistant 2014 IEEE AP-S/URSI Symposium

Reviewing & Editing Activities

ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)
IEEE Transactions on Antennas and Propagation
IEEE Antennas and Propagation Magazine
IEEE International Conference on Computational Electromagnetics (ICCEM)
International Workshop on Computing, Electromagnetics, and Machine Intelligence
IEEE International Parallel and Distributed Processing Symposium (IPDPS, PC Member)
International Symposium on Computer Architecture (ISCA)
Elsevier Parallel Computing (PARCO)
D. Kirk and W.-M. W. Hwu, *Programming Massively Parallel Processors.* 4th ed., 2021.

Involved Centers & Projects

Industry	**Supporter**
Computational Methods for Antennas Mounted on Platforms (PLANT-I)	ASELSAN-SSM
Jet Trainer/Fighter Radar Cross Section Analysis (FX/TX)	TAI-SSM
Radar Cross Section Calculations of Chaff Clouds	ASELSAN
NVIDIA Center of Excellence - UIUC	NVIDIA
Center for Cognitive Computing Systems Research (C3SR)	IBM
Applications Driving Architectures (ADA) Center	SRC-DARPA
Government	
Breast Cancer Detection via Inverse Scattering Algorithms	TÜBİTAK
Parallel Electromagnetic Equivalence Principle Algorithm	TÜBİTAK
Petascale Application Improvement Discovery (PAID-IME)	NSF-NCSA
Sustained-Petascale In Action: Blue Waters Enabling Transformative Science and Engineering	NSF
Vancouver II: Improving Programmability of Contemporary Heterogeneous Architectures	DOE
Leadership Class Scientific and Engineering Computing: Breaking Through the Limits	NSF
High Accuracy, Broadband Simulation of Complex Structures with Quantum Effects, Parallel Fast Algorithm, and Integral Equation Domain Decomposition	NSF
Rapid Analysis of Various Emerging Nanoelectronics (RAVEN)	IARPA
CORAL: Collaboration of Oak Ridge, Argonne, and Lawrence Livermore EPC: Exascale Computing Project	DOE
University	
Alchemy: University Technology Foundry	UIUC
A New Paradigm in Ultrasonic Image Formation: Inverse Scattering	UIUC

ASELSAN: *Military Electronic Industries Inc. (of Turkey)*
SSM: *Undersecretariat for Defense Industries (of Turkey)*
TAI: *Turkish Aircraft Industries Inc.*
TÜBİTAK: *Scientific and Technological Research Council of Turkey* (NSF of Turkey)

Book Chapter

W. C. Chew, Q. I. Dai, Q. S. Liu, T. Xia, T. E. Roth, H. Gan, A. Liu, S. C. Chen, **M. Hidayetoglu**, L. J. Liang, S. Sun, and W.-M. Hwu, *New Trends in Computational Electromagnetics.* Ö. Ergül, Ed. London: The Institute of Engineering and Technology, Dec. 2019.

Journal Papers

M. Hidayetoğlu, T. Biçer, S. Garcia de Gonzalo, B. Ren, D. Gürsoy, R. Kettimuthu, I. T. Foster, and W.-M. W. Hwu, "MemXCT: Design, optimization, scaling, and reproducibility of X-ray tomography imaging," *IEEE Trans. Parallel Distrib. Sys. (IEEE TPDS)*, vol. 33, no. 9, 2014–2031, Sep. 2022.

L. L. Meng, **M. Hidayetoğlu**, T. Xia, Wei E. I. Sha, L. J. Jiang, and W. C. Chew, "A wide-band two-dimensional fast multipole algorithm with a novel diagonalization form," *IEEE Trans. Antennas Propag. (IEEE TAP)*, vol. 66, no. 12, pp. 7477–7482, Dec. 2018.

D. J. Ching, **M. Hidayetoğlu**, T. Biçer, and D. Gürsoy, "Rotation-as-fast-axis scanning-probe x-ray tomography: the importance of angular diversity for fly-scan modes," *Appl. Opt.*, vol. 57, no. 30, pp. 8780–8789, Oct. 2018.

M. Hidayetoğlu, M. Oelze, E. Kudeki, and W. C. Chew, "Fast numerical integration techniques for 2.5-Dimensional Inverse Problems," *IEEE J. Multiscale Multiphysics Comput. Tech.*, on revision. [Arxiv]

Conference Papers

*Presenting Author

S. W. Min, K. Wu, **M. Hidayetoğlu**, J. Xiong, Xiang Song, and W.-M. Hwu, "Graph Neural Network Training with Data Tiering,"*ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'22)*, Washington, DC, Aug. 2022. [Arxiv]

S. Duranni, M. S. Chughati, **M. Hidayetoglu**, R. Tahir, A. Dakkak, L. Rauchwerger, F. Zaffar, W.-m. Hwu, "Accelerating Fourier and number-theoretic transforms using tensor cores and warp shuffles," *Int. Conference on Parallel Architectures and Compilation Techniques (PACT 2021)*, Sep. 2021.

S. W. Min, K. Wu, S. Huang, **M. Hidayetoğlu**, J. Xiong, E. Ebrahimi, D. Chen, W.-m. Hwu, "Large graph convolutional network training with GPU-oriented data communication architecture," *International Conference on Very Large Data Bases (VLDB'21)*, Copenhagen, Denmark, Aug. 2021. [Arxiv] **(Implemented in AWS Deep Graph Library v0.8)**

M. Hidayetoğlu*, T. Bicer, S. Garcia de Gonzalo, B. Ren, V. De Andrade, D. Gursoy, R. Kettimuthu, I. T. Foster, and W.-M. W. Hwu, "Petascale XCT: 3D image reconstruction with hierarchical communications on multi-GPU nodes," *The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC20)*, Atlanta, GA, Nov. 2020. **(Best Paper - winner)** [Arxiv]

M. Hidayetoğlu*, C. Pearson, V. S. Mailthody, E. Ebrahimi, J. Xiong, R. Nagi, and W.-M. Hwu, "At-scale sparse deep neural network inference with efficient GPU implementation," *IEEE High Performance Extreme Computing (HPEC'20)*, Waltham, MA, Sep. 2020. **(Graph Challenge Champion)** [Arxiv]

M. Hidayetoğlu*, T. Biçer, S. Garcia de Gonzalo, B. Ren, D. Gürsoy, R. Kettimuthu, I. T. Foster, and W.-M. W. Hwu, "MemXCT: Memory-centric X-ray CT reconstruction with massive parallelization," *The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC19)*, Denver, CO, Nov. 2019. **(~21% acceptance rate, SC20 reproducibility challenge benchmark)**

O. Anjum, S. Garcia de Gonzalo, **M. Hidayetoğlu**, and W.-M. Hwu*, "An efficient GPU implementation technique for higher-order 3D stencils," *Int. Conf. High Performance Computing and Communications (HPCC-2019)*, Zhangjiajie, China, Aug. 2019. (**~19% acceptance rate won the best paper award)**

M. Hidayetoğlu*, C. Pearson, I. El Hajj, L. Gürel, W. C. Chew, and W.-M. Hwu, "A fast and massively-parallel inverse solver for multiple-scattering tomographic image reconstruction," *IEEE Int. Parallel Distributed Processing Symp. (IPDPS 2018)*, Vancouver, Canada, May 2018. (**~20% acceptance rate)**

Workshop Papers & Extended Abstracts

C. Pearson*, **M. Hidayetoğlu**, M. Almasri, O. Anjum, I-H. Chung, J. Xiong, and W.-M. Hwu, "Node-aware stencil communication for heterogeneous supercomputers," *Int. Workshop on Automatic Performance Tuning (iWAPT 2020 - IPDPS Workshop)*, New Orleans, LA, May 2020.

M. Hidayetoğlu*, W.-M. Hwu, and W. C. Chew, "Supercomputing for full-wave tomographic image reconstruction in near-real time," *IEEE Int. Symp. on Antennas and Propagation and USNC-URSI Radio Science Meeting (AP-S/URSI 2018)*, Boston, MA, July 2018.

M. Hidayetoğlu*, W.-M. Hwu, and W. C. Chew, "Seeing the invisible: limited-view imaging with multiple-scattering reconstruction," *USNC-URSI Nat. Radio Science Meeting*, Boulder, CO, Jan. 2018.

M. Hidayetoğlu*, C. Pearson, I. El Hajj, W. C. Chew, L. Gürel, and W.-M. Hwu, "Scaling analysis of large inverse multiple-scattering solutions," *The International Conference on High Performance Computing, Networking, Storage and Analysis (SC17)*, Denver, CO, Nov. 2017.

W.-M. Hwu*, **M. Hidayetoğlu**, W. C. Chew, C. Pearson, S. Garcia, S. Huang, and A. Dakkak, "Thoughts on massively-parallel heterogeneous computing for solving large problems," *CEM'17 Computing and Electromagnetics Int. Workshop*, Barcelona, Spain, June 2017.

M. Hidayetoğlu*, C. Pearson, L. Gürel, W.-M. Hwu, and W. C. Chew, "Scalable parallel DBIM solutions of inverse-scattering problems," *CEM'17 Computing and Electromagnetics Int. Workshop*, Barcelona, Spain, June 2017.

C. Pearson*, **M. Hidayetoğlu**, Wei Ren, W. C. Chew, and W.-M. Hwu, "Comparative performance evaluation of multi-GPU MLFMM implementation for 2-D VIE problems," *CEM'17 Computing and Electromagnetics Int. Workshop*, Barcelona, Spain, June 2017.

M. Hidayetoğlu*, C. Pearson, W. C. Chew, L. Gürel, and W.-M. Hwu, "Large inverse-scattering solutions with DBIM on GPU-enabled supercomputers," *Applied and Computational Electromagnetics Symp. (ACES 2017)*, Florence, Italy, Mar. 2017.

M. Hidayetoğlu, C. Yang, L. Wang, A. Podkowa, M. Oelze, W.-M. Hwu, and W. C. Chew*, "Large-scale inverse scattering solutions with parallel Born-type fast solvers (Invited)," *Progress on Electromagnetics Research Symp. (PIERS 2016)*, Shangai, China, Aug. 2016.

M. Hidayetoğlu and W. C. Chew*, "On computational complexity of the multilevel fast multipole algorithm in various dimensions," *IEEE Int. Symp. on Antennas and Propagation/USNC-URSI Nat. Radio Science Meeting (AP-S/URSI 2016)*, Fajardo, Puerto Rico, June 2016.

M. Hidayetoğlu and L. Gürel*, "Full-wave and approximate solutions of large electromagnetic scattering problems," *IEEE Int. Symposium on Antennas Propagation and North American Radio Science Meeting (AP-S/URSI 2015)*, Vancouver, Canada, July 2015.

M. Hidayetoğlu* and L. Gürel, "An MPIxOpenMP implementation of the hierarchical parallelization of MLFMA," *Computational Electromagnetics Int. Workshop (CEM'15)*, Izmir, Turkey, July 2015.

M. Hidayetoğlu and L. Gürel*, "Parallel out-of-core MLFMA on distributed-memory computer architectures," *Computational Electromagnetics Int. Workshop (CEM'15)*, Izmir, Turkey, July 2015.

M. Salim*, A. O. Akkirman, **M. Hidayetoğlu**, and L. Gürel, "Comparative benchmarking: matrix multiplication on a multicore processor and a GPU," *Computational Electromagnetics Int. Workshop (CEM'15)*, Izmir, Turkey, July 2015.

M. Hidayetoğlu* and L. Gürel, "MLFMA memory reduction techniques for solving large-scale problems," *2014 IEEE Int. Symp. on Antennas and Propagation and USNC-URSI National Radio Science Meeting (AP-S/URSI)*, Memphis, TN, July 2014.

M. Hidayetoğlu*, B. Karaosmanoğlu, and L. Gürel, "Reducing MLFMA memory with out-of-core implementation and data-structure parallelization," *Computational Electromagnetics Int. Workshop (CEM'13)*, İzmir, Turkey, Aug. 2013.

Invited Talks

Generalized hierarchical communication, Stanford University, Department of Computer Science, Stanford, CA, Sep. 2023. [link]

Optimizing collective communications on hierarchical networks, Barcelona Supercomputing Center, Spain, July 2023.

Performance modeling of sparse matrix multiplication, Stanford University Department of Computer Science, Stanford, CA, Oct. 2022. [Link]

DOE Seminar series on large-scale X-ray tomography on synchrotron accelerator light sources

- CIDR Seminar, Los Alamos National Laboratory, 22 Oct. 2020. Host: Brendt Wohlberg
- XCT Interest Group, Lawrence Berkeley National Laboratory, 18 Nov. 2020. Host: Dula Parkinson

Memory-centric, low complexity image reconstruction for the exascale era of computing, Bilkent University Computer Engineering Department, Ankara, Turkey, Jan. 2020. [Link]

Supercomputing for full-wave tomographic image reconstruction in near-real time, National Magnetic Resonance Research Center, Ankara, Turkey, Sep. 2018. [Link]

Low complexity, petascale, heterogeneous inverse solvers on Blue Waters, Coordinated Science Laboratory, Urbana, IL, Feb. 2018. [Link]

Fast and parallel algorithms for large full-wave image reconstructions, Argonne National Laboratory, Lemont, IL, Dec. 2017. [Link]

Fast and parallel algorithms for inverse multiple-scattering solutions and applications on tomographic imaging, National Center for Supercomputing Applications, Urbana, IL, Sep. 2017. [Link]

Fast and parallel algorithms for multiple-scattering imaging, The University of Hong Kong, Hong Kong S.A.R., China, Aug. 2016. [Link]

Conference Talks

M. Hidayetoğlu*, "Large-scale inverse multiple-scattering imaging on GPU supercomputers with real data," *The 1st Conference on High Performance Computing on Imaging (HPCI), Invited Speaker*. San Francisco, CA, Jan. 2022.

M. Hidayetoğlu*, "Hierarchical Communications for 3D image reconstruction with synchrotron light source and 24,576 GPUs," *The 1st Conference on High Performance Computing on Imaging (HPCI), Invited Speaker*. San Francisco, CA, Jan. 2022.

M. Hidayetoğlu*, W.-M. Hwu, and W. C. Chew, "High performance inverse multiple-scattering imaging," IEEE Int. Conf. Computational Electromagnetics *(ICCEM 2020)*, Singapore, Aug. 2020.

M. Hidayetoğlu, W.-M. Hwu, and W. C. Chew*, "Efficient integration paths for fast 2.5-D Scattering," *Progress in Electromagnetics Research Symp. (PIERS 2018)*, Toyama, Japan, Aug. 2018.

L. L. Meng*, **M. Hidayetoğlu**, T. Xia, W. C. Chew, W. E. I. Sha, and L. J. Jiang, "A novel diagonalization in two-dimensional fast multipole algorithm based on discrete Fourier transform," *Progress on Electromagnetics Research Symp. (PIERS 2017)*, Singapore, Nov. 2017.

W.-M. Hwu*, **M. Hidayetoğlu**, C. Pearson, S. Garcia, S. Huang, and A. Dakkak, "Massively-parallel heterogeneous computing for solving large problems," *CEM'17 Computing and Electromagnetics Int. Workshop*, Barcelona, Spain, June 2017. **(Plenary Talk)**

M. Hidayetoğlu*, A. Podkowa, M. Oelze, W.-M. Hwu, and W. C. Chew, "Fast DBIM solutions on supercomputers with frequency-hopping for imaging of large and high-contrast objects," *Progress on Electromagnetics Research Symp. (PIERS 2017)*, St. Petersburg, Russia, May 2017.

M. Hidayetoğlu*, A. Podkowa, M. L. Oelze, L. Gürel, W.-M. Hwu, and W. C. Chew, "Incorporating multiple scattering in imaging with iterative Born methods," *USNC-URSI Nat. Radio Science Meeting*, Boulder, CO, Jan. 2017.

A. Podkowa*, **M. Hidayetoğlu**, W. C. Chew, and M. Oelze, "Reconstruction of spatially varying sound speed distributions from pulse-echo data," *Meeting Acoustic Society America*, Honolulu, HI, Dec. 2016.

M. Hidayetoğlu and L. Gürel*, "Accelerating hybrid integral-equation and physical-optics solutions with MLFMA," *URSI Atlantic Radio Science Conf. (AT-RASC 2015)*, Gran Canaria, Spain, May 2015.

Posters & Other Presentations

M. Hidayetoğlu and W.-M Hwu (advisor), "Memory-centric 3D image reconstruction with hierarchical communications on multi-GPU node architecture," ACM *Student Research Competition (SRC) of SC20*, Atlanta, GA, Nov. 2020. **(Won the ACM Student Research Competition at SC20)**

S. L. Harrel, M. Taufer, B. Plale, V. M. Vergara, S. Michael, **M. Hidayetoglu**, and T. Bicer, SC20 vSCC Reproducibility Challenge, Aug. 2020. [Link]

M. Hidayetoğlu, Efficient inference on GPUs for the sparse deep neural network challenge 2020, IBM-Illinois Center for Cognitive Systems Research, Urbana, IL, Jul. 2020.

M. Hidayetoglu, Memory-Centric 3D Image Reconstruction on 24,576 GPUs, IBM-Illinois Center for Cognitive Systems Research, Urbana, IL, May 2020.

M. Hidayetoğlu, Remedies Towards Breaching Memory Wall for Sparse Computations, IBM-Illinois Center for Cognitive Systems Research, Urbana, IL, Oct. 2019. [Slides]

M. Hidayetoğlu, Mohammad Al Masri, Carl Pearson, Jinjun Xiong, Rakesh Nagi, Wen-mei W. Hwu, "Efficient sparse veryDNN Inference," *IBM-Illinois C3SR Open House*, Urbana, IL, Oct. 2019.

M. Hidayetoğlu, T. Biçer, S. Garcia de Gonzalo, B. Ren, D. Gürsoy, R. Kettimuthu, W. C. Chew, I. Foster, and W.-M. Hwu, "Memory-centric iterative X-ray image reconstruction," *PhD Forum of IPDPS 2019*, Rio de Janeiro, Brazil, May 2019.

M. Hidayetoğlu, C. Pearson, I. El Hajj, W. C. Chew, L. Gürel, and W.-M. Hwu, "Large and massively-parallel image reconstruction accelerated with the multilevel fast multipole algorithm," *PhD Forum of IPDPS 2018*, Vancouver, Canada, May 2018. **(Won the second place among 32 posters.)**

M. Hidayetoğlu, W. C. Chew, and W.-M. Hwu, "Scalable full-wave image reconstruction on Blue Waters," *Coordinated Science Laboratory Student Research Conference (CSLSC)*, Urbana, IL, Feb. 2018.

M. Hidayetoğlu and W.-M. Hwu, "Massively-parallel full-wave (nonlinear) tomographic imaging," *Supercomputing (SC17)*, Denver, CO, Oct. 2017 (showcase for Illinois Parallel Computing Institute.)

M. Hidayetoğlu, C. Pearson, W.-M. Hwu, and W. C. Chew, "A 2-D volume equation solver on GPU for solutions of light scattering problems," *International Year of Light at UIUC*, Urbana, IL, USA, Sep. 2015.

M. Hidayetoğlu and Ö. İlday, "A parallel physical optics solver for solving large-scale electromagnetics scattering problems," *Bilkent IEEE Grad. Research Conf. (GRC'15)*, Ankara, Turkey, Mar. 2015.

M. Hidayetoğlu and L. Gürel, "Hybrid PO-MoM solutions of electromagnetic scattering problems involving PEC geometries," *Bilkent IEEE Grad. Research Conf. (GRC'14)*, Ankara, Turkey, Mar. 2014.

M. Hidayetoğlu and L. Gürel, "Memory reduction by parallelizing data structures of MLFMA," *Bilkent IEEE Graduate Research Conference (GRC'13)*, Ankara, Turkey, Mar. 2013.

M. Hidayetoğlu, B. Karaosmanoğlu, and L. Gürel, "MLFMA solutions of electromagnetic scattering from chaff clouds," *Bilkent IEEE Graduate Research Conference (GRC'12)*, Ankara, Turkey, Mar. 2012.

Dissertation

M. Hidayetoğlu, "Large-scale solutions of electromagnetics problems using the multilevel fast multipole algorithm and physical optics," M.S. Thesis, Bilkent University, Ankara, Turkey, Apr. 2015.

M. Hidayetoğlu, "Hierarchical sparse computations and communications for solving inverse problems on supercomputers with multi-GPU nodes," Ph.D. Dissertation, University of Illinois at Urbana-Champaign, Urbana, USA, July 2022.

Featured News & Stories

Bilkent News, *Mert Hidayetoglu receives ACM/IEEE-CS George Michael Memorial HPC Fellowship*, Nov. 2021 **[**Link]

ACM/IEEE-CS George Michael Memorial HPC Fellowship, Oct. 2021

- **HPC Wire** [Link]
- **ACM News** [Link]
- **IEEE-CS News** [Link]

IBM-Illinois C3SR Newsletter, *C3SR Team named MIT/Amazon/IEEE Graph Challenge champion for accelerating sparse neural network inference on Summit*, Apr. 2021. [Link]

CSL News, *CSL students lead interdisciplinary team, continue to earn accolades*, Feb. 2021 [Link].

SC20 Best Paper Award News, Nov. 2020

- **SC20 Newsletter** [Link]
- **Inside HPC** [Link]
- **Barcelona Supercomputing Center** [Link]
- **Scientific Computing World** [Link]
- **Argonne National Laboratory** [Link]
- **EurekAlert (AAAS)** [Link]
- **Newswise** [Link]
- **HPC Wire** [Link]

CSL News, *CSL team crowned IEEE HPEC Graph Challenge champions*, Oct. 2020. [Link]

CSL News, *CSL student's paper selected for international reproducibility competition*, May 2020. [Link]

SC20 Newsletter, *SC20 Student Cluster Reproducibility Committee chooses benchmark wisely*, Apr. 2020. [Link]

APS Science 2017, *Real-time data analysis and experimental steering at the APS using large-scale computing*, Aug. 2018. [Link]

HPC Wire, *34 University of Illinois researcher teams awarded allocations on Blue Waters supercomputer*, June 2018. [Link]

Blue Waters Annual Report, *Parallelization of the multilevel fast multipole algorithm (MLFMA) on heterogeneous CPU-GPU architectures*, 2017. [Link]

ECE Illinois Newsletter and CSL News, *Hidayetoğlu tackles complex imaging as CSE Fellow*, June 2017. [ECE Link], [CSL Link]

Blue Waters Annual Report, *Parallelization of the multilevel fast multipole algorithm (MLFMA) on heterogeneous CPU-GPU architectures*, 2016. [Link]

Bilkent News, *BiLCEM researchers making aircraft stealthier*, Mar. 2014. [Link]

Petascale XCT: 3D Image Reconstruction with Hierarchical Communications on Multi-GPU Nodes

Mert Hidayetoğlu, Tekin Bicer[†], Simon Garcia de Gonzalo[‡], Bin Ren[§], Vincent De Andrade[†], Doga Gursoy[†], Raj Kettimuthu[†], Ian T. Foster[†], and Wen-mei W. Hwu

University of Illinois at Urbana-Champaign, USA
[†]Argonne National Laboratory, IL, USA,
[‡]Barcelona Supercomputing Center, Spain
[§]College of William & Mary, VA, USA

***Abstract*—X-ray computed tomography is a commonly used technique for noninvasive imaging at synchrotron facilities. Iterative tomographic reconstruction algorithms are often preferred for recovering high quality 3D volumetric images from 2D X-ray images, however, their use has been limited to small/medium datasets due to their computational requirements. In this paper, we propose a high-performance iterative reconstruction system for terabyte(s)-scale 3D volumes. Our design involves three novel optimizations: (1) optimization of (back)projection operators by extending the 2D memory-centric approach to 3D; (2) performing hierarchical communications by exploiting "fat-node" architecture with many GPUs; (3) utilization of mixed-precision types while preserving convergence rate and quality. We extensively evaluate the proposed optimizations and scaling on the Summit supercomputer. Our largest reconstruction is a mouse brain volume with 9K×11K×11K voxels, where the total reconstruction time is under three minutes using 24,576 GPUs, reaching 65 PFLOPS: 34% of Summit's peak performance.**

I. Introduction

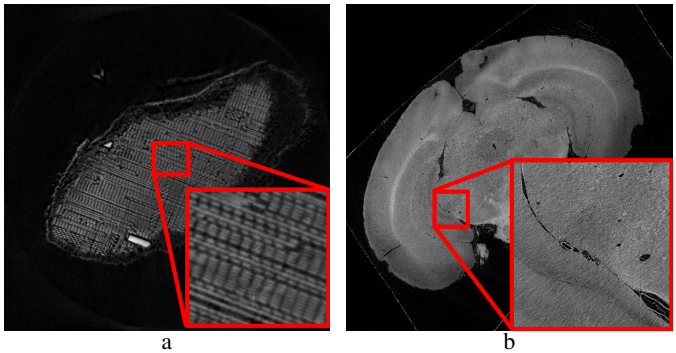
Synchrotron light source facilities around the world help tens of thousands of researchers every year carry out extremely challenging experiments and ground-breaking research. X-ray computed tomography (XCT) is one of the widely used imaging modalities at synchrotron light sources for imaging materials, samples and biological specimens in 3D with high temporal and spatial resolution, ranging from several micrometers down to sub-20 nanometer resolution. The high-energy synchrotron X-ray sources, such as the Advanced Photon Source (APS) at Argonne National Laboratory (ANL), enable imaging thick specimens that can yield massive amounts of measurements, exceeding tens of GB/s rates and producing TBs-scale data per experiment [1]. For example, imaging a single adult mouse brain of a few centimeters diameter at μm resolution requires a "tiled" tomography experiment that produces more than 1.7 TB (9K×11K images with 4.5K angles) measurement data. Further, the reconstruction of such data generates more than 4.3 TB 3D volumetric image (with 9K×11K×11K voxels) [2]. High-performant scalable solvers are needed to reconstruct these large experimental datasets, especially considering that advancements in domain sciences, such as neuroscience, require imaging and reconstruction of many of these samples.

Due to experimental constraints, such as extreme conditions (high radiation dose) or physical limitations (vibrations or drifts during data acquisition), produced data can be far from the ideal and can consist of noisy images with undesired artifacts. These imperfect measurements can adversely impact the reconstruction process resulting in unsatisfactory output. The ability to mitigate such noise and artifacts are important considerations when it comes to choosing between the two broad categories of reconstruction approaches: analytical methods and iterative solvers. While analytical methods, such as filtered-backprojection, are typically fast algorithms, they produce sub-optimal reconstructions with imperfect (noisy) measurement data. In contrast, iterative tomographic reconstruction solvers enable integration of advanced regularizers and models, and iteratively reach for a solution by solving an optimization problem. They also provide an essential framework for incorporating imperfections into the model and therefore, mitigate these artifacts and achieve the desired resolution. However, such improved reconstruction comes at the expense of significantly higher computational cost.

Iterative reconstruction solvers have so far been used for small/medium scale tomography datasets mostly due to the aforementioned computational costs. Different parallelization methods have been developed to ease this cost, from naive data-parallel approaches that process different slices of the measurement data (sinograms) in parallel to advanced in-slice and memory-centric approaches [3]–[5]. For parallel beam geometry, sinogram-based reconstruction methods exploit data parallelism by reconstructing each 2D slice (sinogram) independently. After the reconstruction is done, all reconstructed slices (tomograms) are gathered to a 3D volume. This approach provides reasonable execution time for most smaller datasets, but larger datasets require more aggressive parallelization. In-slice parallelization techniques improve the speed of single sinogram reconstruction by distributing parts of a sinogram to several processes, but introduce synchronization and communication overheads during iterations since the processes that are involved in the processing of the same sinogram need to combine their intermediate results. The MemXCT advanced memory-centric parallelization technique [4] mitigates the synchronization and communication overheads by using memoization, and provides efficient sparse matrix representations and communication patterns.

While the MemXCT parallelization technique provides sig-

SC20, November 9-19, 2020, Is Everywhere We Are
978-1-7281-9998-6/20/$31.00 ©2020 IEEE



a b

Fig. 1: (a) Tilted slice of an IC chip reconstruction and (b) horizontal slice of a mouse brain reconstruction.

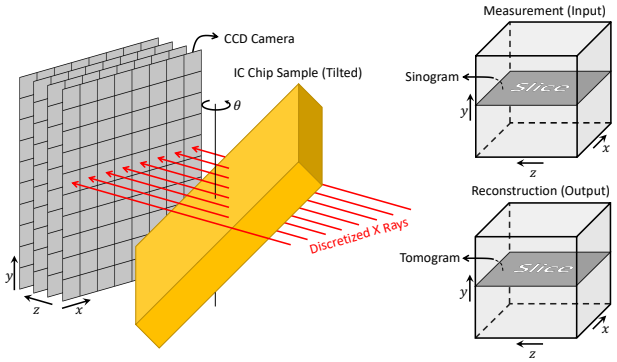


Fig. 2: An experimental setup that shows tomographic data acquisition.

nificant performance improvement compared to its alternatives, reconstruction of full-sized volumes of extreme-scale samples still requires long processing time. For example, authors in [4] report that reconstruction of a single mouse brain sinogram (a slice of the measurement data) requires 10 secs using 256K-cores at Theta supercomputer at ANL. The full reconstruction of the sample (9K sinograms) requires more than 25 hours with the whole supercomputer. Fig. 1(a) and Fig. 1(b) show reconstructions of a medium-scale integrated circuit (IC) and a large-scale mouse brain tomograms (slices of 3D images), respectively. The reconstruction quality is extremely important to distinguish features for these samples (transistors and wires for IC, and blood vessels and myelinated axon tracts for brain sample), therefore iterative solvers are preferred method for reconstruction. Note that a typical science study will likely require the full reconstruction of many samples.

Large-scale GPU resources, such as the Summit supercomputer at the Oak Ridge National Laboratory (ORNL) provide opportunities to apply iterative reconstruction algorithms to extremely large tomography datasets. However, further optimizations and advanced parallelization techniques must be used to achieve maximum efficiency on such resources. In this paper, we introduce novel optimizations for state-of-the-art memory-centric iterative reconstruction approaches to enable reconstruction of extremely large tomography datasets. Specifically, we make the following contributions:

- We introduce improved data partitioning and parallelization techniques that extend 2D MemXCT *data parallelism* with *3D batch parallelism*. Our approach enables optimized SpMM operations that perform common computational kernels on different voxels (fusing), while performing memoization of irregular data accesses at 3D space (compared to 2D plane in MemXCT).
- We present a hierarchical communication strategy that exploits multi-GPU node architecture. Our approach leverages asynchronous multi-level data reduction to minimize communication overhead between nodes.
- We propose mixed-precision implementation that performs computations using single (or double) precision operations, whereas it stores and communicates data in half precision for reduced memory and communication footprint.
- We provide a comprehensive evaluation of our optimizations and system with four real-world tomography datasets using up to 24K NVIDIA V100 GPUs and demonstrate sustained peta-scale performance. In particular, we reconstruct a mouse brain volume with 9K11K11K voxels within 3 minutes using the whole Summit supercomputer, reaching 65 PFLOPS throughput (or 34% of Summit's theoretical peak performance).

II. Background

In this section, we briefly explain tomographic data acquisition with synchrotron light sources and the basics of iterative reconstruction process.

A. Tomography Experiments

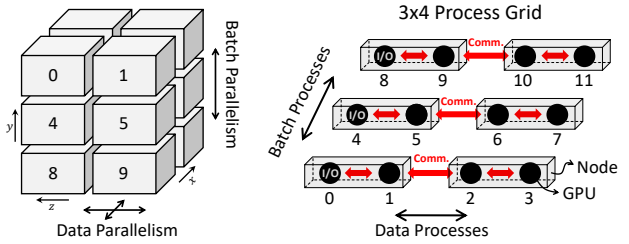
During a tomography experiment, a sample is placed on top of a rotation stage and exposed to X-ray beams. As X-ray beams travel through sample, the photons are attenuated by the sample according to Beer-Lambert law [6], [7]. The attenuated beams are, then, measured at the detector and an X-ray *projection* of the sample is recorded at the detector, as illustrated in Fig. 2. This process is repeated for different rotational views of the sample, θ, with the aim of meeting Crowther criterion [8]. Consequently, this process generates a set of projections, p_θ, where, for example, $\theta = \{0°, 1°, 2°, \ldots, 179°\}$ degrees.

Iterative tomographic reconstruction aims to solve an optimization problem. The following cost function is often used:

$$\hat{x} = \underset{x \in C}{\operatorname{argmin}} \{\|y - Ax\|_2^2 + R(x)\} \quad (1)$$

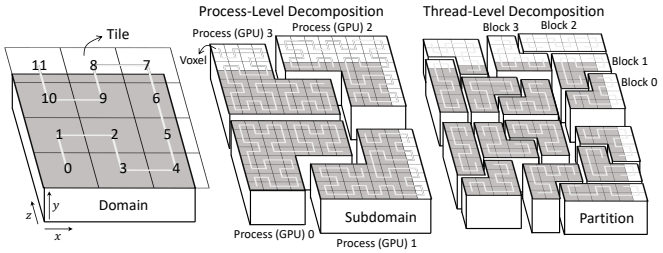
Here, y is the measurement data, x represents the estimated versions of the unknown object, C is a constraint on x and $R(x)$ is a regularizer function. A is the system matrix or forward operator that depicts how the X-ray beams intersect the voxels in each rotational view and thus the relationship between x and y, i.e. the sinograms and the object, respectively. The solution $\hat{x}$ is the version of the estimated object that minimizes the cost function.

A generic optimization solver requires computing the gradients and updating of the object based on those gradients in an iterative fashion. First, the *forward operator*, A, is applied



(a) (b)

Fig. 3: (a) 3D domain partitioning and (b) assignment on GPUs.



(a) (b) (c)

Fig. 4: 3D Hilbert-ordering domain decomposition at (a) tile (b) process, i.e., GPU, and (c) thread, i.e., block, levels.

to previously estimated x_i object, then the result is subtracted from sinograms, $y - Ax_i$, and residual r_i is computed based on a norm, e.g., Euclidean norm as in equation 1. Next, r_i is *back projected* on x_i to compute the gradients. Finally, the estimated object x_i is updated based on the computed gradients and the new object x_{i+1} derived for next iteration. In this work, we consider parallel beam geometry for experiments at the synchrotrons and that all beams travel perpendicularly to the axis of rotation, which enables independent reconstruction of slices in 3D object volume from their corresponding sinograms. We also perform an optimized version of Siddon's algorithm for accurate forward and back projection operators [9].

B. Challenges and Opportunities

The sparse system matrix A in Eq. 1 represents the incident voxels of each ray, and therefore its memory footprint can be large. Forward and backprojection operators perform irregular accesses to A and x_i while computing residuals and updating objects. These operations introduce significant overhead during the iterations. Further, if the A matrix cannot fit into available memory, incident rows and columns need to be computed repetitively before each forward and backprojection operation. Furthermore, if some A and x elements involved in an inner product calculation performed by a process are in the memory of another process, that missing information must be communicated.

Prior work MemXCT addresses communication and irregular data access overheads using multi-level Hilbert ordering to maximize the likelihood that all system matrix A elements involved in an inner product of the sparse matrix-vector multiplication are in the same partition [4]. It also avoids repetitive computation of A with memoization while partitioning large memory footprint to many processes. However, the MemXCT approach processes each sinogram independently of others, hence the optimizations are performed only within 2D slices.

In this work, we identify and exploit the following optimization opportunities in 3D:

- Assuming a ray, $u_{i,j}$, where i and j are the location of the ray in the y and x dimensions, respectively, all rays $u_{*,j}$ trace the same voxels in their respective slices for all rotational views. This property can be used to *fuse* rays along the y dimension so that the A elements can be reused when processing these rays.
- MemXCT approaches typically utilize single data type throughout their execution. This can introduce unnecessary overhead during data movement. Mixed precision data types can minimize data transfer volume while meeting precision requirements of computation.
- Direct communication between reconstruction processes can create redundant overhead. Reducing data between co-located processes can significantly improve data transfer efficiency.

III. Algorithm Design & Optimizations

In this section, we present five categories of optimizations that exploit the opportunities identified in Section II.

A. Partitioning and Parallelization

We propose a data partitioning and parallelism arrangement strategy for both input (sinogram) and output (tomogram) data cubes as depicted in Fig. 3(a). The proposed strategy combines two principal parallelism modalities: *batch parallelism* and *data parallelism*.

Batch parallelism takes advantage of the fact that there is no data dependency between the slices in the y direction. Therefore, tomogram and sinogram slices are partitioned equally among *batch processes*[1] and reconstructed in an embarrassingly-parallel fashion. However, because batch parallelism does not involve partitioning of data in the x and z dimensions, the per-process memory footprint of slices may not fit within the GPU memory of batch processes.

Data parallelism solves this problem by partitioning slices among *data processes* in the x-z plane along with their corresponding data structures. As a result, per-process memory footprint is reduced so that it can fit within GPU memory. Nevertheless, data parallelism comes with a communication overhead because of dependency between data partitions in the x and z dimensions. The detailed design of the proposed strategy consists of the following components to minimize communication and maximize data reuse.

1) Hilbert-Ordering Domain Decomposition: In order to partition a batch of slices in the x-z plane while maintaining data locality, we extend the MemXCT 2D Hilbert-ordering domain decomposition to 3D, where it is applied to all tomogram and sinogram slices in the y direction. Fig. 3 and Fig. 4 depict decomposition at the process and thread levels: First, both tomogram and sinogram domains are tiled into square patches and ordered with pseudo-Hilbert ordering. These patches are partitioned equally among processes, where each partition

[1] In the context of this paper, each process corresponds to a GPU.

corresponds to a *subdomain* as shown in Fig. 4(b). Each subdomain is processed by a single GPU.

Fig. 3(a) shows that a domain is partitioned into 12 subdomains: two in the x direction, two in the z direction, and three in the y direction. Fig. 3(b) depicts assignment of these 12 subdomains to a GPU grid with 12 processes. The GPU grid consists of six nodes and each node contains two GPUs. Since communications within node have a higher bandwidth than that of between nodes, data processes processing adjacent subdomains are placed in the same node whenever possible.

Then each subdomain is partitioned among GPU thread blocks as shown in Fig. 4(c). Data connectivity and locality of partitions provided by the Hilbert-ordering domain decomposition are essential for optimized SpMM design (Sec. III-B) and hierarchical communications (Sec. III-D) – two of the major contributions of this paper.

2) Batch Processing Pipeline: To overlap MPI communication and GPU computation, we partition each batch into smaller *I/O batches* that are processed sequentially, i.e., one I/O batch is reconstructed at a time. As discussed in Sec. III-B, optimzed SpMM fuses multiple slices in a I/O batch into *minibatches*. Processing of these minibatches are pipelined by overlapping MPI communications and GPU computations as explained in Sec. III-E. To achieve sufficient overlapping, there needs to be at least a few minibatches in an I/O batch.

3) Optimal Partitioning Strategy: In order to minimize the communication overhead of data parallelism, it is better to minimize partitioning of the 3D data cube in the x-z dimension; only until per-process memory footprint fits into GPU memory. Then batch partitioning should take over in the y dimension with no additional overhead. The level of batch parallelism is limited by the total number of slices in the y direction and the need to fuse slices so that the sparse matrix $\boldsymbol{A}$ can be reused from register by the optimized SpMM.

TABLE I: Computational Complexity

	Per Process	Total
Comput.	$MN^2/P_bP_d + MN/P_b\sqrt{P_d}$	$MN^2 + MN\sqrt{P_d}$
Memory	$N^2/P_d + N/\sqrt{P_d}$	$N^2P_b + NP_b\sqrt{P_d}$
Comm.*	$MN/P_b\sqrt{P_d}$	$MN\sqrt{P_d}$

M row channels, N column channels, P_b batch processes, P_d data processes.
*Latency and contention terms are omitted.

4) Computational Complexity: The computational cost of a projection depends on size of the 2D detector grid depicted in Fig. 2. Here, M and N are the number of row and column channels in the detector grid. Since each discretized ray measured by a detector propagates through $\mathcal{O}(N)$ voxels, each projection takes $\mathcal{O}(MN^2)$ time. Parallel rays have the same trajectory in all slices, and therefore it is sufficient to store a single sparse matrix with $\mathcal{O}(N^2)$ nonzeroes and reuse it from memory for all M slices. The total memory footprint is increased by a factor of batch processes (P_b) because of the sparse matrix duplication. On the other hand, data parallelization partitions the sparse matrix into P_d data processes, where each process computes a partial projection. The geometrical shapes of the partial data is shown in Fig. 7(b). The partial data of a processes scales with $MN/\sqrt{P_d}$ because the cross-section of each subdomain on the detector halves only when P_d is quadrupled. To obtain the total projection, the partial data is communicated among all data processes to be reduced at the receiving data processes. This additional communication and reduction is the overhead of data parallelism. Table I summarizes per-process and total computation, memory, and communication complexities of the proposed 3D partitioning.

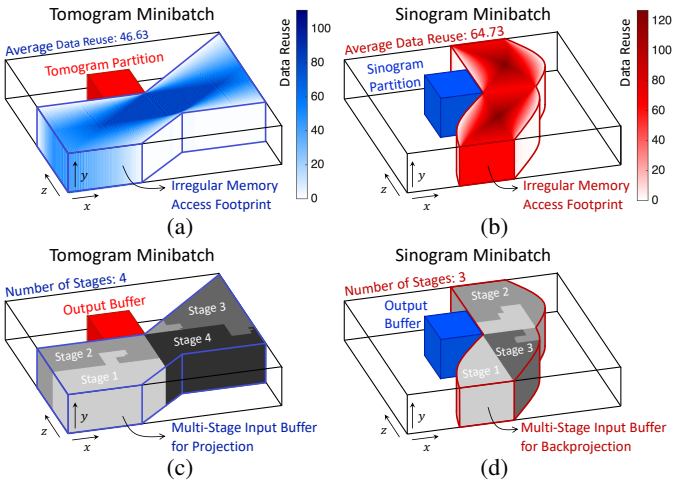


Fig. 5: (a, b) Tomogram and sinogram partitions and respective data access footprints, and (c, d) multi-stage buffer shapes.

B. XCT-Optimized SpMM Design

By considering memory access patterns specific to XCT, we optimize the proposed mixed-precision SpMM kernel via a 3D input/output buffering algorithm. This subsection discusses these optimizations and explains our CUDA implementation given in Listing 1. The same implementation is used for both projection and backprojection.

1) 3D Input Buffering: A naive SpMV implementation suffers not only from irregularity but also redundant accesses to tomogram and sinogram data because each voxel is (irregularly) accessed more than once by different threads. As a remedy, our optimized kernel stages data accesses in the GPU shared memory via 3D input buffers. Fig. 5 illustrates the basic idea of our optimization by depicting memory access footprints for a 256×256×50 minibatch; (a) shows tomogram voxels accessed by a sinogram partition and (b) shows sinogram voxels accessed by a tomogram partition, where each partition is computed by a single thread block. In 3D input buffering, each thread block loads the highlighted input data (Fig. 5: (a) and (b)) to shared memory once, and reuses it (irregularly) from shared memory. As a result, memory accesses are reduced by the factor of data reuse shown in the figure, i.e., darker shade represents higher data reuse from shared memory. With these reuses, the GPU performance becomes limited by the memory bandwidth consumed when reading $\boldsymbol{A}$ and $\boldsymbol{A}^T$ from memory (see the roofline analysis in Sec. IV-C1). We overcome this memory-bandwidth bottleneck by reusing $\boldsymbol{A}$ elements from registers as we discuss next.

2) Register Reuse: Even with input buffering, the optimized SpMV utilizes only less than two percent of the GPU's theoretical peak compute throughput because of its low arithmetic intensity (FLOPS/byte). This is because each fused multiply add (FMA) requires two data elements from memory: (1) shared-memory *index* of the visited voxel by X-ray and (2) intersection *length* of the ray going through the voxel. To increase the arithmetic intensity, we propose to reuse index and value data from registers. That is, many SpMVs in a minibatch are fused as $\boldsymbol{AX} = \boldsymbol{B}$, where each column of $\boldsymbol{X}$ and $\boldsymbol{B}$ corresponds to a slice of tomogram and sinogram values in the minibatch, respectively. As a result, the arithmetic intensity of the operation increases by the fusing factor, a.k.a., minibatch size. However, the fusing factor cannot be arbitrarily large because fusing imposes register pressure as discussed next.

3) 3D Output Buffering: To provide data reuse from register, each thread loads one index and one length at a time and then reuses this pair from register for all corresponding output voxels in the y direction of the minibatch. To accumulate the partial sums, each thread allocates an output buffer (`acc` in Line 10 of Listing 1). Line 28 shows how a thread reuses the index and value pairs to access its corresponding data in the 3D input buffer. However, since each streaming multiprocessor (SM) has a limited number of registers that are used by all threads, enlarging the minibatch size, i.e., `FFACTOR`, can increase the number of registers required by each thread and hence elevate register pressure on the SMs. When threads collectively use more registers than available in SM, register contents are spilled to slower memory and hamper GPU performance. Results show that GPUs are able to obtain 34% of their theoretical compute throughput by carefully tuning the minibatch size as shown in Sec. III-B.

Listing 1: Optimized SpMM Mixed-Precision Kernel

```
1  //MATRIX STRUCTURE
2  struct matrix{ unsigned short ind; half len; };
3  //PROJECTION KERNEL
4  __global__ void kernel_project(half *y, half *x, matrix *mat,
5                                 int *displ, int numrow, int numcol,
6                                 int *buffdispl, int *buffmap,
7                                 int *mapdispl, int *mapnz,
8                                 int buffsize ){
9    extern __shared__ half shared[];
10   float acc[FFACTOR] = 0.0;
11   int wind = threadIdx.x%WARPSIZE;
12   for(int buff = buffdispl[blockIdx.x]; buff <
13                                   buffdispl[blockIdx.x+1]; buff++){
14     int mapoffset = mapdispl[buff];
15     for(int i = threadIdx.x; i < mapnz[buff]; i += blockDim.x){
16       int ind = buffmap[mapoffset+i];
17       #pragma unroll
18       for(int f = 0; f < FFACTOR; f++)
19         shared[f*buffsize+i] = x[f*numcol+ind];
20     }
21     __syncthreads();
22     int warp = (buff*blockDim.x+threadIdx.x)/WARPSIZE;
23     for(int n = displ[warp]; n < displ[warp+1]; n++){
24       matrix mat = indval[n*WARPSIZE+wind];
25       float len = __half2float(mat.len);
26       #pragma unroll
27       for(int f = 0; f < FFACTOR; f++)
28         acc[f] += __half2float(shared[f*buffsize+mat.ind])*len;
29     }
30     __syncthreads();
31   }
32   int row = blockIdx.x*blockDim.x+threadIdx.x;
33   if(row < numrow)
34     #pragma unroll
35     for(int f = 0; f < FFACTOR; f++)
36       y[f*numrow+row] = __float2half(acc[f]);
37 }
```

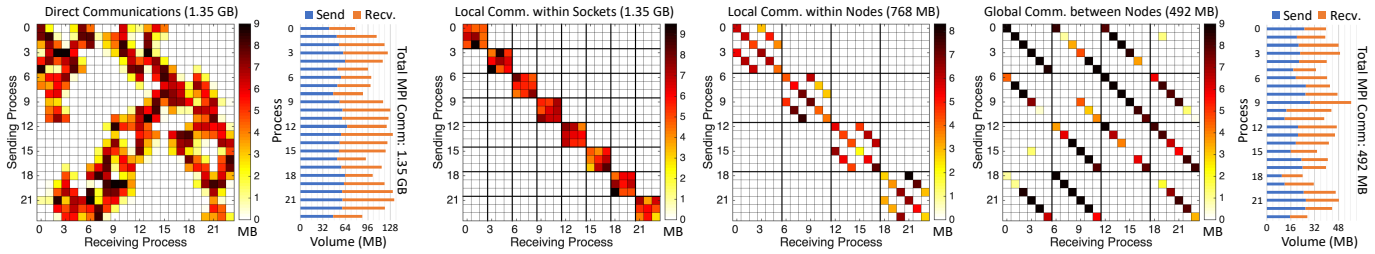
4) Multi-Stage 3D Buffering: Each SM has a limited capacity of shared memory (96 KB for V100 GPUs), which is not enough for large memory access footprint of a 3D partition. Therefore we provide access to input data in multiple stages. Fig. 5: (c) and (d) show four-stage and three-stage bufferings for projection and backprojection, respectively. Each stage loads 96 KB data whose shapes are governed by Hilbert ordering as described in Sec. III-A1. Line 12 in Listing 1 shows the iterations over stages, where only a portion of the 3D input buffer is loaded by mapping `buffmap`, and only a partial value of the 3D output buffer is computed in each stage. Because each staging has a synchronization overhead (seen in Lines 21 and 30), fewer number of stages is desirable. The 3D buffering strategy increases the number of stages since memory footprint of input buffers grow with increasing minibatch size (in the y direction). We mitigate increasing memory footprint using mixed precision implementation as explained in the following subsection.

C. Mixed-Precision Implementation

Mixed-precision implementation stores and communicates data in half precision (16-bit) and performs all arithmetic operations with single precision (32-bit) through in-core data conversions as in highlighted in Listing 1 (red). This approach reduces memory and communication footprint as well as provides a higher GPU throughput by moving data in half-precision while maintaining numerical accuracy by performing FMAs in single-precision. Several issues need to be addressed while using mixed precision implementation: (1) Half-precision has a lower range and quantization, and therefore it can suffer from overflow and underflow. (2) When reading sparse matrix $\boldsymbol{A}$ from memory, each warp of 32 threads accesses only 64 bytes of data at a time, which only utilizes half of the 128-byte GPU cache-line and therefore results in sub-optimal cache utilization. We address these issues with *adaptive normalization* and *data packing*, respectively.

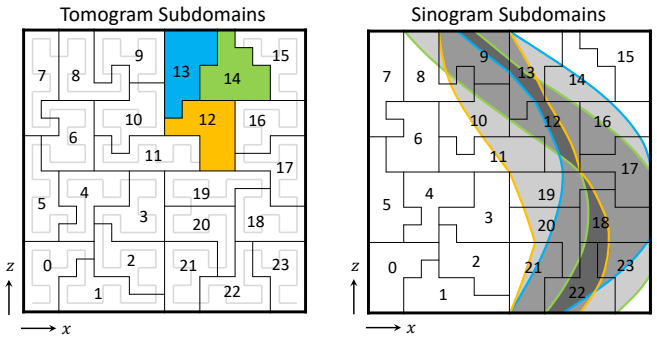
1) Adaptive Normalization: We avoid the half-precision underflows by artificially increasing the voxel size in the tomogram. Since the nonzero sparse matrix values represent the travel lengths of incident X-rays per voxel, increasing the voxel sizes results in larger half-precision values and hence avoids underflows. On the other hand, the overflows are handled by maintaining input and output data in double or single precision before performing kernel operations. We accomplish this by normalization and denormalization of input and output data before and after type castings, respectively. The (de)normalization factor is adaptively changed in each iteration with respect to the max-norm of the evolving input vector to prevent overflows while minimizing underflows.

2) Data Packing: The underutilization of GPU cache line not only wastes memory bandwidth but also introduces latency. As a remedy, we pack both index and length data elements into a single structure of four bytes. This structure is shown in Line 2 of Listing 1: It consists of an unsigned two-byte integer that represents the voxel index in shared-memory, and a two-byte half-precision floating point number representing the length.



(a) (b) (c) (d)

Fig. 6: (a) Direct communication and load balancing. Three-level hierarchical communications: (b) Socket-level communication; (c) Node-level communication; (d) Global communication and load balancing.



(a) (b)

Fig. 7: Tomogram (a) and sinogram (b) subdomains. Partial data footprint of tomogram subdomains 12–14 are shown in (b).

As a result, each warp of 32 threads utilizes the full 128-byte cache line.

D. Hierarchical Communications

Data must be partitioned when solving large problems in order to fit per-process memory footprint within available GPU memory. However, data partitioning requires communication and reduction of partial data, as elaborated in Sec. III-A4. Consequently, image reconstruction time is dominated by communication overhead for large problems. As a solution, we propose a novel hierarchical communication with partial data reduction, another major contribution of this paper. This strategy is designed for multi-GPU node architectures where high-bandwidth connections between *peer* GPUs are exploited. This section describes direct communication of partial data (the baseline), the main idea behind local communication and reduction of partial data, our communication hierarchy, and its efficient implementation.

1) Direct Communications: Fig. 7 shows partitioning of (a) tomogram data and (b) sinogram data into 24 subdomains. The subdomain shapes are governed by Hilbert-ordering domain decomposition (Sec. III-A1). In projection, each process computes only a partial sinogram data which needs to be communicated and reduced to find total sinogram data. As an example, Fig. 7(b) shows the partial data footprints of processes 12–14 on sinogram subdomains, e.g., process 12 sends its corresponding partial data to processes 8–13 and 18–23. The resulting communication matrix is shown in Fig. 6(a). Communication volume between two processes depends on the amount of overlapped area between sender's partial data footprint and receiver's sinogram subdomain. As a result, the communication is sparse and irregular. Following the communication, receiving processes *reduce* (sum) the overlapping partial data coming from sender processes, which completes projection operation. This description is also valid for backprojection as it is a transpose of projection.

Fig. 6(a) also shows communication volume of each process and total amount of communicated partial data. Large problems communicate a large portion of data through slow interconnect between nodes, which constitutes a dominating bottleneck. We relieve this bottleneck by reducing partial data locally within nodes and communicating the reduced data between nodes, as we shall discuss next.

2) Local Reduction of Partial Data: The proposed hierarchical communication exploits high-bandwidth connections between GPUs within nodes by a *local* communicator and reduction of the overlapping partial data among sender processes. To explain, we consider processes 12–14 (Fig. 7(a)) located in the same node. Overlapping portions of their partial data (Fig. 7(b)) are communicated and reduced within the node, rather than communicating the original partial data among nodes directly. Then, the reduced partial data within the node is sent to receivers by a *global* communication, where they are further reduced to find total sinogram values. The locality of subdomains provided by Hilbert ordering is essential in this hierarchical communication because spatially-local subdomains yield more overlapping data and more local partial reductions. Similarly, fatter nodes with more highly-connected GPUs yield more local reduction of partial data.

3) The Three-Level Hierarchy: This paper considers Summit's node architecture, where each node has two CPU *sockets*. Each socket is connected to three GPUs that are densely-connected with high-bandwidth interconnect. The CPU sockets within a node are connected with a slower data bus. Each node is connected to an even slower infiniband network. Although, we explain our hierarchical communication and reduction in the context of the Summit architecture, the method is general and applicable to other node architectures with different number of sockets and GPUs.

Since the data bus between sockets is slow, we do not perform local reduction directly among six GPUs within a node. Instead, we first perform a socket-level communication and reduction among three GPUs within a socket. Then, an

additional node-level communication and reduction among six GPUs within a node. Finally, a global communication and reduction among all GPUs between nodes is performed.

Fig. 6(b) presents the socket-level communication matrix as a block-diagonal structure because each GPU talks only to three GPUs (including itself) in the same socket. The socket-level reduction reduces the original 1.35 GB partial data down to 768 MB (43% reduction). Then, the reduced partial data is further reduced within nodes among six GPUs. Fig. 6(c) shows the intra-node local communication matrix. After the node-level communication, the partial data is further reduced to 492 MB (36% additional reduction). Finally, GPUs send reduced partial data only among nodes as shown in Fig. 6(d). As a result, the total data communicated among slowly-connected nodes is reduced by 64% compared to direct communication.

4) Efficient Implementation: To implement the proposed three-level hierarchical communication, we leverage `MPI_Comm_split` to define local communicators within sockets and nodes, and a global communicator among nodes, respectively. To prevent data from being staged through the CPU during local communications, we use CUDA inter-process communication (IPC) capability, i.e., communicating data directly from GPU to GPU bypassing CPU. Furthermore, local communications are overlapped using CUDA streams. Then local reductions are performed on GPUs. Global communications are implemented with MPI with CPU staging through pinned buffers. We employ non-blocking `MPI_Issend` and `MPI_Irecv` for overlapping global communications. The partial data is then moved back to the GPUs at the receiving processes for global reductions.

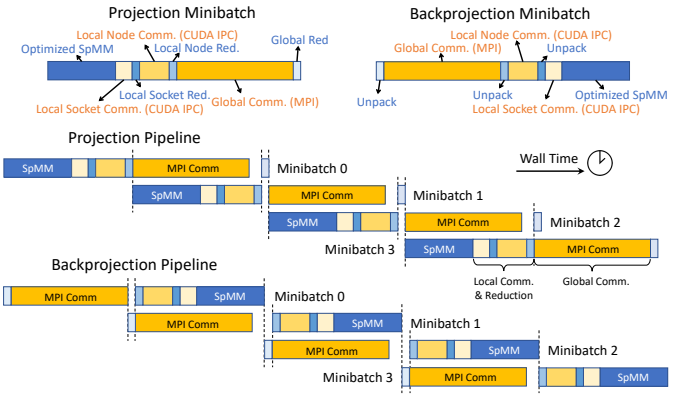


Fig. 8: Overlapping four minibatches in the batch processing pipeline.

E. Communication Overlapping

Even when communication time is reduced by applying hierarchical communication strategy, iterative solution time remains bounded by MPI communications for large problems (see Fig. 11). To further alleviate this communication bottleneck, we propose an overlapping strategy that exploits multiple slices in a reconstruction batch which is only possible in 3D image reconstruction. That is, global (MPI) communication of a minibatch is overlapped with local operations of another minibatch involving the optimized SpMM kernel, local reduction kernels, and local (socket-level and node-level) communications. Fig. 8 depicts overlapping of four minibatches during projection and backprojection operations. Projection overlaps global MPI communication of a minibatch with local operations of the next minibatch. On the other hand, backprojection overlaps local operations with next minibatch's MPI communication. This strategy is the most effective when kernel time and communication time are comparable.

IV. Experimental Results

This section introduces an extensive evaluation of our approaches. We first evaluate the performance of our optimizations and designs on Summit supercomputer using four real-world experimental datasets. Then, we investigate strong and weak scaling properties of our system. Finally, we compare convergence of our optimizations with four different levels of precision.

A. Experimental Setup

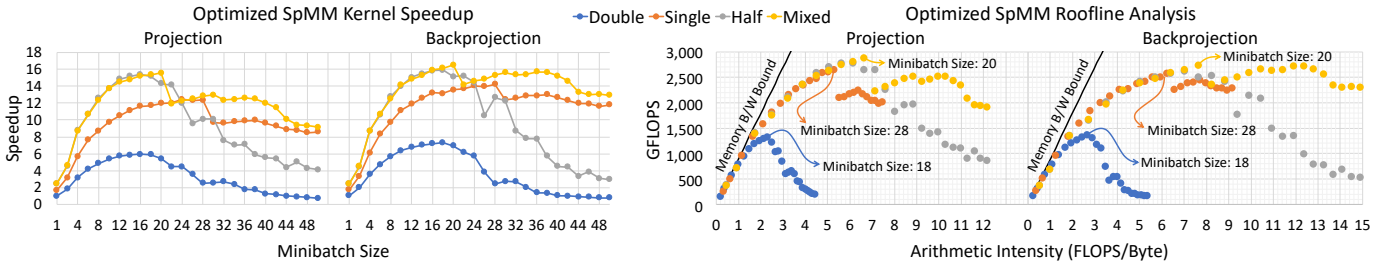
1) Summit Supercomputer: All experiments are conducted on Summit supercomputer at computing facility of ORNL. Summit consists of 4,608 IBM AC922 nodes with two POWER9 CPUs and six NVIDIA V100 GPUs per node. Nodes are connected via dual-raid fat-tree network. Each CPU in the node is densely-connected to three GPUs with Nvidia NVlinks, where each link has 50 GB/s one-way and 100 GB/s bidirectional bandwidth. We refer to each CPU and its corresponding densely-connected GPUs as a *socket* in this paper. Each node consists of two sockets. Sockets are connected via an X bus with 64 GB/s bidirectional theoretical bandwidth. Each CPU has 22 cores and 250 GB memory and each GPU has 80 SM and 16 GB memory.

TABLE II: Datasets and Memory Footprints

Sample	Measurement Data Cube ($K \times M \times N$)	I/O Data Footprint	Memory Footprint
Shale Rock	$1501 \times 1792 \times 2048$	52.1 GB	120 GB
IC Chip	$1210 \times 1024 \times 2448$	36.7 GB	139 GB
Activated Charcoal	$4500 \times 4198 \times 6613$	1.23 TB	2.82 TB
Mouse Brain	$4501 \times 9209 \times 11\,283$	6.56 TB	10.9 TB

2) Datasets Used for Experiments: Table II characterizes the four datasets used in our evaluation[2]. Dimensions are given in $K \times M \times N$, where K denotes the number of projections and M and N denote the number of vertical and horizontal channels in the 2D detector grid, respectively. All measurements follow parallel beam scan geometry as described in Sec. II. The corresponding I/O data and memory footprints are given for all datasets for single precision. All measurements are obtained by experiments at APS, ANL. The `Shale` and `Charcoal` datasets are open, while `Chip` and `Mouse` are proprietary [2], [10], [11]. Even though `IC Chip` and `Shale` have similar dimensions, we prefer to provide computational performance for the freely available `Shale` for benchmarking purposes. Nevertheless, we use `IC Chip` for the convergence

[2]In the rest of the paper, we refer to these datasets as `Shale`, `Chip`, `Charcoal`, and `Brain`, according to their order in Table II.



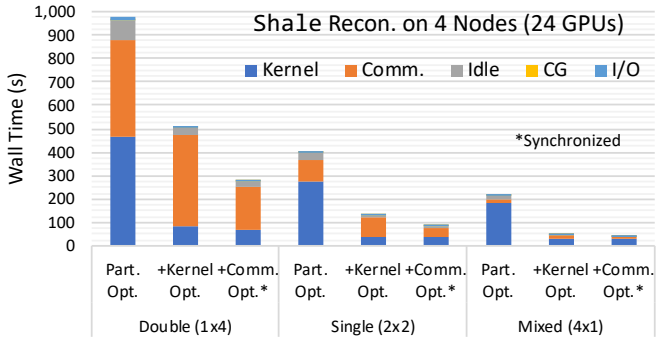
(a) (b)

Fig. 9: (a) Speedup and (b) per-GPU performance of XCT-Optimized SpMM kernel as a function of the slice fusing factor, i.e. minibatch size. Speedup is based on double-precision projection with no optimization (fusing factor 1).

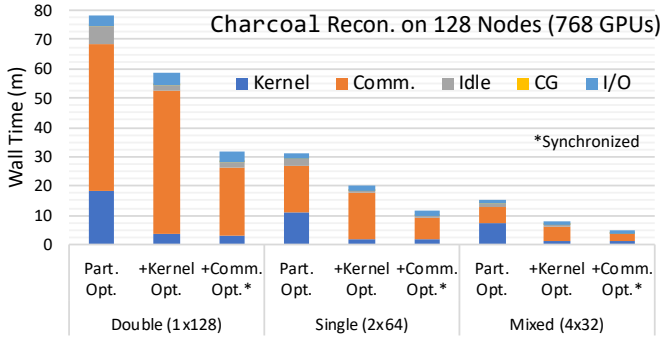
TABLE III: Overall Reconstruction Speedup

	Prec.	Shale on Four Nodes Part.*	Recon.	Speed.	Charcoal on 128 Nodes Part.*	Recon.	Speed.
Part. Opt.	Double	**1**×(**4**×6)	979 s	1×	**1**×(**128**×6)	78.4 m	1×
	Single	**2**×(**2**×6)	405 s	2.42×	**2**×(**64**×6)	31.3 m	2.51×
	Mixed	**4**×(**1**×6)	215 s	4.56×	**4**×(**32**×6)	15.1 m	5.20×
Part.+ **Kernel Opt.**	Double	**1**×(**4**×6)	513 s	1.91×	**1**×(**128**×6)	58.4 m	1.34×
	Single	**2**×(**2**×6)	134 s	7.30×	**2**×(**64**×6)	20.4 m	3.85×
	Mixed	**4**×(**1**×6)	51.1 s	19.2×	**4**×(**32**×6)	8.0 m	9.78×
Part.+ Kernel+ **Comm. Opt.**	Double	**1**×(**4**×6)	218 s	4.49×	**1**×(**128**×6)	27.0 m	3.00×
	Single	**2**×(**2**×6)	76.5 s	12.79×	**2**×(**64**×6)	10.0 m	7.87×
	Mixed	**4**×(**1**×6)	42.2 s	23.19×	**4**×(**32**×6)	4.30 m	18.19×

*Total Number of Partitions = **Batch Nodes** × (**Data Nodes** × Partitions per node). Data partitions per node is set to six because each node consists of six GPUs.



(a)



(b)

Fig. 10: Breakdown of end-to-end reconstruction times for (a) `Shale` and (b) `Charcoal`. *Overlapping does not applied (i.e., communications are synchronized) to measure each portion's time.

results (Sec. IV-F) because it is a numerically challenging case with contaminating noise.

B. Overall Reconstruction Performance

This subsection evaluates the overall reconstruction speedup with our optimizations. Table III reports end-to-end reconstruction time for `Shale` on four nodes and `Charcoal` on 128 nodes. These are the minimum number of nodes required to fit the corresponding memory footprints of double-precision reconstructions. The first three rows report the result without optimized SpMM, hierarchical communications, or communication overlapping. We only optimize the baseline implementation so that it uses the optimal combinations of batch and data parallelism according to the level of precision used for storing the data in memory. That is, lower precision representations shrink the memory footprint, allowing more batch parallelization and less data partitioning. As explained in Sec. III-A3, data partitioning comes with communication overhead and thus it is desirable to employ more batch parallelism and less data partitioning. In this example, as shown in Table III, we perform partitioning in node granularity, where each node handles six data partitions (one per GPU). The batch nodes partition slices in the 3D reconstruction domain (data duplication, no communication) and data nodes partition each slice (data partitioning, communication). The next three rows apply kernel optimizations via optimized SpMM implementation (Sec. III-B). The last three rows apply hierarchical communications (Sec. III-D) and overlapping (Sec. III-E). As seen in Table III, overall speedup over double-precision baseline is 23.19× and 18.19× for `Shale` and `Charcoal` reconstructions, respectively.

To further study the effect of each optimization, Fig. 10 shows the breakdown of the end-to-end reconstruction time. The communications are synchronized (i.e., not overlapped) in order to accurately measure the time taken by each activity. However, we do not synchronize GPU kernels and thus idle time reflects the corresponding load imbalance. These results show that optimized SpMM reduces kernel execution time significantly in all cases. The performance of the kernel is measured at 75 TFLOPS for `Shale` on four nodes (24 GPUs) and 2.38 PFLOPS for `Charcoal` on 128 nodes (768 GPUs). As shown in Fig. 10, execution time is dominated by communication for most of the cases. Hierarchical communications reduce the communication time by more than 50% in all cases. Sec. IV-C and Sec. IV-D further analyze the optimized SpMM and communication optimizations in detail.

C. Optimized SpMM Performance

Fig. 9(a) shows optimized SpMM speedup with varied minibatch sizes: Performance increases in all cases with larger minibatching due to the aforementioned register reuse

TABLE IV: Communicated Data* and Effective System Bandwidth

	Prec.	Socket-Level Comm. Data	Socket-Level Comm. B/W	Node-Level Comm. Data	Node-Level Comm. B/W	Global Comm. Data	Global Comm. B/W	Memcpy B/W
Direct	Double	N/A		N/A		36.6 TB	1.61 TB/s	35.2 TB/s
	Single					18.3 TB	1.61 TB/s	34.9 TB/s
	Mixed					9.16 TB	1.59 TB/s	34.6 TB/s
Hierar.	Double	36.6 TB	174 TB/s	21.4 TB	21.3 TB/s	15.2 TB	1.58 TB/s	34.9 TB/s
	Single	18.3 TB	170 TB/s	10.7 TB	22.8 TB/s	7.58 TB	1.55 TB/s	34.5 TB/s
	Mixed	9.16 TB	164 TB/s	5.35 TB	23.5 TB/s	3.79 TB	1.49 TB/s	33.6 TB/s

*Per projection (and backprojection). Fig. 11 involves 30 projections and 31 backprojections.

provided by our optimized SpMM. However, the kernel performance stagnates when minibatch size reaches around 16, and then it drops with larger minibatch sizes. This is due to a combination of register pressure and synchronization overhead introduced by large minibatch sizes as discussed in Sec. III-B. Nevertheless, minibatch sizes of 18, 28, 16, and 20 provide a maximum of 6.47×, 7.77×, 6.30×, and 6.58× kernel speedup compared to no minibatching with double, single, half, and mixed precision, respectively. Overall, mixed precision achieves the best performance of 15.66× speedup over the double precision baseline.

1) Roofline Analysis: To analyze the optimized SpMM performance further, Fig. 9(b) shows the roofline analysis plot, where the horizontal axis shows the arithmetic intensity, i.e., the number of floating-point operations per byte accessed from memory, and the vertical axis shows the GFLOPS performance per GPU. Each data point in the figure corresponds to one data point in Fig. 9(a), where increasing minibatch size increases the arithmetic intensity thanks to the data reuse from registers (Sec. III-B). Lowering precision naturally increases arithmetic intensity due to the less number of bytes per data element. Half and mixed precisions yield the same arithmetic intensity. The memory bandwidth bound (maximum performance possible) with respect to the theoretical 900 GB/s memory bandwidth of V100 GPU is included in Fig. 9(b). This figure shows that the performance with small arithmetic intensities is well-bounded by the memory bandwidth and starts to diverge when intensity increases. This is mainly due to the synchronization overhead of multi-stage input buffering.

Double and half precision performance start to degrade for minibatch sizes larger than 18 because of the register spilling as a consequence of the pressure incurred (see Sec. III-B3). On the other hand, single and mixed precisions do not experience register spilling up to a minibatch size of 50. This is mainly because register usage is well optimized for single precision unrolled FMAs (Line 26 of Listing 1). Nevertheless, performance drops significantly at minibatch sizes of 28 and 20 for single and mixed precisions, respectively. This sharp drop in performance may be due to a change in compiler optimization strategy to prevent register spilling by sacrificing performance under a high register pressure. However, `nvcc` is proprietary, preventing us from further investigation or verification. 16 is set as the minibatch size for all our experiments since the slices count is often a multiplication of 16.

2) Comparison with cuSPARSE: We compare our optimized SpMM performance with `cusparseSpMM`, which uses cuSPARSE library[3]. In order to find the best performance of both implementations, we aggressively try all minibatch sizes up to 50 and select the best ones. Our results show that our optimizations provide 1.53× to 2.38× speedup for double and single precision types, respectively. Unfortunately, `cusparseSpMM` is not able to converge to a solution with half precision type. Therefore we cannot make a direct comparison for other types. We are investigating the reason of this.

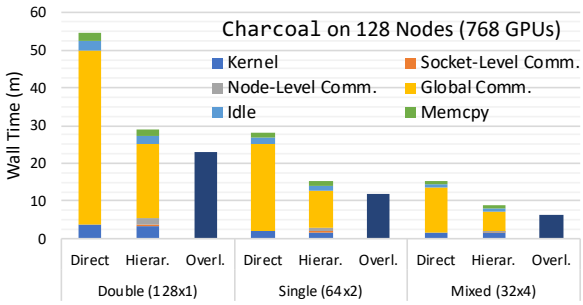


Fig. 11: Communication time breakdown, `Charcoal` on 128 nodes.

D. Communication Performance

To investigate communication optimizations further, Fig. 11 shows the breakdown of communication time for `Charcoal` reconstruction on 128 nodes. Communication time with direct and hierarchical communication strategies (Sec. III-D), and total time with communication overlapping (Sec. III-E) are investigated. Fig. 11 shows that hierarchical communication reduces the total communication time by 52%. Communication overlapping offers an additional speedup of 21% to 29% in total execution time. As opposed to the example in Fig. 8, `Charcoal` reconstruction has less overlapping opportunities because the global communication dominates the execution, bounding the overlapping efficiency.

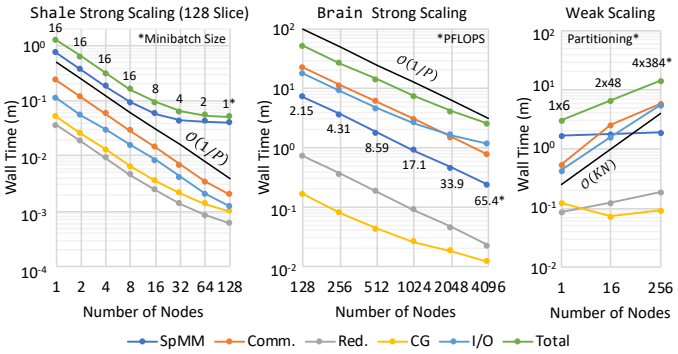
Table IV shows the amount of communicated data in TB and corresponding effective system bandwidth in different levels of the communication hierarchy. Communication data is naturally smaller in all cases with lower precision. Effective system bandwidth is calculated by dividing communication data amount by respective communication time shown in Fig. 11. Table IV shows that the effective bandwidth within each socket is about 100× faster than that among nodes. Similarly, the effective bandwidth among sockets is 15× faster than that among nodes. This high bandwidth within nodes alleviates the local communication overhead for hierarchical communications as seen in Fig. 11. Overall, hierarchical communication reduces the communication among nodes by 58% for all precision types.

E. Scaling Performance

To investigate the scaling properties of the proposed application, we perform two strong and one weak scaling experiments with all optimizations applied. Communication overlapping is disabled to correctly measure individual timings. All experiments perform 30 CG iterations with mixed precision.

[3]https://developer.nvidia.com/cusparse

1) Strong Scaling: First, we scale 128 slices of `Shale` up to 128 nodes. Fig. 12(a) shows the scaling results. This experiment demonstrates the limited scalability when there is a small number of slices. This experiment scales up to only 128 nodes where each node reconstructs only one slice. In the base case, there are eight minibatches (each contains 16 slices) so the solution can scale efficiently only up to eight nodes (see III-A3). We need to reduce the size of the minibatches after eight nodes to achieve more parallelism by working on finer-granularity. However, SpMM performance drops due to reduced data reuse from registers. As a result, we scale up to 128 nodes (one slice per node) but the performance is suboptimal due to the bottleneck of reduced SpMM performance as shown in Fig. 12(a).



(a) (b) (c)

Fig. 12: Strong and weak scaling results: Shale and Brain datasets.

Fig. 12(b) shows the strong scaling result of `Brain` reconstruction from 128 nodes (the minimum number of nodes that `Brain` fits) to 4096 nodes (89% of Summit). As opposed to the previous experiment, there are 9209 slices in `Brain` (see Table II) that allow us to scale without compromising from SpMM performance. As a result, the total reconstruction time is in agreement with $\mathcal{O}(1/P)$ curve, where P is the number of nodes. The I/O performance starts to degrade at 1024 nodes because of the contention that parallel I/O puts into the file system. Nevertheless, thanks to our proposed optimizations, **we reconstruct 3D `Brain` dataset in 2.5 minutes on Summit.** To the best of our knowledge, this is the largest XCT image reconstruction in near-real time. The optimized SpMM kernel performance reaches 65.4 PFLOPS on 24 576 GPUs (six GPUs per node): 34% of Summit's theoretical peak performance.

2) Weak Scaling: To investigate the weak scaling properties, we take `Shale` as the base dataset, and then synthetically double each dimension in the aforementioned measurement data cube ($K \times M \times N$). Each time we double all dimensions, and the nominal computation (excluding the parallelization overhead) increases by 16× (see Table I). Accordingly, we increase the number of nodes 16× each time we double measurement dimensions. As discussed in Table I, the memory footprint increases by 8× at each step of the scaling. We apply the suggested optimal partitioning strategy in Sec. III-A3 by partitioning data structures among eight nodes and slices between two nodes. Fig. 12(c) shows the weak scaling results. Since nominal computation per node remains the same, SpMM time remains constant accordingly. However, communication and I/O time increases and becomes the bottleneck for large problems. I/O could be further optimized either by a custom design or a high-performance library, which is beyond the scope of this work.

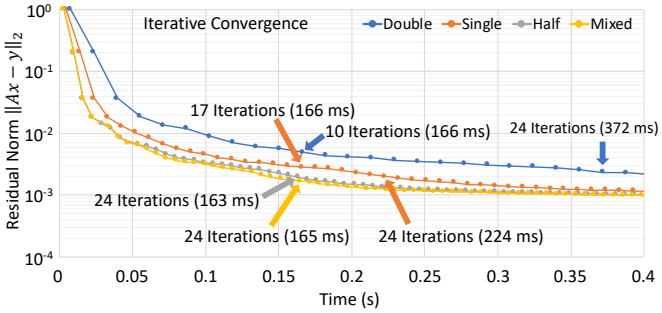


Fig. 13: Convergence speed for `Chip` with various precisions.

F. Convergence Performance

To investigate iterative convergence rates, we reconstruct a slice from `Chip` dataset. Since this dataset is noisy, iterative solution experiences noise *overfitting* after 24 iterations and even though the residual norm shrinks further, measurement noise manifests into the image reconstruction. To prevent that, we terminate the iterative solution after 24 iterations. No serious convergence problem is observed with reduced precisions. This is mainly because the numerical noise floor is well below the measurement noise level—this applies to other datasets as well. Fig. 13 shows the (relative) residual norm for a single slice with respect to the execution time with double, single, half, and mixed precisions. Mixed (and half) precision performs 24 iterations in 165 ms, while single- and double-precision implementations complete the same number of iterations with 224 and 372 ms, respectively.

V. Related Work

Tomographic reconstruction has been researched extensively over the years. Iterative reconstruction algorithms usually show better image quality compared to analytical approaches [12]–[16] and have been used to accommodate limitations on experimental dataset, e.g. noisy measurements, missing angles and others [17]–[19].

Many parallelization techniques have been developed to ease the computational requirements of iterative algorithms, including distributed computing methods using multi-core [20]–[23]. However, they found limited applicability for large-scale dataset due to the computational requirements.

Many-core architectures, such as GPUs, have been widely used for reconstruction small to medium dataset [24]–[26] and received significant attention in recent years [4], [5], [27]–[29]. Medical imaging is one of the areas that extensively utilizes advanced tomographic reconstruction techniques. Since limiting dose exposure to patients is crucial, iterative reconstruction approaches are widely used to accommodate noisy

measurements [30]–[32]. Many-core systems can deliver the computational throughput required by the reconstruction tasks, however their limited memory and (host-device) communication cost can introduce significant overhead [33]. In contrast, our solution provides mixed precision computations and multi-level reduction techniques to ease communication cost.

Synchrotron radiation facilities can generate small to extreme-scale data using variety of imaging modalities, including tomography [34]–[36]. High performance software infrastructures have been built to handle tomographic reconstruction workflows that require large-scale compute resources [37]–[42]. The end-to-end performance of these systems heavily relies on underlying reconstruction engines and can directly benefit from the optimizations proposed in our work.

VI. Conclusion

In this work, we introduce novel optimization techniques for MemXCT approach that exploit 3D volume properties and multi-GPU node architecture during (back)projection operators. Specifically, our simultaneous data and batch partitioning scheme provides configurable volume distribution among processes and enables reconstruction of extremely large tomography data; XCT-Optimized SpMM design considers the sparse matrix structure and efficiently provides GPU utilization by reusing data from shared-memory and registers; hierarchical communications performs extra intra-node communications to minimize the inter-node communication; and finally, effective use of mixed-precision types further reduces memory footprint and communication volume while maintaining the reconstruction quality. The algorithm design and optimizations described in this paper enable an 11k×11k×9k 3D brain image reconstruction under three minutes using 24,576 GPUs on Summit supercomputer, reaching 65 PFLOPS: 34% of Summits peak performance.

Acknowledgments

This material was partially supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research and Basic Energy Sciences, under Contract DE-AC02-06CH11357. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. We thank Narayanan (Bobby) Kasthuri from UChicago/Argonne, and Rafael Vescovi and Ming Du from Argonne for sharing the mouse brain dataset. The mouse brain, IC chip, and activated charcoal data were collected by Vincent De Andrade at beamline 32-ID, Advanced Photon Source, Argonne National Laboratory. This work is supported by IBM-ILLINOIS Center for Cognitive Computing Systems Research (C3SR). This research is based in part upon work supported by the Center for Applications Driving Architectures (ADA), a JUMP Center co-sponsored by SRC and DARPA. This material is supported by funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skodowska-Curie grant agreement H2020-MSCA-COFUND-2016-754433.

References

[1] "APS Science 2018," Tech. Rep. ANL-18/40, ISSN 1931-5007, Advanced Photon Source, Argonne National Laboratory, January 2019.

[2] R. Vescovi, M. Du, V. de Andrade, W. Scullin, D. Gürsoy, and C. Jacobsen, "*Tomosaic*: efficient acquisition and reconstruction of teravoxel tomography data using limited-size synchrotron X-ray beams," *Journal of Synchrotron Radiation*, vol. 25, pp. 1478–1489, Sep 2018.

[3] T. Bicer, D. Gursoy, R. Kettimuthu, F. De Carlo, G. Agrawal, and I. T. Foster, "Rapid tomographic image reconstruction via large-scale parallelization," in *Euro-Par 2015: Parallel Processing*, pp. 289–302, Springer, 2015.

[4] M. Hidayetoğlu, T. Biçer, S. G. De Gonzalo, B. Ren, D. Gürsoy, R. Kettimuthu, I. T. Foster, and W.-m. W. Hwu, "Memxct: Memory-centric x-ray ct reconstruction with massive parallelization," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–56, 2019.

[5] X. Wang, V. Sridhar, Z. Ronaghi, R. Thomas, J. Deslippe, D. Parkinson, G. T. Buzzard, S. P. Midkiff, C. A. Bouman, and S. K. Warfield, "Consensus equilibrium framework for super-resolution and extreme-scale ct reconstruction," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–23, 2019.

[6] A. Beer, "Bestimmung der absorption des rothen lichts in farbigen flussigkeiten," *Ann. Physik*, vol. 162, pp. 78–88, 1852.

[7] J.-H. Lambert, *JH Lambert,... Photometria, sive de Mensura et gradibus luminis, colorum et umbrae*. sumptibus viduae E. Klett, 1760.

[8] R. A. Crowther, D. DeRosier, and A. Klug, "The reconstruction of a three-dimensional structure from projections and its application to electron microscopy," *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, vol. 317, no. 1530, pp. 319–340, 1970.

[9] R. L. Siddon, "Fast calculation of the exact radiological path for a three-dimensional ct array," *Medical Physics*, vol. 12, no. 2, pp. 252–255, 1985.

[10] F. De Carlo, D. Gürsoy, D. J. Ching, K. J. Batenburg, W. Ludwig, L. Mancini, F. Marone, R. Mokso, D. M. Pelt, J. Sijbers, *et al.*, "Tomobank: a tomographic data repository for computational x-ray science," *Measurement Science and Technology*, vol. 29, no. 3, p. 034004, 2018.

[11] M. Du, R. Vescovi, K. Fezzaa, C. Jacobsen, and D. Gürsoy, "X-ray tomography of extended objects: a comparison of data acquisition approaches," *JOSA A*, vol. 35, no. 11, pp. 1871–1879, 2018.

[12] T. Bicer, D. Gürsoy, V. D. Andrade, R. Kettimuthu, W. Scullin, F. D. Carlo, and I. T. Foster, "Trace: A high-throughput tomographic reconstruction engine for large-scale datasets," *Advanced Structural and Chemical Imaging*, vol. 3, p. 6, Jan 2017.

[13] K. Mohan, S. Venkatakrishnan, J. Gibbs, E. Gulsoy, X. Xiao, M. De Graef, P. Voorhees, and C. Bouman, "Timbir: A method for time-space reconstruction from interlaced views," *Computational Imaging, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

[14] S. V. Venkatakrishnan, C. A. Bouman, and B. Wohlberg, "Plug-and-play priors for model based reconstruction," in *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*, pp. 945–948, IEEE, 2013.

[15] P. P. Bruyant, "Analytic and iterative reconstruction algorithms in spect," *Journal of Nuclear Medicine*, vol. 43, no. 10, pp. 1343–1358, 2002.

[16] J. A. Fessler, M. Sonka, and J. M. Fitzpatrick, "Statistical image reconstruction methods for transmission tomography," *Handbook of medical imaging*, vol. 2, pp. 1–70, 2000.

[17] S. Aslan, V. Nikitin, D. J. Ching, T. Bicer, S. Leyffer, and D. Gürsoy, "Joint ptycho-tomography reconstruction through alternating direction method of multipliers," *Optics express*, vol. 27, no. 6, pp. 9128–9143, 2019.

[18] V. Nikitin, S. Aslan, Y. Yao, T. Biçer, S. Leyffer, R. Mokso, and D. Gürsoy, "Photon-limited ptychography of 3d objects via bayesian reconstruction," *OSA Continuum*, vol. 2, no. 10, pp. 2948–2968, 2019.

[19] D. J. Ching, M. Hidayetoğlu, T. Biçer, and D. Gürsoy, "Rotation-as-fast-axis scanning-probe x-ray tomography: the importance of angular diversity for fly-scan modes," *Applied optics*, vol. 57, no. 30, pp. 8780–8789, 2018.

[20] J. Treibig, G. Hager, H. G. Hofmann, J. Hornegger, and G. Wellein, "Pushing the limits for medical image reconstruction on recent standard multicore processors," *International Journal of High Performance Computing Applications*, 2012.

[21] J. Agulleiro and J.-J. Fernandez, "Fast tomographic reconstruction on multicore computers," *Bioinformatics*, vol. 27, no. 4, pp. 582–583, 2011.

[22] M. Jones, R. Yao, and C. Bhole, "Hybrid MPI-OpenMP programming for parallel OSEM PET reconstruction," *Nuclear Science, IEEE Transactions on*, vol. 53, pp. 2752–2758, Oct. 2006.

[23] C. Johnson and A. Sofer, "A data-parallel algorithm for iterative tomographic image reconstruction," in *7th Symposium on the Frontiers of Massively Parallel Computation*, pp. 126–137, Feb 1999.

[24] A. Sabne, X. Wang, S. J. Kisner, C. A. Bouman, A. Raghunathan, and S. P. Midkiff, "Model-based iterative CT image reconstruction on GPUs," in *22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 207–220, ACM, 2017.

[25] W. van Aarle, W. J. Palenstijn, J. De Beenhouwer, T. Altantzis, S. Bals, K. J. Batenburg, and J. Sijbers, "The ASTRA Toolbox: A platform for advanced algorithm development in electron tomography," *Ultramicroscopy*, vol. 157, pp. 35–47, 2015.

[26] X. Li, Y. Liang, W. Zhang, T. Liu, H. Li, G. Luo, and M. Jiang, "cuMBIR: An efficient framework for low-dose x-ray CT image reconstruction on GPUs," in *International Conference on Supercomputing*, pp. 184–194, ACM, 2018.

[27] X. Yu, H. Wang, W.-c. Feng, H. Gong, and G. Cao, "Gpu-based iterative medical ct image reconstructions," *Journal of Signal Processing Systems*, vol. 91, no. 3-4, pp. 321–338, 2019.

[28] P. Chen, M. Wahib, S. Takizawa, R. Takano, and S. Matsuoka, "ifdk: a scalable framework for instant high-resolution image reconstruction," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–24, 2019.

[29] M. Hidayetoglu, C. Pearson, I. El Hajj, L. Gurel, W. C. Chew, and W. Hwu, "A fast and massively-parallel inverse solver for multiple-scattering tomographic image reconstruction," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 64–74, 2018.

[30] D. Lee, I. Dinov, B. Dong, B. Gutman, I. Yanovsky, and A. W. Toga, "CUDA optimization strategies for compute-and memory-bound neuroimaging algorithms," *Computer Methods and Programs in Biomedicine*, vol. 106, no. 3, pp. 175–187, 2012.

[31] C.-Y. Chou, Y.-Y. Chuo, Y. Hung, and W. Wang, "A fast forward projection using multithreads for multirays on GPUs in medical image reconstruction," *Medical Physics*, vol. 38, no. 7, pp. 4052–4065, 2011.

[32] B. Jang, D. Kaeli, S. Do, and H. Pien, "Multi GPU implementation of iterative tomographic reconstruction algorithms," in *Biomedical Imaging: From Nano to Macro, 2009. ISBI'09. IEEE International Symposium on*, pp. 185–188, IEEE, 2009.

[33] T. B. Jablin, P. Prabhu, J. A. Jablin, N. P. Johnson, S. R. Beard, and D. I. August, "Automatic CPU-GPU communication management and optimization," in *ACM SIGPLAN Notices*, vol. 46, pp. 142–151, ACM, 2011.

[34] D. Gürsoy, T. Biçer, A. Lanzirotti, M. G. Newville, and F. De Carlo, "Hyperspectral image reconstruction for x-ray fluorescence tomography," *Optics express*, vol. 23, no. 7, pp. 9014–9023, 2015.

[35] D. J. Duke, A. B. Swantek, N. M. Sovis, F. Z. Tilocco, C. F. Powell, A. L. Kastengren, D. Gürsoy, and T. Biçer, "Time-resolved x-ray tomography of gasoline direct injection sprays," *SAE International Journal of Engines*, vol. 9, no. 1, pp. 143–153, 2016.

[36] D. Gürsoy, T. Biçer, J. D. Almer, R. Kettimuthu, S. R. Stock, and F. De Carlo, "Maximum a posteriori estimation of crystallographic phases in x-ray diffraction tomography," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 373, no. 2043, p. 20140392, 2015.

[37] T. Bicer, D. Gursoy, R. Kettimuthu, I. T. Foster, B. Ren, V. De Andrede, and F. De Carlo, "Real-time data analysis and autonomous steering of synchrotron light source experiments," in *2017 IEEE 13th International Conference on e-Science (e-Science)*, pp. 59–68, IEEE, 2017.

[38] R. J. Pandolfi, D. B. Allan, E. Arenholz, L. Barroso-Luque, S. I. Campbell, T. A. Caswell, A. Blair, F. De Carlo, S. Fackler, A. P. Fournier, *et al.*, "Xi-cam: a versatile interface for data visualization and analysis," *Journal of synchrotron radiation*, vol. 25, no. 4, pp. 1261–1270, 2018.

[39] T. Bicer, D. Gürsoy, R. Kettimuthu, F. De Carlo, and I. T. Foster, "Optimization of tomographic reconstruction workflows on geographically distributed resources," *Journal of synchrotron radiation*, vol. 23, no. 4, pp. 997–1005, 2016.

[40] D. Morozov and T. Peterka, "Block-parallel data analysis with diy2," in *2016 IEEE 6th Symposium on Large Data Analysis and Visualization (LDAV)*, pp. 29–36, 2016.

[41] O. Yildiz, J. Ejarque, H. Chan, S. Sankaranarayanan, R. M. Badia, and T. Peterka, "Heterogeneous hierarchical workflow composition," *Computing in Science Engineering*, vol. 21, no. 4, pp. 76–86, 2019.

[42] Z. Liu, T. Bicer, R. Kettimuthu, and I. Foster, "Deep learning accelerated light source experiments," in *2019 IEEE/ACM Third Workshop on Deep Learning on Supercomputers (DLS)*, pp. 20–28, IEEE, 2019.

MemXCT: Design, Optimization, Scaling, and Reproducibility of X-Ray Tomography Imaging

Mert Hidayetoğlu, Tekin Biçer, *Senior Member, IEEE*, Simon Garcia de Gonzalo, Bin Ren, Doğa Gürsoy, Rajkumar Kettimuthu, *Senior Member, IEEE*, Ian T. Foster, *Fellow, IEEE*, and Wen-Mei W. Hwu, *Fellow, IEEE*

Abstract—This work extends our previous research entitled "MemXCT: Memory-centric X-ray CT Reconstruction with Massive Parallelization" that was originally published at SC19 conference (Hidayetoğlu *et al.*, 2019) with reproducibility of the computational imaging performance. X-ray computed tomography (XCT) is regularly used at synchrotron light sources to study the internal morphology of materials at high resolution. However, experimental constraints, such as radiation sensitivity, can result in noisy or undersampled measurements. Further, depending on the resolution, sample size and data acquisition rates, the resulting noisy dataset can be in the order of terabytes. Advanced iterative reconstruction techniques can produce high-quality images from noisy measurements, but their computational requirements have made their use an exception rather than the rule. We propose a novel memory-centric approach that avoids redundant computations at the expense of additional memory complexity. We develop a memory-centric iterative reconstruction system, MemXCT, that uses an optimized SpMV implementation with two-level pseudo-Hilbert ordering and multi-stage input buffering. We evaluate MemXCT on various supercomputer architectures involving KNL and GPU. MemXCT can reconstruct a large (11K $\times$ 11K) mouse brain tomogram in 10 seconds using 4096 KNL nodes (256K cores). The results presented in our original article at the SC19 were based on large-scale supercomputing resources. The MemXCT application was selected for the Student Cluster Competition (SCC) Reproducibility Challenge and evaluated on a variety of cloud computing resources by universities around the world in the SC20 conference. We summarize the results of the top-ranked SCC Reproducibility Challenge teams and identify the most pertinent measures for ensuring the reproducibility of our experiments in this article.

Index Terms—X-ray tomography, space-filling curves, cache utilization, knights landing, GPU computing, SpMV, reproducibility

1 Introduction

X-ray computed tomography (XCT) is a widely used non-destructive 3D imaging technique for observing and understanding the internal morphology of samples. Synchrotron light sources, such as the Advanced Photon Source (APS), can provide high-brilliance X-rays that enable tomographic imaging of centimeter sized samples at sub-micrometer (μm) spatial resolution. Such experiments can generate from a few GBs to TBs of data volumes in a short time period with the typical pixelated detectors that can run at 16 GB/s [2]. However, the quality of data collected from CT experiments depends heavily on factors such as radiation exposure time (dose) and target spatial resolution. Much effort has been devoted to develop and implement advanced reconstruction algorithms to improve image quality when collected data are noisy or imperfect (e.g., due to limited dose).

Modern XCT reconstruction approaches involve either (i) direct solvers based on analytical inversion or (ii) iterative solvers that can accurately model the experimental conditions and/or constrain the solution based on prior knowledge of the sample. Analytical methods such as the filtered backprojection (FBP) algorithm are computationally efficient, but reconstruction quality is often poor when measurements are noisy or undersampled. Iterative methods, on the other hand, can use advanced optimization and regularization techniques to handle inherent noise in X-ray measurements [4], [5], [6], [7], [8], [9], [10]. However, these methods are computationally more demanding than the analytical methods, since forward and backprojection operations require many (irregular) accesses and computation at each iteration. Thus, efficient implementation and parallelization are crucial for high-quality large-scale image reconstructions.

Most state-of-the-art reconstruction software and libraries can perform parallel reconstruction to some extent. However, these implementations mostly rely on data replication [11], [12] and/or redundant computations [13], [14], each of which significantly limits the runtime performance of the system due to repeated computation of intermediate indices of ray tracing and irregular data accesses.

MemXCT, our advanced reconstruction system, uses a memory-centric approach to avoid redundant computation by removing on-the-fly operations at the expense of greater memory complexity. Specifically, it utilizes an optimized

- *Mert Hidayetoğlu, Simon Garcia de Gonzalo, and Wen-Mei W. Hwu are with the University of Illinois at Urbana-Champaign, Champaign, IL 61810 USA. E-mail: {hidayet2, grcdgnz2, w-hwu}@illinois.edu.*
- *Tekin Biçer, Doğa Gürsoy, Rajkumar Kettimuthu, and Ian T. Foster are with the Argonne National Laboratory, Lemont, IL 60439 USA. E-mail: {tbicer, dgursoy, kettimuthu, foster}@anl.gov.*
- *Bin Ren is with the College of Willam & Mary, Williamsburg, VA 23185 USA. E-mail: bren@cs.wm.edu.*

Manuscript received 27 Aug. 2021; revised 26 Oct. 2021; accepted 8 Nov. 2021. Date of publication 23 Nov. 2021; date of current version 24 Jan. 2022.
This work was partially supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research and Basic Energy Sciences, under Grant DE-AC02-06CH11357.
(Corresponding author: Mert Hidayetoğlu.)
Recommended for acceptance by S. L. Harrell and S. A. Michael.
Digital Object Identifier no. 10.1109/TPDS.2021.3128032

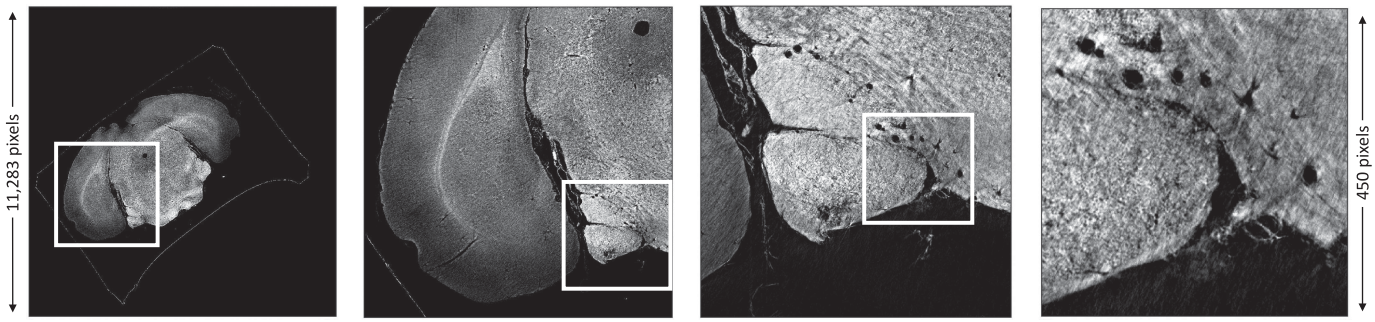


Fig. 1. Multiple zooms on a single 11293×11293 2D slice of a mouse brain image generated by reconstruction, with the methods described here, from a single sinogram with size 4501×11293. Data were collected by Dyer *et al.* [3] at the APS. Single-slice reconstruction with 30 CG iterations on 4096 KNL nodes takes 10 seconds. Memory footprint of the application is 10.2 TB. The full mouse brain consists of 11293 slices.

sparse matrix-vector multiplication (SpMV) implementation with *multi-stage buffering* and *two-level pseudo-Hilbert ordering* to optimize data communication, partitioning, and accesses at different analysis levels and on both measurement (sinogram) and reconstructed (tomogram) data.

MemXCT memoizes on-the-fly operations and therefore requires more memory than alternatives. However, its overall per-node memory footprint decreases linearly with increasing computing resources, promoting scalability with massive parallelization and enabling iterative reconstruction of extremely large datasets with efficient resource use. Further, MemXCT stores intermediate data structures in compressed format and locality-enhancing layout order to minimize memory footprint and data movement.

To illustrate the practical implications of MemXCT optimizations, we show in Fig. 1a 2D tomogram from a 3D mouse brain image with total size $11293 \times 11293 \times 11293$, obtained by reconstructing a sinogram with 4501×11293 dimensions. This single-slice reconstruction was generated with 30 conjugate gradient (CG) iterations in about 10 seconds on 4096 KNL nodes (256K cores) using MemXCT. This high speed capability makes it feasible, for the first time, to apply advanced iterative reconstruction algorithms to extremely large CT datasets. The multi-scale nature of the reconstruction is presented by zooming progressively to the brain arteries, where great detail can be seen. High reconstruction quality at such scales is important as subsequent data analysis steps, such as segmentation of blood vessels and myelinated axon tracts, are highly dependent on the quality of the reconstruction.

The MemXCT work was originally published at the SC19 conference[1]. In 2020, the MemXCT was selected as the benchmark application for the Student Cluster Competition (SCC) Reproducibility Challenge and evaluated on a variety of cloud computing resources by 16 university teams. Students were tasked to reconstruct several images using MemXCT and compared their measured performance results with the evaluations reported in the SC19 paper as well as in this paper. We summarize the SCC Reproducibility Challenge results of the top-ranked SCC teams and present key insights from their results at the end of the paper.

The principal contributions of this paper are as follows:

- We propose a memory-centric approach that uses an *extended SpMV* to address race conditions due to scatter operations. Our implementation replaces scatter operations on sinogram and tomogram domains with gather operations.
- We implement a *two-level pseudo-Hilbert ordering* to organize communications and data accesses, further improving communication performance, utilization of different levels of memory hierarchy, and cache utilization.
- We introduce a *multi-stage input buffering*. Domain partitioning improves process-level data communication and locality; multi-stage buffering enables data reuse in first-level caches and minimizes memory latency.
- We extensively evaluate *MemXCT* performance on both KNL and GPU systems up to 4096 nodes, and show architecture-specific considerations and optimizations.
- We present the key observations from student teams who successfully reproduced the SC19 paper results and identify the key measures in the MemXCT repository that are the most helpful to their efforts.

2 Background and Motivation

We first explain XCT experiments and their data acquisition process. Then, we describe the steps involved in iterative image reconstruction. Finally, we explain computational bottlenecks in iterative reconstruction approaches and introduce our solutions.

2.1 Data Acquisition and Measurement Process

In an XCT experiment, the sample is placed on a rotation stage and illuminated by an X-ray source, while collecting 2D images through a detector as the sample is being rotated: see Fig. 2. The mathematical model of the measurement process is based on Beer's law [15] that describes interaction between X-rays and matter: $I_\theta(s) = I_0(s)\exp[-p_\theta(s)]$, where $I_0(s)$ is the incident X-ray illumination on the sample and $I_\theta(s)$ are the collected measurements at a number of different θ angles as a result of a tomographic scan.

In Fig. 2, a set of collected projections from a sample is shown with p_{θ^i}. Considering parallel beam geometry, the sinogram $p_\theta(s)$ is a cross section of projections (shown in blue grid on p_{θ^*}) that consists of I_θ measurements from f. The goal of a tomographic reconstruction algorithm is to recover 2D image slice (tomogram) from its corresponding sinogram $p_\theta(s)$.

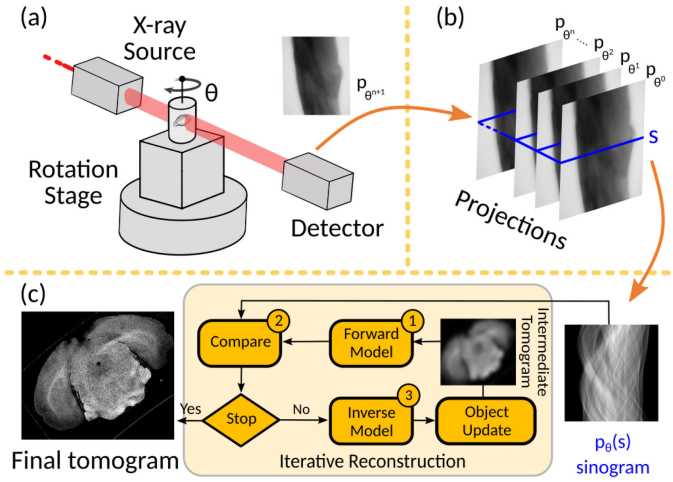


Fig. 2. (a) Illustration of the experimental setup where a target object is illumined with a synchrotron light source. Photons are attenuated by different amounts according to the object's attenuation coefficient profile and the photons' travel distance within the object. (b) Presentation of how the detector measurements are organized and stored as projections (from different angles). Once the experiment is completed, the sinograms are extracted from the projections, and the images corresponding to the object are iteratively reconstructed as shown in (c).

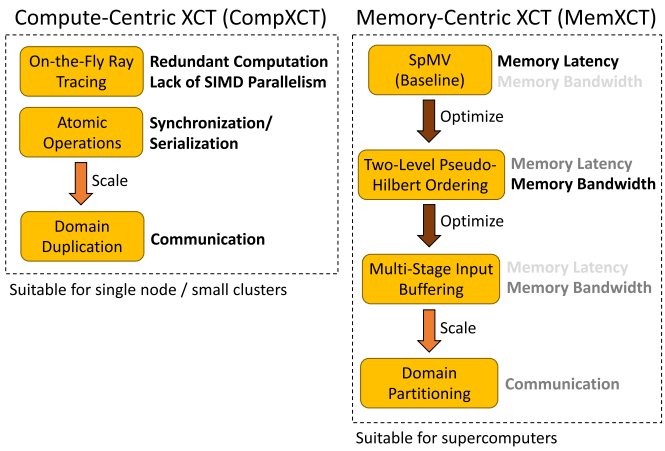


Fig. 3. Memory-centric and compute-centric approaches for XCT and their areas for potential optimizations. Shades of black color indicate higher contribution to performance bottleneck.

2.2 Iterative Formulation

Tomographic reconstruction solves the problem

$$\hat{x} = \underset{x \in C}{\operatorname{argmin}} \|y - Ax\|^2 + R(x), \tag{1}$$

where $\hat{x}$ is the reconstructed tomogram, A is the forward model, y is the sinogram, $R(x)$ is a regularizer functional, x is the search variable, and C is a constraint on x.

Almost all iterative solvers based on gradient descent perform three common steps in each iteration as depicted in Fig. 2. Specifically, for iteration i: first, the residual $r_i = y - Ax_i$ is found through *forward projection*; second, the gradient is found through *backprojection* as $\nabla\|y - Ax_i\|^2 = A^T r_i$; and finally, the candidate solution is updated in the negative-gradient direction as $x_{i+1} = x_i - \alpha\nabla\|y - Ax_i\|^2$, where α is the step size. Iterative approaches can also involve additional updates due to regularizer $R(x)$ and constraint C. However, this paper focuses on the common computational costs involving aforementioned steps.

2.3 Forward Projection and Backprojection

Forward and backprojections are the two most computationally demanding kernels in iterative tomographic reconstruction approaches. Many reconstruction libraries, such as Trace and TomoPy [11], [14], implement Siddon's algorithm [16] to perform ray tracing on tomogram domain so that the exact length and weight information on each voxel can be computed for each intersecting X-ray. For forward model, this information is used for computing the residuals, whereas for backprojection they are used for gradient and update operations. Since storing voxel indices and length information for all rays and voxels require significant memory, most state-of-the-art approaches perform ray tracing computations on-the-fly. For instance, a sinogram with dimensions 1501×2048 requires 56 GB memory to store intermediate data structures. Listing 1 illustrates the high level implementation of iterative reconstruction and repetitive computations of indices and lengths arrays. We term reconstruction algorithms that compute on-the-fly ray tracing information as *compute-centric XCT* (CompXCT).

Listing 1. High Level Implementation of CompXCT

```
1    for (int i = 0; i < num_iters; ++i)
     { //iterations
2    for (int j = 0; j < sinogram_rows; ++j){
3     float theta = rotations(j); // θ for row j
4     for (int k = 0; k < sinogram_cols; ++k){
5      int **indices =
6            intersecting_voxels (j, k, theta,
             tomogram);
7      float **lengths =
8        compute_lengths (j, k, theta, tomogram);
9    float m = sinogram[j, k]; // Data for simulated
     ray
10     float residual =
11       forward_model (m, indices, lengths,
         tomogram);
12     // Apply inverse model and then update
     tomogram
13   backprojection (residual, indices, lengths,
   tomogram);
14     }
15    }
16   }
```

2.4 Computational Bottlenecks and Overview of Our Solution: MemXCT

Fig. 3 illustrates the techniques used for traditional CompXCT and proposed MemXCT approaches in yellow boxes. The existing bottlenecks and their influence are given next to the areas for potential optimizations with different shades of black, in which darker shades indicate higher contribution to performance bottleneck.

CompXCT eliminates the need for storing intermediate data structures; however, since these data structures are recalculated for each iteration, they introduce additional computational complexity to the execution. Further, on-the-fly calculation of indices limits the optimizations for

vectorization as repeated sorting and padding operations nullify any performance benefit. As a remedy, MemXCT embraces a memory-centric approach that memoizes ray-tracing operations and performs efficient SpMV using compressed (vectorizable) data structures on both sinogram and tomogram domains. Further, the irregular data access patterns are normalized using pseudo-Hilbert ordering and first level cache utilization are improved using multi-stage buffering. These optimizations result in high-performance computations and converts performance bottlenecks from computation to memory.

The parallelization of CompXCT approaches is typically based on X-rays, where each measured ray can independently be projected on tomogram domain. This type of parallelization performs well for forward projection kernel (residual computation) since the main operation is reading the intersected voxel values from tomogram domain, i.e., rays perform *gather operations* during forward projection. However, subsequent backprojection and update steps require synchronization among parallelized rays since there can be many updates from different rays on the same voxel, i.e., rays perform *scatter operations* during backprojection and updates. Recent work deals with race conditions by either applying atomic operations [17] or duplicating the pixel domain across threads/processes and then performing a reduction [11], [12]. Unfortunately, the performance of atomic operations hinges on hardware implementations, which differ substantially across architectures [18], and typically results in significant performance degradation on massively parallel architectures. Domain duplication is also impractical for GPU-like architectures, since each parallel unit requires a replica of the domain and the total memory footprint almost always exceed available resources. Further, distributed memory parallelization using duplication can result in redundant communication cost [11].

In contrast, MemXCT partitions both sinogram and tomogram domains among parallel units and transforms scatter operations to gathers. Note that this transformation can result in irregular data accesses on both domains, however our pseudo-Hilbert index ordering coupled with multi-level buffering amortize performance penalty due to irregular data accesses. MemXCT also exploits Hilbert ordering (hence named two-level pseudo-Hilbert ordering) for process-level domain partitioning, which results in efficient communication and better connectivity between processes [19], [20]. We describe these optimization in detail in the following section.

3 MemXCT Optimizations

In this section, we provide detailed information about MemXCT. First, we explain the *baseline implementation*, in which we focus on explicit SpMV operations. Then, we present our system optimizations that use *two-level pseudo-Hilbert ordering* and *multi-stage input buffer*. We also explain our distributed memory parallelization considerations with MPI.

3.1 Baseline Implementation

MemXCT performs forward projection and backprojection operations as explicit SpMV operations to remove redundant and inefficient computations. Listing 2 shows the baseline compressed sparse row (CSR) SpMV kernel used for both forward and back projection. In forward projection, `x` and `y` correspond to tomogram and sinogram data, respectively, and vice-versa in backprojection. In either case, `ind` and `val` data correspond to precomputed pixel-ray intersection indices and lengths that are reused to avoid redundant computation.

Listing 2. Baseline MemXCT Kernel

```
1   #pragma omp parallel for schedule(dynamic,
    partsize)
2  for (int i = 0; i < numrow; ++i){
3    float acc = 0;
4    //vectorize
5    for (int j = displ[i]; j < displ[i+1]; ++j)
6      acc += x[ind[j]]*val[j];
7    y[i] = acc;
8  }
```

3.1.1 Regular and Irregular Accesses

Each fused multiply-add (FMA) operation in Listing 2 performs three important memory accesses: `ind` and `val` are *regular*, and `x` is *irregular*. The *regular* accesses are sequential and hence exhibit low memory latency. The *irregular* accesses to `x` exhibit long latency due to high L2 miss rates. MemXCT addresses this problem by improving cache reuse through novel data layout techniques that increase locality, as illustrated in Section 3.2.

3.1.2 Row Partitioning & Parallelization

Listing 2 involves gather operations only and hence is suitable for massive parallelization. MemXCT parallelizes the outer loop among *row partitions*. On KNL, partitions are distributed across dynamically scheduled OpenMP threads. On GPU, each partition corresponds to a CUDA thread block. Each OpenMP thread processes many row partitions, whereas each CUDA thread processes a single row in a partition.

3.1.3 Vectorization on KNL

MemXCT enables efficient instruction-level parallelization through vectorization of the inner loop in Listing 2. Each KNL vector-processing unit (VPU) can load multiple regular and irregular data, and multiply them in one step with an AVX-512 instruction. Then, they perform parallel reductions on partial data, and add results to the accumulator. We provide necessary data alignments for efficient vectorization.

3.1.4 Coalesced Memory Access on GPU

Our GPU implementation modifies Listing 2 to use the ELL (column-major) storage format instead of CSR. Transposed ELL data structures provide coalesced memory access through consecutive threads accessing consecutive memory locations. We minimize redundant computations and memory accesses due to ELL format by performing zero padding on thread-block (i.e., partition) level, rather than on matrix level.

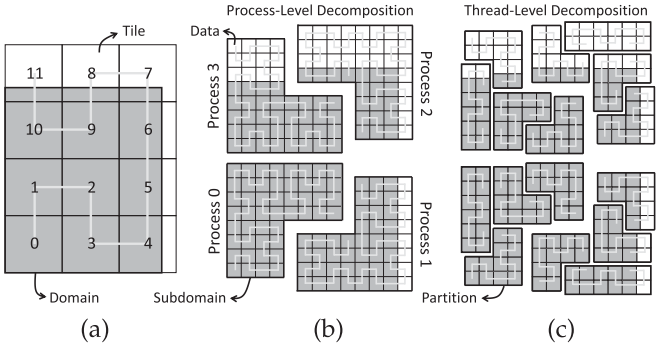


Fig. 4. Two-level pseudo-Hilbert ordering and domain decomposition at (a) tile level, (b) process level, and (c) thread level.

3.2 Two-Level Pseudo-Hilbert Ordering

MemXCT implements a two-level pseudo-Hilbert ordering on both tomogram and sinogram data of arbitrary size. Fig. 4 shows an example for a 13×11 domain. First, the domain is tiled with a minimum number of equi-sized square tiles with dimension a power of two. In the figure, tile size is 4×4 and 12 tiles are used to cover the 13×11 domain. These tiles are indexed with a Hilbert ordering for rectangular domains [21], as shown in Fig. 4a. A second-level Hilbert ordering is then applied to the data within each tile. Necessary rotations are performed to provide data connectivity among tiles, so as to achieve both data locality and connectivity for domain decomposition at the process and thread levels, as discussed next. The process-level domain decomposition is essential for MPI parallelization, as discussed in Section 3.4.

3.2.1 Irregular Data Access Patterns

As shown in Fig. 5, the processing of a single sinogram or tomogram results in a linear or a sinusoidal memory access footprint, respectively. In this example, they perform 25 and 30 accesses on respective domains. With row-major (naive) ordering of 2D data and 64 B cache line, each row in Fig. 5 would correspond to a single cache line. In this case, both tomogram and sinogram data would have 16 cache misses, yielding miss rates of 64% and 53%, respectively. Row-major (or column-major) ordering of 2D domains is very inefficient for XCT because of its poor cache locality. That is, a single cache line does not provide enough data reuse except on a few instances.

3.2.2 Cache Locality

In Fig. 5, Hilbert ordering maps each cache line to a 4×4 block in a 2D domain, increasing cache data reuse. As a result, cache misses are reduced to six and seven, yielding rates of 24% and 23% for forward projection and backprojection, respectively. Hilbert ordering is portable to different cache-line sizes thanks to its recursive nature. The cache locality provided by Hilbert ordering eliminates the memory latency bound of MemXCT to a great extent, as results show in Section 4.2.2.

3.2.3 Partition Locality

Hilbert ordering not only provides cache locality but also partition locality. That is, the outer loop of SpMV in Listing 2 is partitioned with respect to the index of y, as described in Section 3.1.2. In MemXCT, each partition remains connected at both the process and thread levels, thanks to the connectivity provided by two-level pseudo Hilbert ordering. In contrast, a Morton ordering would yield disconnected partitions, since it does not guarantee adjacent memory locations to have adjacent locations in the 2D domain. Partition locality is essential for our further optimizations.

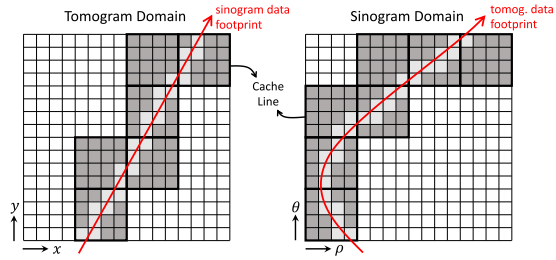


Fig. 5. Data access patterns on 2D domains.

3.3 Multi-Stage Input Buffering

MemXCT implements a multi-stage buffering mechanism for: (1) further reducing L2 cache miss rates of irregular data by explicit staging of corresponding accesses on an L1 buffer, and (2) reducing memory bandwidth consumed for index data by using two-byte addressing. Listing 3 shows SpMV kernel with input buffering, where stagedispl and stagenz arrays correspond to starting points of (multi-stage) buffers and number of (nonzero) elements in buffers, respectively.

Listing 3. Optimized MemXCT Kernel

```
1  #pragma omp parallel for schedule(dynamic)
2  for (int part = 0; part < numparts; ++part){
3    float output[partsize] = {0};
4    float input[buffsize];
5    for (int stage = partdispl[part];
6                    stage < partdispl[part+1]; ++
                     stage){
7      int start = stagedispl[stage];
8      //vectorize
9      for (int i = 0; i < stagenz[stage]; ++i)
10       input[i] = x[map[start+i]];
11     for (int j = 0; j < partsize; ++j){
12       int start = stage*partsize;
13       //vectorize
14       for(int i = displ[start+j];
15                   i < displ[start+j+1]; ++i)
16         output[j] += input[ind[i]]*val[i];
17     }
18   }
19   int start = part*partsize;
20   //vectorize
21   for (int i = 0; i < partsize; ++i)
22     if (start+i < numrows)
23       y[start+i] += output[i];
24 }
```

3.3.1 Data Reuse From Input Buffer

Fig. 6a shows memory access footprints for processing a 64×64 tomogram and sinogram partition in a 256×256

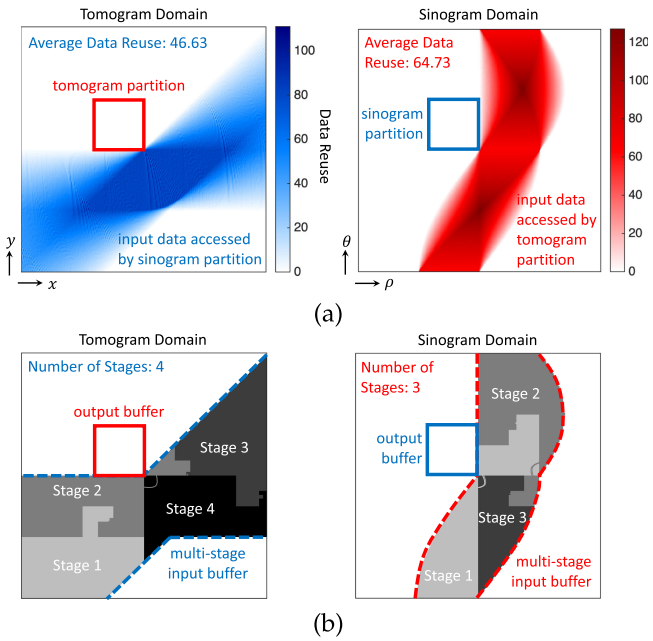


Fig. 6. (a) Tomogram and sinogram partitions and respective data access footprints, and (b) multi-stage buffer shapes.

tomogram and sinogram domain, respectively. The tomogram partition reads from the sinogram domain and the sinogram partition from the tomogram domain. Data access footprints are shown in tomogram and sinogram domains. Darker shades represents higher data reuse. For processing each partition, MemXCT explicitly moves required data from memory to an L1 buffer through `map` at line 9 of Listing 3. Then it reuses buffered data for irregular accesses and hence performs less memory access.

3.3.2 Multi-Staging

In a single-stage (naive) buffering, the required buffer size grows with the problem geometry. If the buffer is too large, it cancels the buffering benefit since it leaks into L2 or even memory [22]. As a remedy, MemXCT implements a multi-stage buffering strategy with a constant buffer size. In this case, irregular data are accessed through multiple buffer stagings. Stages are determined with respect to Hilbert ordering, ensuring data locality. As an example, Fig. 6b shows mapping of buffer stages on 2D tomogram and sinogram domains. With a buffer size of 32 KB, MemXCT performs irregular accesses through four and three stages for projection and backprojection, respectively. On KNLs, buffer size should be smaller than 32 KB L1 cache to avoid spill to L2 cache. On GPUs, the buffer is allocated through CUDA shared memory, guaranteeing reuse from L1 cache.

3.3.3 Staging Overhead

Input buffering involves a staging overhead. On KNL, an OpenMP thread stays idle while waiting staging to be completed. Similarly on GPU, CUDA threads across warps stall for synchronization before and after each staging. Input buffering also consumes some additional memory bandwidth for reading `map` data. The next two subsections describe two techniques for hiding buffering overhead and also to save some bandwidth.

3.3.4 Overlapping Staging and FMAs

MemXCT multi-stage buffering lends itself well to be used by the underlined hardware to hide staging overhead. On KNL, this is done through SMT (simultaneous multithreading): when there are multiple threads running on a single core, the hardware scheduler finds overlapping opportunities across buffer stagings and FMAs among threads. As a result, MemXCT enables effective SMT utilization. Similarly, GPU hardware also takes advantage of overlapping through block scheduling on SMs (streaming multiprocessors).

3.3.5 Saving Memory Bandwidth

Hilbert ordering and input buffering eliminate most of the memory latency due to irregular data accesses, and therefore MemXCT is bounded by memory-bandwidth consumed by regular data. To alleviate this bottleneck, we use 16-bit addressing to access input buffer (see `ind` at line 14 on Listing 3), rather than 32-bit addressing (as in `ind` at line 6 on Listing 2). 16-bit addressing can address buffer sizes up to 256 KB. This saves 25% of total bandwidth consumption of regular data, and provides additional speedup (see Section 4.2.3 for results.)

3.4 MPI Parallelization

Traditionally, one domain (either tomogram or sinogram) is partitioned while the other is duplicated across all processes. MemXCT partitions both tomogram and sinogram domains by distributing tiles evenly across MPI processes, as seen in Fig 4b. Each process is responsible for a single *tomogram subdomain* and a single *sinogram subdomain*. Each subdomain consists of a single or several tiles, e.g., subdomain 0 consists of tiles 0–2. While processes are not perfectly load balanced, it can be improved by finer tile granularity at the cost of more preprocessing.

3.4.1 Sparse Communications

MemXCT communicates only necessary data through `MPI_Alltoallv`. To explain, Fig. 7b shows communication footprint of two 256×256 tomogram subdomains shown in Fig. 7a. For example, tomogram subdomain 7 interacts only with sinogram subdomains 1, 2, 8–11, 13, and 14. Fig. 7c shows the corresponding communication matrix, where each entry represents communication between two processes. Communication size between each pair depends on the interaction footprint, e.g., process 7 sends more data to process 1 than 14, as seen in Fig. 7d. Partition locality provided by two-level pseudo-Hilbert ordering (described in Section 3.2) minimizes the footprint, and hence increase data reuse and reduce communications.

3.4.2 Overlapped Interactions

MemXCT communicates sinogram data rather than tomogram data because: (1) sinogram data are smaller in many applications, and (2) it yields more data reuse as compared to tomogram data. In forward projection, tomogram subdomains send partial sinogram data to corresponding sinogram subdomains where overlapped data are reduced, e.g., processes 8, 9, and 11 reduce partial sinogram data received

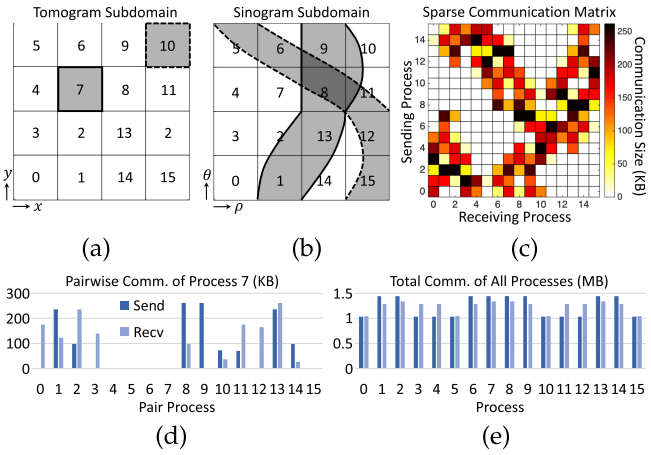


Fig. 7. Communication footprint of two (a) tomogram subdomains on (b) sinogram subdomains. (c) Resulting commnication matrix for all processes. (d) The amount of outgoing/incoming data from/to process 7 to/from others. (e) Total incoming and outgoing data for all processes.

from processes 7 and 10. In backprojection, processes duplicate overlapped sinogram data and send them to interacted processes which perform backprojections on their respective tomogram subdomains. The communication matrix for backprojection is the transpose of the one in Fig. 7c. The total amount of communications and load balancing for all processes is shown in Fig. 7e.

3.4.3 Parallelization Overhead

MemXCT parallelization of forward projection mathematically corresponds to a factorization of the projection matrix as $A = RCA_p$. As a result, forward projection can be modeled as three fundamental steps: A_p, C, and R which correspond to partial forward projection, communication, and reduction operations. Backprojection can also be seen as simply $A^T = A_p^T C^T R^T$. The multiplication costs of A_p and original A are the same because they involve the same number of $\mathcal{O}(MN^2)$ nonzeroes. Thus, communication C and reduction R can be seen as the MemXCT parallelization overhead. Both C and R have $\mathcal{O}(MN\sqrt{P})$ nonzeroes, where P is the number of processes. It is because when P quadruples, total communication footprint on sinogram domain doubles. For communications, there is an extra $\mathcal{O}(\sqrt{P})$ term comes due to the handshake overhead between processes. On the other hand, compute-centric approach involves race conditions and hence it duplicates the tomogram domain across all processes [11]. In that case, duplicated domains has to be reduced through `MPI_Allreduce` at the end of each backprojection, yielding $\mathcal{O}(N^2 \log P)$ parallelization overhead. Resulting computational complexities are shown in Table 1.

3.5 Other Details

MemXCT requires an extra preprocessing step for avoiding redundant and inefficient computations as opposed to a compute-centric approach. The preprocessing involve: (1) Hilbert ordering and domain decomposition, (2) ray tracing for constructing forward projection matrix, (3) sparse transposition for constructing backprojection matrix, and (4) row partitioning and building corresponding buffer data structures. Preprocessing is MPI+OpenMP parallel and is performed on CPU.

TABLE 1
Computational Complexities

	Sequential	Trace	MemXCT
Memory	$MN + N^2$	$MN/P + N^2$	$MN^2/P + MN/\sqrt{P}$
Comput.	MN^2	MN^2/P	$MN^2/P + MN/\sqrt{P}$
Comm.	N/A	$N^2 \log P$	$MN/\sqrt{P} + \sqrt{P}$

M: # of projections, N: # of channels P: # of processes.

3.5.1 Preserving Data Locality

MemXCT preserves data locality through all matrix manipulations, which is essential for our performance optimizations. For example, constructing backprojection matrix requires a sparse matrix transposition of the forward projection matrix. MemXCT performs this through a scan-based matrix transposition [23] which preserves data ordering, rather than an atomic-based transposition which randomizes data ordering.

3.5.2 Iterative Solution

There is a plethora of solution schemes for iterative XCT. To mention a few, recent work implements SIRT [11], SGD [17], and ICD [24] iterations. Any of them can be implemented for our proposed memory-centric approach in a plug-and-play manner with minor modifications. Nevertheless, MemXCT implements CG iterations [25], which has a faster convergence rate than any of them (at a higher per-iteration cost). It is simply because: (1) a full gradient is found rather than a partial gradient, (2) optimal step size is found analytically via an additional forward projection, and (3) visiting redundant directions is prevented via three-term recursion. We use a heuristic early termination of iterations to prevent *overfitting*, which is practically considered as a regularization method. Convergence results are shown in Section 4.1.2.

4 Numerical Results

All experiments in this paper are performed on ALCF Theta [26], NCSA Blue Waters [27], ALCF Cooley [28], an IBM Minsky node [29], and an Nvidia DGX-1 node [30]. Table 2 characterizes these machines. The fifth column (B/W) is the theoretical device memory bandwidth, assuming that ECC (error correcting code) degrades theoretical bandwidth of K20X and K80 by 15% [31]. Link describes interface between host and device.

We used six datasets, as shown in Table 3, in order to evaluate application performance. The first four are artificially created for performance evaluation and the latter two are from real synchrotron experiments at APS. Measurement sinograms are given for a single slice. The artificial datasets follow parallel raster scan geometry just as the real datasets. The irregular and regular data footprints, defined in Section 3.1.1, are given for all datasets in Table 3. The first/second entry for memory footprints are accessed in forward/backprojection, respectively. RDS1 involves a

TABLE 2
Key Features of Machines Used for Experiments

Machine	Nodes	Accel.	Device Mem.	B/W	Node Mem.	Link
Theta	4392	KNL	16 GB MCDRAM	400 GB/s	192 GB	90 GB/s
Blue W.	4228	K20X	6 GB DDR5	121.5 GB/s	32 GB	PCIe
Cooley	126	2×K80	12 GB DDR5	204 GB/s	384 GB	PCIe
Minsky	1	4×P100	16 GB HBM2	720 GB/s	128 GB	NVLink
DGX-1	1	8×V100	16 GB HBM2	900 GB/s	512 GB	NVLink

TABLE 3
Dataset Details and Memory Footprints

Name	Sinogram ($M \times N$)	Sample	Irregular Data	Regular Data
ADS1	360×256	Artificial	256/360 KB	215/215 MB
ADS2	750×512	Artificial	1.0/1.5 MB	1.8/1.8 GB
ADS3	1500×1024	Artificial	4.0/6.0 MB	14/14 GB
ADS4	2400×2048	Artificial	16/19 MB	90/90 GB
RDS1	1501×2048	Shale Rock	16/12 MB	56/56 GB
RDS2	4501×11283	Mouse Brain	500/198 MB	5.1/5.1 TB

TABLE 4
Comparison With Compute-Centric Approach

		Preproc.	Reconst.	Per-Iter.	Speedup
ADS2	**Trace**	N/A	26.05 s	579 ms	1×
	MemXCT	4.00 s	0.53 s	11.8 ms	49.2×
RDS1	**Trace**	N/A	425.3 s	9.45 s	1×
	MemXCT	146.3 s	62.00 s	1.37 s	6.86×

shale sample [32] and RDS2 involves a brain sample. RDS1 is available open source [33], and RDS2 is proprietary.

4.1 Evaluating Overall Performance

4.1.1 Comparison With Compute-Centric Approach

We compare our memory-centric MemXCT with compute-centric Trace [11], an open-source high-performance implementation that employs SIRT iterations. To enable a one-to-one comparison, we implement SIRT, and run 45 iterations with both codes on a single KNL for the ADS2 and RDS1 datasets (see Table 3). Table 4 reports solution times and corresponding speedups. In the best case, where MemXCT memory footprint fits within MCDRAM, it performs each iteration 49.2× faster than Trace. In the worst case, where MemXCT is bounded by slow DRAM bandwidth due to its large memory footprint, it still performs each iteration 6.86× faster. Trace memory footprint fits within MCDRAM in both cases, but it has to perform redundant and inefficient computation.

Table 4 reports preprocessing overhead of MemXCT. Although the preprocessing step appears to be a significant fraction of overall time, when it comes to many-slice reconstruction, the preprocessing cost is paid only once for the first slice. It is then reused for all the remaining slices, as shown in Table 5.

TABLE 5
Reconstruction on Various Nodes-Machines

Nodes-Machine	Preproc.	Speed.	Recon.	Speed.	All Slices
1-Theta (1 KNL)	139 s	1×	63.3 s	1×	1.44 d*
8-Theta (8 KNL)	16.5 s	8.42×	3.33 s	19.0×	1.89 h
8-Cooley (16 K80)	25.5 s	5.45×	2.89 s	21.9×	1.64 h
32-Blue W. (32 K20X)	14.6 s	9.52×	1.82 s	34.8×	62.1 m
32-Theta (32 KNL)	4.54 s	30.6×	1.37 s	46.2×	46.8 m
32-Cooley (64 K80)	6.31 s	22.0×	1.22 s	51.9×	41.6 m

* *Calculated based on single-slice execution time.*

4.1.2 Iterative Convergence

MemXCT uses CG for iterative solutions, as discussed in Section 3.5.2, as opposed to SIRT used by Trace. Fig. 8a presents convergence properties by comparing L-curves of CG and SIRT up to 500 iterations: horizontal and vertical axes represents residual and reconstruction norms, respectively (see Section 2.2 for iterative formulation). As L-curve suggests, CG solution experiences overfitting soon after 30 iterations where the image does not further improve, but

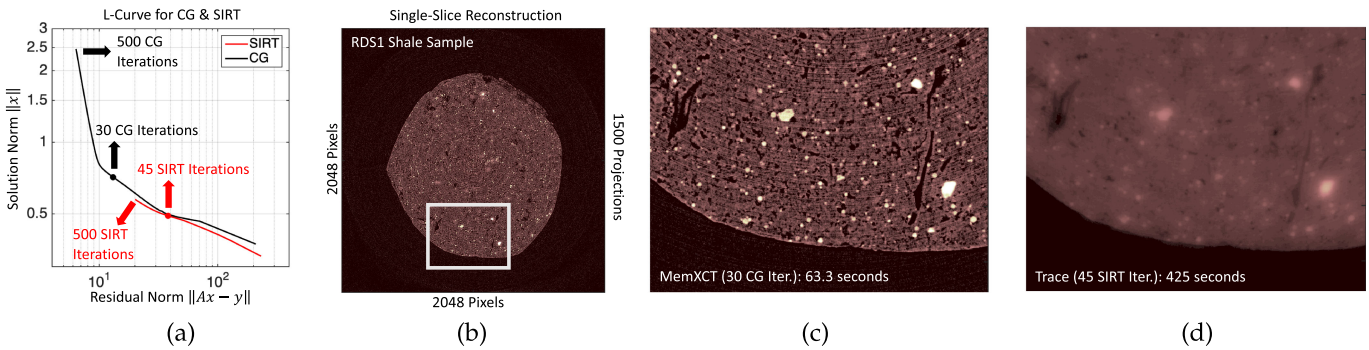


Fig. 8. For RDS1: (a) L-curves for CG and SIRT iterations. (b) Single-slice reconstruction with (c) CG and (d) SIRT iterations.

instead starts to be polluted by noise. Therefore we terminate CG solution after 30 iterations. On the other hand, SIRT does not converge even with 500 iterations. Fig. 8b shows a single-slice reconstruction from RDS1, and (c) and (d) compare image details after 30 CG iterations with MemXCT and 45 SIRT iterations with Trace, respectively.

4.1.3 Machine-Specific Considerations

We reconstruct RDS1 on small number of Theta, Cooley, and Blue Waters nodes. Machines were described briefly earlier. Unfortunately, reconstruction does not fit within DGX-1 or on less than eight nodes of Cooley or 32 nodes of Blue Waters, due to limited memory of their respective GPUs and the large memory complexity of MemXCT. Nevertheless, reconstruction fits well into a single Theta node thanks to its large DRAM capacity.

Table 5 reports RDS1 preprocessing and reconstruction times on various numbers of nodes-machines. Results show that 32 nodes of all machines enjoy comparable time to solution: reconstruction of all slices reduces from 1.44 days down to about or less than an hour. Preprocessing time also scales well and, in fact, it is insignificant compared to reconstruction time of all slices. Table 5 also shows the super-linear speedup property of MemXCT on KNLs: 8-Theta is $19\times$ faster than 1-Theta. This is a result of shrinking per-node memory footprint and extra memory bandwidth gain when it fits within 16 GB MCDRAM capacity. The super-linear speedup also demonstrates effective domain decomposition and reduced communications described in Section 3.

4.2 Performance Optimizations

This subsection presents single-device performance gains due to the optimizations described in Section 3. Fig. 9 shows forward and backprojection performance metrics for ADS1 through ADS4 on KNL and GPU. ADS3 and ADS4 are too large to fit in a single GPU (see Table 3). Hilbert ordering and input buffering are applied to the baseline in order because the first optimization enables the second. Since performance is dependent on both dataset and device, we tune all results (including the baseline) independently for maximum GFLOPS as described in Section 4.2.4.

Since there are two FLOPs per non-zero element in the projection matrix (one multiplication and one addition per FMA), we calculate the GFLOPS metric as $2N_{\mathrm{nz}}/t$, where t is the time measured for a single forward/backprojection. Similarly, we calculate average memory bandwidth utilization for regular data only as $N_{\mathrm{nz}} \times B_{\mathrm{reg}}/t$, where B_{reg} is regular data (in bytes) read from memory per FMA, respectively. L2 miss rates are measured by Intel VTune profiler. The performance metrics are aggregated over many iterations.

4.2.1 The Baseline

GFLOPS on KNL drops as dataset size increases, regardless of whether or not it fits into high-bandwidth MCDRAM. The reason for this behaviour is that the baseline is bounded by memory latency rather that memory bandwidth, due to the high L2 miss rates of irregular access. As a result, larger datasets suffer more performance degradation due to their higher L2 miss rates, as seen in Fig. 9b. In contrast, GPU performance improves slightly with larger datasets, as more parallelism hides the latency.

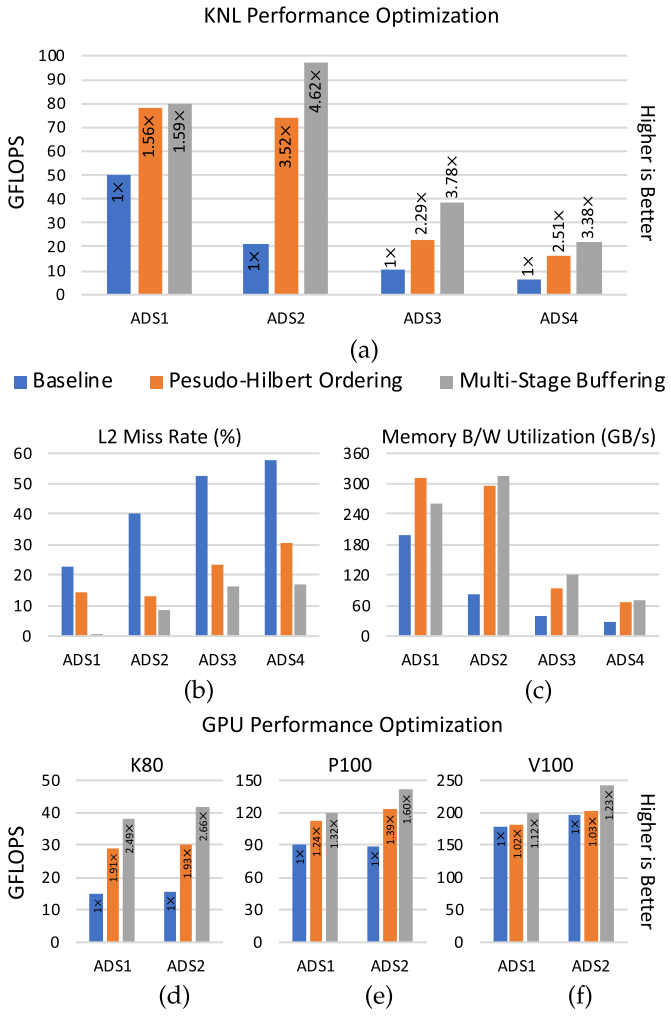


Fig. 9. KNL performance: (a) GFLOPS, (b) L2 miss rate, and (c) memory bandwidth utilization. GPU GFLOPS performance: (d) K80, (e) P100, and (f) V100.

4.2.2 Pseudo-Hilbert Ordering

provides data locality and reduces the L2 miss rates for all datasets as seen in Fig. 9b. ADS1 does not benefit from Hilbert ordering as much as other datasets due to its small size. Nevertheless, as opposed to the baseline, the performance of Hilbert ordering is bounded by memory bandwidth consumed by regular data rather than memory latency. On KNL, regular data smaller than 16 GBs (ADS1 and ADS2) fits completely into MCDRAM and performs better whereas large regular data (ADS3 and ADS4) does not and is bounded by low DRAM memory bandwidth. On GPUs, the improvement is less noticeable with larger L2 cache because irregular data are better cached. As a result, pseudo-Hilbert ordering speeds up K80, P100, and V100 baselines by about $1.93\times$, $1.39\times$, and $1.03\times$, respectively.

This analysis shows that ADS1 and ADS2 use at least 78% and 74% of the theoretical MCDRAM bandwidth, respectively. The ADS3 regular data (28 GB) is larger than MCDRAM (16 GB) where it is cached partially. We cannot say the same for ADS4 because it has too large regular data

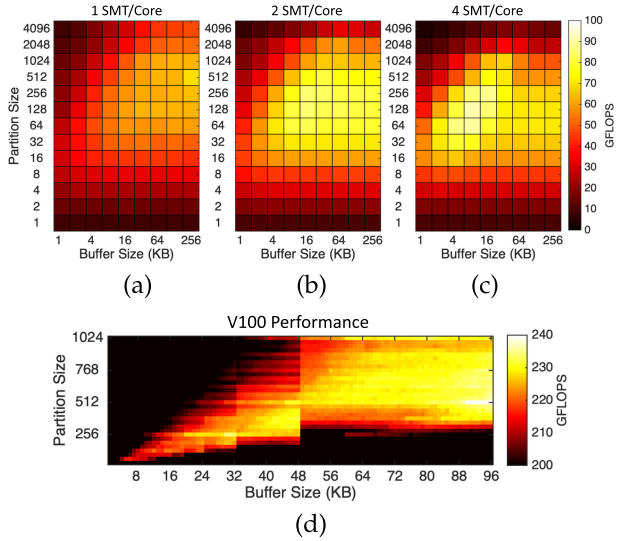


Fig. 10. Tuning input buffer and block size for (a)-(c) various SMT/core on KNL and (d) for V100.

to be significantly cached at MCRAM. As a result, we deduce that ADS4 uses at least 73% of the theoretical DRAM bandwidth. These utilizations agree well with STREAM benchmarks in previous work [34]. Similarly, K80, P100 and V100 use 78%, 69%, and 92% of their theoretical HBM bandwidths, respectively. These numbers agree well with benchmark measurements in previous work [35].

4.2.3 Multi-Stage Buffering

as we described in Section 3.3, comes with an staging overhead. On KNL, the gain amortizes the overhead when dataset is large, as seen in ADS2 and up. ADS1 is not large enough to show any further performance improvement with input buffering. The bandwidth utilization for input buffering in Fig. 9c and Figs. 9d, 9e, and 9f is adjusted with respect to additional memory-to-buffer mappings as well as reduced bytes for buffer-address indices.

It is worth to comment that reduced L2 miss rate saves significant memory bandwidth on K80 since its utilization due to regular data increases to at least 67% of theoretical peak as seen in Fig. 9d. The respective utilizations on P100 and V100 drop slightly, if not remain the same, because their L2 miss rates are already low and bandwidth utilizations are already high thanks to Hilbert ordering and their large L2 capacity. As a result, we can deduce that GFLOPS improvements on P100 and V100 are solely provided by the bandwidth saving due to reduced number of bytes needed for addressing shared-memory.

4.2.4 Tuning

The baseline and Hilbert ordering are relatively simple on both KNL and GPU architectures. We perform an exhaustive search and find out that blocks size of 128 scheduled dynamically among 128 threads (2 SMT/core) provides good single KNL performance for ADS1 through ADS4. Similarly, block size of 32 or 64 provides good single GPU performance.

For input buffering optimization, parameters should be re-tuned along with buffer size for effective SMT utilization on KNL. Figs. 10a, 10b, and 10c shows GFLOPS heat maps for ADS2 with various block and buffer sizes, and different numbers of SMT/core. Results show that input buffering can use more SMTs per core because it finds opportunity to overlap buffer stagings and accesses among SMTs. The following factors need to be considered when using SMTs in this way: A larger block size increases the per-block memory footprint and thus data reuse from the buffer, but also increases the number of buffer stagings per block, which comes with an overhead. In that case, it is better to increase buffer size for limiting the number of stagings. However, too few stagings decreases the opportunity to overlap buffer stagings and accesses among SMTs. Also, too large a buffer size can result in leaks to L2.

On KNL, peak GFLOPS performance for ADS1 and ADS2 is achieved with 4 SMT/core and buffer size of 8 KB. Tuning for ADS3 and ADS4 is cumbersome since they are severely limited by DRAM bandwidth and SMT effect is not that apparent. Therefore we use 2 SMT/core, and buffer size of 16 KB for them. Block size of 128 provides good performance for all datasets.

On GPUs block size of 512 or 1024 and buffer size of 48 KB or 96 KB provides good single-GPU performance for all GPUs. It is worth to note that addressable shared-memory size is limited to 48 KB on P100 and K80 and therefore tuning space is half the size. We follow a similar tuning strategy for GPUs as explained for KNL, as seen in Fig. 10d.

TABLE 6
Comparison With MKL and cuSPARSE for ADS2

	KNL	K80	P100	V100
MKL/cuSPARSE	1×	1×	1×	1×
MemXCT Baseline	1.42×	0.52×	1.39×	1.79×
Pseudo-Hilbert Ordering	4.99×	1.13×	1.93×	1.84×
Multi-Stage Buffering	6.55×	1.56×	2.23×	2.11×

4.2.5 Comparison With Existing Libraries

We compare MemXCT forward/backprojection kernel performance with SpMV functions of existing libraries. We use Intel MKL library on KNL, and Nvidia cuSPARSE library on GPU. Table 6 reports speedups of our MemXCT baseline implementation and optimizations compared to CSR (on KNL) and column-major ELL (on GPU) SpMV of respective libraries. Evidently, MemXCT baseline implementation is faster than the existing libraries, with the exception for K80. This is due to K80 small L2 cache compared to P100 and V100 GPUs. On P100 and V100, our baseline is 1.39× and 1.79× faster since cuSPARSE pads CSR matrix with −1 [36] which requires extra branches whereas we pad with 0 and perform redundant multiplication with 0 to avoid thread divergence on GPUs. Also, cuSPARSE pads CSR on a matrix level whereas we pad on a thread-block (partition) level, which saves some of the redundant operations. As a result, we outperform existing libraries thanks to our application-specific SpMV optimizations. Nevertheless, both MKL and cuSPARSE are optimized for general SpMV whereas our kernels are optimized for XCT application.

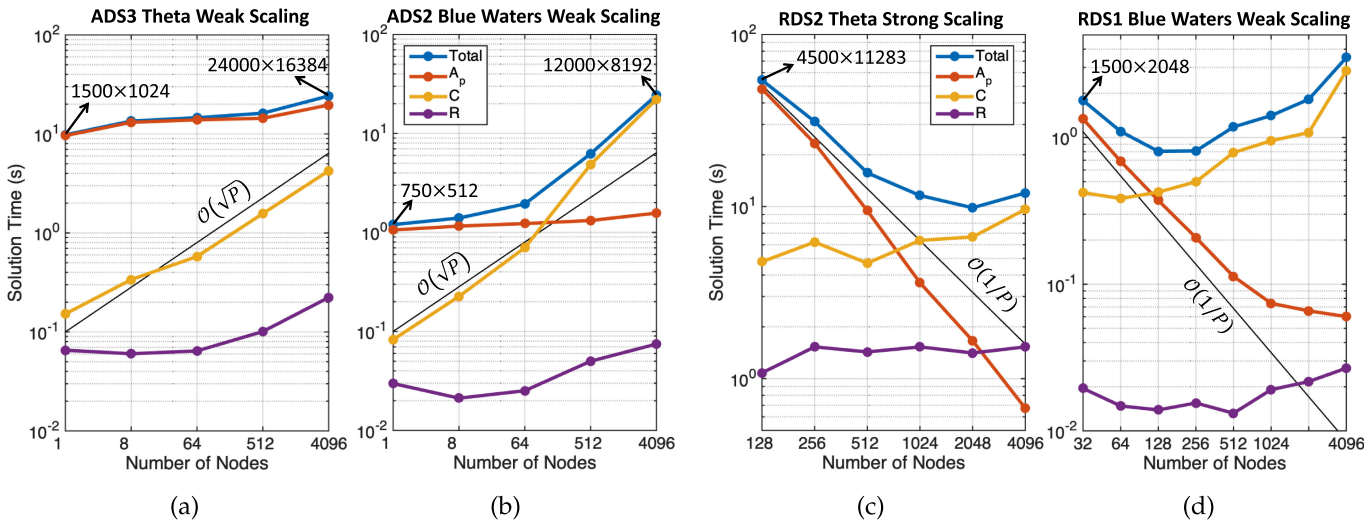


Fig. 11. Weak scaling: (a) ADS3/Theta, (b) ADS2/Blue Waters. Strong scaling: (c) RDS2/Theta, (d) ADS3/Blue Waters.

4.3 Scalability

For presenting scaling of MemXCT on a large number of nodes, we perform weak and strong scaling experiments on Theta and Blue Waters. For each experiment, we perform a full solution with 30 CG iterations, and report total reconstruction times as well as A_p, C, and R kernel times as they are defined in Section 3.4. Kernel times involve both forward and backprojection. In order to accurately measure each individual kernel, we placed the necessary artificial barriers, though they slightly increase total execution time. Among the kernels, C involves inter-node communication therefore we include its host-device communication times. For demonstrating scaling, $\mathcal{O}(\sqrt{P})$ and $\mathcal{O}(1/P)$ curves are included in Fig. 11 as well as the sinogram data dimensions.

4.3.1 Weak Scaling

Each weak scaling experiment is performed by starting from the reconstruction of a root dataset, and then doubling the number of channels and projections in a sinogram on each step. As computational cost increases eight times per step, the number of nodes is increased eight times accordingly. Figs. 11a and 11b show weak scaling of root datasets ADS3 and ADS2 on Theta and Blue Waters, respectively. Both experiments exhibits good weak scaling except communication operations C because of its $\mathcal{O}(\sqrt{P})$ complexity, as explained in Table 1. On Blue Waters, weak scaling is bounded by communication on 512 nodes and up; on Theta, communication is not the bounding component, but rather A_p. The difference is due to architectural differences across machines.

4.3.2 Strong Scaling

For strong scaling, we reconstruct RDS2 (brain) and RDS1 (shale) samples on Theta and Blue Waters respectively. We increase numbers of nodes and the dataset sizes are fixed. The minimum number of nodes needed for RDS2 and RDS1 to fit well within their respective systems is 128 and 32 nodes respectively. Reconstructions are scaled up to 4096 nodes of each system. Figs. 11c and 11d show strong scaling results. Theta exhibits good scaling up to 2048 nodes where as Blue Waters scales up to 128 nodes. The main reason (apart from the difference in network bandwidth and topology) is that the RDS2 solution is $\sim$91 $\times$ more costly than the RDS1, and therefore the former scales better than the latter. Also, A_p kernel exhibits super-linear speedup, demonstrating efficient domain partitioning and high-bandwidth memory utilization, as discussed in Section 4.1.3. RDS2 reconstruction results are in Fig. 1. In contrast, A_p performance saturates on 1024 nodes of Blue Waters (and up) since smaller per-node computation penalizes GPU performance.

TABLE 7
Comparison of Theta and Blue Waters

	RDS1	RDS2	12000×8192
BW	805 ms (128 K20X)	74 s (4096 K20X)	24.4 s (4096 K20X)
Theta	474 ms (128 KNL)	10 s (2048 KNL)	3.25 s (4096 KNL)

4.3.3 Comparison of Theta and Blue Waters Systems

For cross-comparing Theta and Blue Waters systems, we pick the fastest reconstructions of both RDS datasets and their corresponding week scaling versions when ran on 4096 nodes. Results are shown in Table 7. RDS1 (shale) sample reconstruction runs fastest on 128 nodes on both Theta and Blue Waters. In this case, Theta is about 1.7 $\times$ faster than Blue Waters. RDS2 (brain) sample reconstruction runs fastest on 2048 nodes of Theta and requires at least 4096 nodes to fits well into Blue Waters. In this case, Theta is 7.4 $\times$ faster than Blue Waters. Lastly, Theta is about 7.5 $\times$ faster on reconstructing the 12000 $\times$ 8192 dataset.

5 Related Work and Discussion

Iterative reconstruction algorithms provide superior image quality considering analytical approaches, however their usage has so far been limited with their computational demands [37], [38], [39].

The multicore parallelization of iterative reconstruction algorithms has long been studied [40], [41], [42], [43]. However, most prior approaches provide limited scalability and only consider optimizations in the object domain. Further,

they suffer from redundant computations when dealing with large datasets.

With the emergence of high-throughput many-core architectures and state-of-the-art reconstruction algorithms, the applicability of iterative techniques has become more feasible [13], [24], [44], [45]. Especially in medical imaging, iterative reconstruction approaches have been extensively used to provide high-quality 3D reconstructions [46], [47], [48] in order to limit patient radiation exposure [4], [49]. Most of these approaches rely on GPUs to meet computational requirements of these algorithms [50], [51], [52]. Although GPUs can provide sufficient computational throughput, their limited memory can accommodate only small- to medium-scale datasets. For large-scale dataset host-device communication is known to be dominating overhead in GPUs [53]. Further, most of these approaches rely on on-the-fly computation of ray tracing, i.e., CompCXT type of data access pattern, which shows suboptimal performance compared to MemCXT.

Li *et al.* developed cuMBIR [17], a framework for model-based iterative reconstruction on GPUs. They implement two solvers ICD and SGD, and propose several optimizations including unified thread mapping. Their method is also based on on-the-fly computation of A matrix, and performs redundant computations. Further, their solution focuses on small to medium scale datasets that can fit into single node GPUs. In comparison, MemCXT is optimized for very large tomographic datasets that require not only single node performance but also efficient large-scale execution.

Wang *et al.* [44] use KNL many-core capabilities to reconstruct 1024^2 tomograms (or 1024^3 3D volumes). They introduce the *non-uniform parallel super-voxel* to exploit sinusoidal bands in sinograms and optimize data access patterns. Our approach shares similarities with their work, but we consider parallelization and performance optimizations in both the tomogram and sinogram domains. Moreover, we further improve data access patterns and communication using two-level pseudo Hilbert ordering and multi-stage input buffering on both domains. Our evaluation also extends to extremely large objects (11293^2 rather than 1024^2), showing the usability of our approach for very large datasets.

In a more recent work by our group, we further extended the optimizations proposed in this paper and improved the reconstruction performance with high-end GPU clusters [54]. Specifically, we applied hierarchical communications and reductions to minimize inter-node communication overhead, improved the memory access performance with 3D batch parallelism, and used mixed-precision types to maximize throughput and bandwidth utilization. These optimizations enabled reconstruction throughput to reach 34% of Summit's peak performance (or 65 PFLOPS).

The two-level pseudo Hilbert ordering technique used in this work is a form of space-filling curves. To our knowledge, the first space-filling curve example was presented by Peano in 1890 [55]. Later, space-filling curves have been successfully applied to improve performance of many applications [56], [57], [58], [59], [60], [61], [62]. In [63], Mellor-Crummey *et al.* showed that data and computation reordering based on space-filling curves can improve the performance of irregular applications significantly. Moon *et al.* analyzed clustering properties of Hilbert space-filling curve [64] on different query shapes [65]. Reissmann *et al.* showed the effects of Hilbert and Morton curves[66] on energy and locality [62]. In our work, we applied two-level pseudo Hilbert ordering to optimize the performance of tomographic reconstruction data access patterns and inter-process communication.

6 Reproducibility

Reproducibility of experimental results is essential to scientific research. To promote the reproducibility of experimental results in the HPC research community, the SC Student Cluster Reproducibility Committee has, since 2016, selected one of the papers from the past year's SC conference to serve as the benchmark for the Reproducibility Challenge [67]. Based on our paper's artifact descriptor and its suitability to the SCC, we were deeply honored that our SC19 paper [1] was selected to serve as the 2020 Reproducibility Challenge benchmark.[1] Nineteen teams, each of which is composed of six college students and their advisor who design, construct and utilize their own clusters to run computational experiments, were selected to participate SCC at SC20 and were tasked to replicate the findings of the publication.[2]

This section describes the pertinent features of the MemXCT codebase and collateral materials for the Reproducibility Challenge. Some of these features were already in our repository so that future papers can reproduce our results for comparison purposes. Some were added and/or enhanced in consultation with the Student Cluster Reproducibility Committee. We summarize the key findings of the nine top-scored teams and present the insights that can be potentially applicable to other HPC research experiments.

6.1 Datasets Used for Reproducibility

To support the reproducibility and for benchmarking, we provide *Test Datasets* and *Challenge Datasets* to the participating teams.

6.1.1 Test Datasets

The Test datasets are released two weeks in advance to the SCC kickoff. This allowed students to: 1) Compile and run the codes, 2) Verification by visualizing the results, and 3) Tuning system performance according to the teams' respective hardware. The Test datasets comprise ADS1, ADS2, ADS3, and ADS4, the artificial portion of the benchmark set that were used to generate the results presented in this paper. These benchmark datasets were chosen to represent very different points in the four dimensions (columns) shown in Table 3.

A major constraint in MemXCT is the memory footprint, which grows with respect to the memory complexity as formulated in Table 1. For example, ADS1 and ADS2 are small enough to fit into a single GPU whereas ADS3 and ADS4

1. Please see the blog post entitled "SC20 Student Cluster Reproducibility Committee Chooses Benchmark Wisely" by Scott Michael and Stephen Harrell for details.

2. Please see the blog post entitled "Global Lineup Will Compete at SC's First Virtual Student Cluster Competition" by Christine Baissac-Hayden for details.

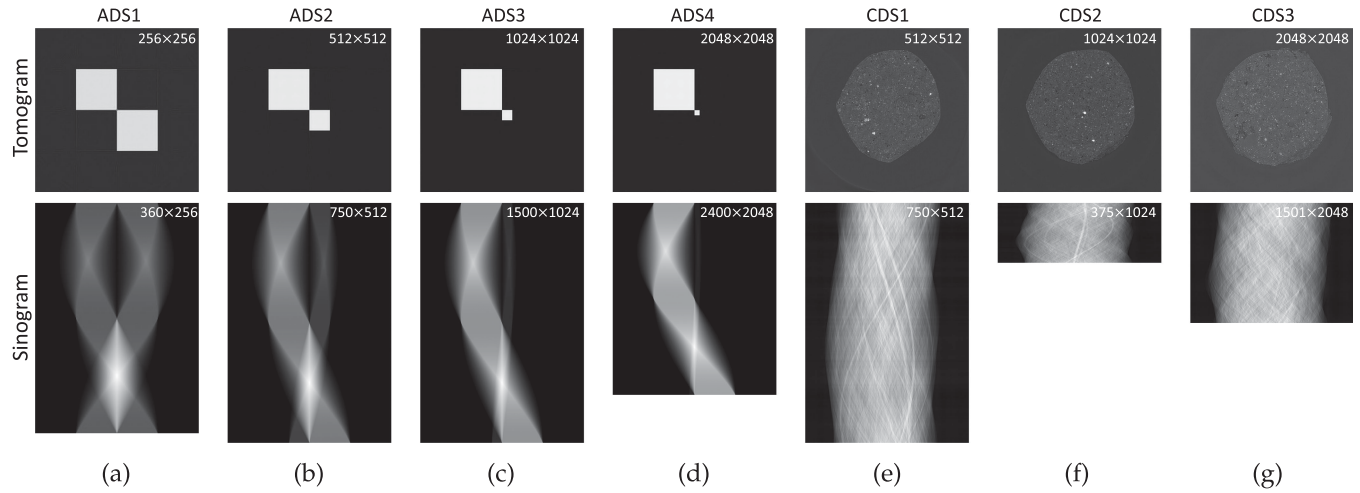


Fig. 12. (a)–(d) Artificial *Test Datasets* are used for performance benchmarks for the original paper and at the reproducibility challenge. (e)–(g) Real *Challenge Datasets* that are introduced for additional reproduction.

TABLE 8
Challenge Datasets Used for Reproducibility

Name	Sinogram ($M \times N$)	Sample	Irregular Data	Regular Data
CDS1	750×512	Shale Rock	1.0/1.5 MB	1.8/1.8 GB
CDS2	375×1024	Shale Rock	4.0/1.5 MB	3.5/3.5 GB
CDS3	1501×2048	Shale Rock	16/12 MB	56/56 GB

are larger. Therefore, ADS3 and ADS4 datasets are easier to accommodate in KNL nodes. Providing the exactly same benchmark datasets used in our original experiments gives students the opportunity to directly compare their performance results to those in the original paper.

For verification, we asked students to visualize their input (sinogram) and output (tomogram) datasets by using the Fiji software [68], although, students are free to use any other visualization tool. They import the raw output image files and inspect these to verify the execution of their code. The input (sinogram) and output (tomogram) Test Datasets are shown in Figs. 12a, 12b, 12c, and 12d.

6.1.2 Challenge Datasets

We revealed the Challenge Datasets by the time of the SCC. These datasets comprise CDS1, CDS2, and CDS3 of different dimensions and are listed in Table 8 along with their memory footprints. We generated these datasets by rescaling the sinogram of RDS1 in Table 3. Each scaling was applied to a different slice of the original 3D dataset and the output tomograms were not revealed to the students. We matched the dimensions of CDS1 and CDS3 with those of ADS2 and RDS1, respectively, to provide students a one-to-one benchmarking opportunity. We chose dimension of CDS2 large enough to require substantial computational throughput but small enough to still fit into a single GPU memory. The sinogram and tomogram of the Challenge Datasets are shown in Figs. 12e, 12f, and 12g. All datasets were provided through the repositories as explained in the next subsection.

6.2 Codebases and Repositories

The CPU and GPU codebases that were used for performing benchmarks in this paper and for the SCC reproducibility challenge were provided in respective GitHub repositories:

```
https://github.com/merthidayetoglu/MemXCT-CPU
https://github.com/merthidayetoglu/MemXCT-GPU
```

The MemXCT application requires MPI and C++ compiler (and CUDA compiler for GPU). Instructions for compiling, job execution, and preparing input file were provided to the participant teams. The repositories include persistent links to the Testing Datasets ADS1, ADS2, ADS3, and ADS4 and Challenge Datasets CDS1, CDS2, and CDS3. We also included the visualizations of the Test Datasets in the repository, for student teams to verify their results against any bugs.

6.3 Machines Used for Reproducibility

In a typical year, each of the SCC teams designs and assembles their own cluster on the SC show floor. This often results in a very diverse set of hardware and software configurations. However, due to the COVID-19 pandemic, the SC20 SCC was completely virtual. Student teams utilized virtual machines provided by Microsoft Azure cloud service. Although, the students had access to a wide variety of Azure instances, some of the more exotic hardware configurations that had been seen in the past with the physical competition were not available. Details of the virtual machine configurations and the teams that utilized them are shown in Table 9. Each team configured their own software environment, including OS, drivers, compilers, and MPI implementations. The details of the software configurations can be found in teams' critiques in the special issue herein.

6.4 Reproducibility Results

For reproducibility, we turned on all MemXCT optimizations on, i.e., pseudo-Hilbert ordering and multi-stage buffering, and asked students to first replicate the computational throughput (in GFLOPS) and bandwidth utilization (in GB/

TABLE 9
Machines on Azure Used for Reproducibility

Machine	CPU	Cores	GPU	Interconnect	Team(s)
NC24r Promo	Intel Xeon E5-2690 v3	2 x 12	4 x NVIDIA K80	N/A	Tsinghua University
NC24rs v2	Intel Xeon E5-2690 v4	2 x 12	4 x NVIDIA P100	ConnectX-3	ETH Zürich, Peking University, ShanghaiTech University, Tsinghua University
NC12s v2	Intel Xeon E5-2690 v4	1 x 12	2 x NVIDIA P100	N/A	Clemson, University of Texas
NC24rs v3	Intel Xeon E5-2690 v4	2 x 12	4 x NVIDIA V100	ConnectX-3	Nanyang Technological University, ETH Zürich, Georgia Institute of Technology, University of California San Diego, Tsinghua University
HC44rs	Intel Platinum 8168	2 x 21	N/A	ConnectX-6	ETH Zürich, University of Texas, Peking University
F16s v2	AMD EPYC 7551	1 x 32	N/A	ConnectX-4	Tsinghua University
HB60rs	AMD EPYC 7551	2 x 32	N/A	ConnectX-5	Nanyang Technological University, Tsinghua University
HB120rs v2	AMD EPYC 7V12	2 x 60	N/A	ConnectX-6	ETH Zürich, Georgia Institute of Technology, University of California San Diego, ShanghaiTech University, Tsinghua University

s) in Fig. 9 on a single CPU and single GPU. Then, we asked students to observe strong scaling from single node to four nodes.

6.4.1 Single-CPU Results

From sampling the student reports, we see that one of the challenges in reproducing the results is the resource utilization in the cloud that inherently includes virtualization overhead. Further, the cloud resources are much more limited, both in terms of computational throughput and bandwidth, compared to the (dedicated) high-end resources used in the original experiments. Nevertheless, the students seem to be able to verify that the application is indeed memory bound and the execution time is roughly proportional to the memory bandwidth. The top teams report that MemXCT can achieve high CPU utilization even though it is memory-bound, an important conclusion of the original experiments.

6.4.2 Single-GPU Results

In Fig. 13, we compare the top teams' results against the ones reported in the original paper. Since the teams didn't have access to KNL resources, we only show the GPU results. Specifically, the teams had access to the same generations of GPUs, i.e., K80, P100, and V100, which were used in the original paper. Because of the limited cloud budget given to participating teams, not all teams reproduced results on all generations of GPUs except Tsinghua (the overall winner) and Clemson. Nevertheless, the top teams were able to reproduce the compute throughput and memory bandwidth results within the normal variation of experimental results with their available GPUs. This does not only confirm the integrity of the original measurements but also the stability of GPU execution speed in different deployment environments, i.e., supercomputers versus cloud clusters.

6.4.3 Scaling Results

As for scaling, since the teams didn't have access to large-scale supercomputers, such as Theta or Blue Waters, they scaled MemXCT only up to 4 nodes on the Azure cloud. Still, the majority of the teams seemed to reproduce strong-scaling results with the small number of nodes, and presented a similar trend shown in Fig. 11d. Since cloud resources provide weaker network resources compared to supercomputers, it was interesting to see that the scaling trends still hold. This suggests that the MemXCT application can be used in a modest-sized cloud computing cluster and still achieve desired scaling behavior.

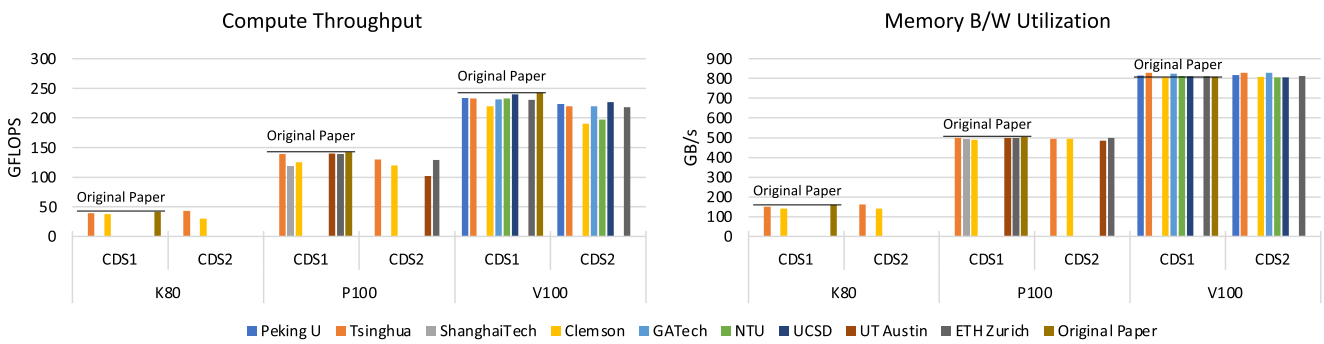


Fig. 13. Reproduction of GPU throughput (in GFLOPS) and bandwidth utilization (in GB/s) through CDS1 (equivalent to ADS2) and CDS2 datasets. The reproduced benchmarkings on three generations of GPUs highly matches with the original results presented in Fig. 9c.

A specific performance trend that was not reproduced is the super-linear strong scaling as in Fig. 11c because it is specific to KNL architecture. That is, MemXCT performance is bounded by memory bandwidth and strong scaling have super-linear speedup property because when each partition fits into the 16-GB high-bandwidth MCDRAM on KNL, we observe an extra speedup. However, CPUs and GPUs that were available in the Azure cloud did not have high-bandwidth memory that would cache each partition, and therefore student teams could not report any super-linear strong scaling.

6.5 Further Insights on Reproducibility

In this section, we discuss a few challenges on different aspects of reproducibility of MemXCT and other applications in general.

6.5.1 Verfication

Each application may require a different reproducibility approach. This mainly comes from the nature of the application, rather than the underlying computational workload. In MemXCT case, for example, the application reconstructs images, so the output is already visual and hence easy to verify by just looking at the screen, in a very specific way for MemXCT. The computational workload of the image reconstruction, however, is exactly the same as of any other iterative solution of a sparse linear system. Except, the technique proposed in this paper is highly optimized for the governing X-ray imaging problem. For verification, we initially thought about defining a distance metric between two images (or using an already-defined metric). However, a single number hardly provides any insight to the students about the correctness of the MemXCT system. Hence, we asked students to observe the iterative convergence of the result and visually verify its correctness by comparing it with the provided ground truth.

6.5.2 System Variations and Scaling

An important challenge experienced by the SCC teams was to reproduce the results when the systems used by the teams are different from the ones used for the original experiments. Hardware variations are obvious. The teams were not able to match the numerical execution speed and memory bandwidth results for CPUs simply because the hardware they used were much weaker than the CPUs used in the original experiments. However, they were able to reproduce similar ratios and trends. In contrast, when the teams used one or more of the GPUs in the original experiments, their results matched closely. A more subtle problem is software variations. Different OS, library versions, and virtualization schemes can affect the achievable execution speed in a significant way.

In the area of scaling, we have scaled the MemXCT application up to 4096 nodes of Blue Waters and Theta. The scale of these systems (described in Table 2) are not available for reproducibility as mentioned before, and the teams attempted to reproduce scaling trends only with 4 nodes (low-node-count regions of Fig. 11.) The MemXCT application reports the times spent for partial matrix multiplication, communication, and reduction shown as A_p, C, and R, respectively, in Fig. 11. Times required for these operations show different properties when the number of nodes is increased. For example, while computation (A_p) efficiently scales with an increasing number of nodes, communication (C) does not. Most of the teams are able to explain these performance properties. Several teams were able to observe the same strong-scaling trend in spite of the weaker networks in the cloud environment. This is likely due to the fact that in these low-node-count settings, the amount of work assigned to each node is still sufficient to hide the communication overhead in spite of the weaker networks in the cloud environment.

6.5.3 Benchmarking Across Systems

Comparing performance across systems, e.g., CPU and GPU, is another challenge for benchmarking and reproducibility. First, the fundamental differences of CPU and GPU architecture make them incomparable. That is, GPUs are extremely good when applications show data parallelism. For example, students report up to $10\times$ speedup of a single GPU over a single CPU. The reported KNL performance in this paper, however, has a comparable performance with GPUs, as shown in Fig. 9. Second, CPUs have much more memory on the host compared to the device memory on GPU. This sometimes prevents one-to-one comparison because a large benchmark does not fit into a single GPU. We solve this problem by benchmarking over four artificial datasets with various sizes, among which ADS1 and ADS2 fit into device memories of both GPU and KNL. Lastly, since GPUs (and other accelerators) have smaller device memory, strong scaling on the whole system may not be feasible when starting from a single device. Mainly because the single-device workload is extremely small when scaled to thousands of devices. That's why we use a large RDS2 benchmark (with 10.2 TB memory footprint) on 128 KNLs as the starting point for strong scaling on Theta. Similarly, we use a smaller RDS1 benchmark (112 GB memory footprint) on 32 K20x as the starting point for strong scaling on Blue Waters. In both cases, we scale up to 4096 nodes of both systems, as shown in Figs. 11c and 11d. The figure clearly contrasts two systems by their super-linear and sub-linear scaling properties.

In general, we can obviously not expect the teams to reproduce the same performance results when they use different hardware/software systems. However, we did expect that their results show similar characteristics and trends to the original paper. In this respect, the top teams were indeed able to observe the desired application performance behavior in their test systems. This gives us confidence that future users of MemXCT will experience these desired performance behavior in their own systems.

7 Conclusion

We have described a memory-centric approach to iterative XCT reconstruction. Our implementation, MemXCT, performs forward projection and backprojection operations as explicit SpMVs with no on-the-fly (redundant) computations or race conditions; these features allow MemXCT to achieve up to $50\times$ speedup over a high-performance compute-centric approach. Although MemXCT has a high

memory complexity, its per-node memory footprint decreases linearly with increasing number of nodes, which favors large-scale resources. For solving the performance bottlenecks, we propose and implement two-level pseudo-Hilbert ordering and multi-stage input buffering techniques in MemXCT. We compare the performance of our computational kernels on KNL and several generations of GPUs, and demonstrate scaling up to 4096 nodes using ALCF Theta (KNL) and NCSA Blue Waters (GPU) systems. We show that MemXCT reconstructs a large-scale ($11K \times 11K$) mouse brain tomogram in 10 seconds using 4096 KNL nodes. Our experimental results were reproduced successfully by the 2020 Student Cluster Competition teams. We present the pertinent features of the MemXCT artifact and collateral materials that enabled teams to verify their results and compare against our results within a strict time limit. Even though the reproducibility approach we take is specific to MemXCT, similar reproducibility design can be adopted by future HPC research artifacts to allow other researchers to reproduce the reported experimental results.

Acknowledgments

The authors would like to thank Stephen Harrell and Scott Michael for their invaluable guidance through the entire Reproducibility Challenge as well as the development of this article. We gratefully acknowledge the computing resources provided and operated by the Argonne Leadership Computing Facility, which is a U.S. Department of Energy, Office of Science User Facility. We thank Vincent De Andrade from Advanced Photon Source for sharing the mouse brain dataset. This research is part of the Blue Waters sustained-petascale computing project, which was supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications. This research was based in part upon work supported by: The Center for Applications Driving Architectures (ADA), a JUMP Center co-sponsored by SRC and DARPA. This material was based in part upon work supported by the XPACC Center for Exascale Simulation of Plasma-Coupled Combustion and Department of Energy, under Award Number DE-NA0002374.

References

[1] M. Hidayetoğlu *et al.*, "MemXCT: Memory-centric X-ray CT reconstruction with massive parallelization," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2019, Art. no. 85.

[2] APS Science 2018, Advanced Photon Source, Argonne Nat. Lab., Lemont, IL, USA, Tech. Rep. ANL-18/40, p. 15, Jan. 2019, ISSN 1931–5007.

[3] E. L. Dyer *et al.*, "Quantifying mesoscale neuroanatomy using X-ray microtomography," *eNeuro*, vol. 4, 2017, Art. no. ENEURO–0195.

[4] E. Y. Sidky, C.-M. Kao, and X. Pan, "Accurate image reconstruction from few-views and limited-angle data in divergent-beam CT," *J. X-Ray Sci. Technol.*, vol. 14, no. 2, pp. 119–139, 2006.

[5] X. Han *et al.*, "Algorithm-enabled low-dose micro-CT imaging," *IEEE Trans. Med. Imag.*, vol. 30, no. 3, pp. 606–620, Mar. 2011.

[6] S. V. Venkatakrishnan, C. A. Bouman, and B. Wohlberg, "Plug-and-play priors for model based reconstruction," in *Proc. IEEE Global Conf. Signal Inf. Process.*, 2013, pp. 945–948.

[7] K. A. Mohan, S. V. Venkatakrishnan, L. F. Drummy, J. Simmons, D. Y. Parkinson, and C. A. Bouman, "Model-based iterative reconstruction for synchrotron X-ray tomography," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2014, pp. 6909–6913.

[8] D. J. Duke *et al.*, "Time-resolved X-ray tomography of gasoline direct injection sprays," *SAE Int. J. Engines*, vol. 9, no. 1, pp. 143–153, 2016.

[9] D. Gürsoy, T. Biçer, A. Lanzirotti, M. G. Newville, and F. D. Carlo, "Hyperspectral image reconstruction for X-ray fluorescence tomography," *Opt. Exp.*, vol. 23, pp. 9014–9023, Apr. 2015.

[10] D. Gürsoy, T. Biçer, J. D. Almer, R. Kettimuthu, S. R. Stock, and F. De Carlo , "Maximum a posteriori estimation of crystallographic phases in X-Ray diffraction tomography," *Philosoph. Trans. Roy. Soc. London A: Math. Phys. Eng. Sci.*, vol. 373, no. 2043, 2015, Art. no. 20140392.

[11] T. Bicer *et al.*, "Trace: A high-throughput tomographic reconstruction engine for large-scale datasets," *Adv. Structural Chem. Imag.*, vol. 3, Jan. 2017, Art. no. 6.

[12] T. Bicer, D. Gursoy, R. Kettimuthu, F. De Carlo , G. Agrawal, and I. T. Foster, "Rapid tomographic image reconstruction via large-scale parallelization," in *Proc. 21st Int. Conf. Parallel Distrib. Comput.*, 2015, pp. 289–302.

[13] W. van Aarle *et al.*, "The ASTRA Toolbox: A platform for advanced algorithm development in electron tomography," *Ultramicroscopy*, vol. 157, pp. 35–47, 2015.

[14] D. Gürsoy, F. De Carlo , X. Xiao, and C. Jacobsen, "TomoPy: A framework for the analysis of synchrotron tomographic data," *J. Synchrotron Radiat.*, vol. 21, pp. 1188–1193, Sep. 2014.

[15] A. Beer, "Bestimmung der absorption des rothen Lichts in farbigen flussigkeiten," *Ann. Physik*, vol. 162, pp. 78–88, 1852.

[16] R. L. Siddon, "Fast calculation of the exact radiological path for a three-dimensional ct array," *Med. Phys.*, vol. 12, no. 2, pp. 252–255, 1985.

[17] X. Li *et al.*, "cuMBIR: An efficient framework for low-dose X-ray CT image reconstruction on GPUs," in *Proc. Int. Conf. Supercomput.*, 2018, pp. 184–194.

[18] S. G. D. Gonzalo , S. Hammond, S. Huang, O. Mutlu, J. Gómez-Luna, and W. Hwu, "Automatic generation of warp-level primitives and atomic instructions for fast and portable parallel reduction on GPUs," *IEEE/ACM Int. Symp. Code Gener. Optim.*, pp. 73–84, 2019, doi: 10.1109/CGO.2019.8661187.

[19] P. M. Campbell, K. D. Devine, J. E. Flaherty, L. G. Gervasio, and J. D. Teresco, "Dynamic octree load balancing using space-filling curves," Williams College Dept. Comput. Sci., Tech. Rep. CS-03-01, 2003.

[20] M. Parashar and J. C. Browne, "On partitioning dynamic adaptive grid hierarchies," in *Proc. 29th Hawaii Int. Conf. Syst. Sci.*, 1996, pp. 604–613.

[21] J. Zhang, S.-I. Kamata, and Y. Ueshige, "A pseudo-hilbert scan algorithm for arbitrarily-sized rectangle region," in *Proc. Int. Workshop Intell. Comput. Pattern Anal. Synthesis*, 2006, pp. 290–299.

[22] Y. Wang *et al.*, "A high-throughput X-ray microtomography system at the advanced photon source," *Rev. Sci. Instrum.*, vol. 72, no. 4, pp. 2062–2068, 2001.

[23] H. Wang, W. Liu, K. Hou, and W.-C. Feng, "Parallel transposition of sparse data structures," in *Proc. Int. Conf. Supercomput.*, 2016, Art. no. 33.

[24] X. Wang, A. Sabne, S. Kisner, A. Raghunathan, C. Bouman, and S. Midkiff, "High performance model based image reconstruction," *ACM SIGPLAN Notices*, vol. 51, 2016, Art. no. 2.

[25] R.Barrett *et al.*, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, vol. 43. Philadelphia, PA, USA: SIAM, 1994.

[26] *ALCF*, "Theta | Argonne leadership computing facility." Accessed: Jan. 16, 2019. [Online]. Available: https://www.alcf.anl.gov/theta

[27] *NCSA*, "Blue waters user portal | home." Accessed: Jan. 16, 2019. [Online]. Available: https://bluewaters.ncsa.illinois.edu/

[28] *ALCF*, "Cooley | Argonne leadership computing facility." Accessed: Jan. 16, 2019. [Online]. Available: https://www.alcf.anl.gov/user-guides/cooley

[29] *IBM*, "IBM power system S822LC for high performance computing - overview." Accessed: Mar. 20, 2019. [Online]. Available: https://www.ibm.com/us-en/marketplace/high-performance-computing

[30] *Nvidia*, "Deep learning server for AI research | NVIDIA CGX-1." Accessed: Mar. 20, 2019. [Online]. Available: https://www.nvidia.com/en-us/data-center/dgx-1/

[31] *OLCF*, "Titan user guide - Oak ridge leadership computing facility." Accessed: Jan. 16, 2019. [Online]. Available: https://www.olcf.ornl.gov/for-users/system-user-guides/titan/titan-user-guide/#nvidia-k20x-gpus

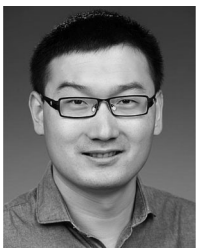
[32] W. Kanitpanyacharoen *et al.*, "A comparative study of X-ray tomographic microscopy on shales at different synchrotron facilities: ALS, APS and SLS," *J. Synchrotron Radiat.*, vol. 20, no. 1, pp. 172–180, 2013.

[33] F. De Carlo *et al.*, "TomoBank: A tomographic data repository for computational X-Ray science," *Meas. Sci. Technol.*, vol. 29, no. 3, 2018, Art. no. 034004.
[34] I. B. Peng, R. Gioiosa, G. Kestor, P. Cicotti, E. Laure, and S. Markidis, "Exploring the performance benefit of hybrid memory system on HPC environments," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops*, 2017, pp. 683–692.
[35] Z. Jia, M. Maggioni, B. Staiger, and D. P. Scarpazza, "Dissecting the NVIDIA Volta GPU architecture via microbenchmarking," 2018. [Online]. Available: https://arxiv.org/abs/1804.06826
[36] *Nvidia*, "CUSPARSE Library." Accessed: Jan. 16, 2019. [Online]. Available: https://developer.download.nvidia.com/compute/DevZone/docs/html/CUDALibraries/doc/CUSPARSE_Library.pdf
[37] M. Beister, D. Kolditz, and W. A. Kalender, "Iterative reconstruction methods in X-Ray CT," *Physica Medica*, vol. 28, no. 2, pp. 94–108, 2012.
[38] J. Hsieh, B. Nett, Z. Yu, K. Sauer, J.-B. Thibault, and C. A. Bouman, "Recent advances in CT image reconstruction," *Current Radiol. Rep.*, vol. 1, no. 1, pp. 39–51, 2013.
[39] J. Amanatides and A. Woo, "A fast voxel traversal algorithm for ray tracing," in *Proc. Eur. Comput. Graph. Conf. Exhib.*, 1987, pp. 3–10.
[40] J. Agulleiro and J.-J. Fernandez, "Fast tomographic reconstruction on multicore computers," *Bioinformatics*, vol. 27, no. 4, pp. 582–583, 2011.
[41] J. Treibig, G. Hager, H. G. Hofmann, J. Hornegger, and G. Wellein, "Pushing the limits for medical image reconstruction on recent standard multicore processors," *Int. J. High Perform. Comput. Appl.*, vol. 27, pp. 162–177, 2012.
[42] C. Johnson and A. Sofer, "A data-parallel algorithm for iterative tomographic image reconstruction," in *Proc. 7th Symp. Front. Massively Parallel Comput.*, 1999, pp. 126–137.
[43] M. D. Jones, R. Yao, and C. P. Bhole, "Hybrid MPI-OpenMP programming for parallel OSEM PET reconstruction," *IEEE Trans. Nucl. Sci.*, vol. 53, no. 5, pp. 2752–2758, Oct. 2006.
[44] X. Wang, A. Sabne, P. Sakdhnagool, S. J. Kisner, C. A. Bouman, and S. P. Midkiff, "Massively parallel 3D image reconstruction," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2017, Art. no. 3.
[45] J. Qi and R. M. Leahy, "Iterative reconstruction techniques in emission computed tomography," *Phys. Med. Biol.*, vol. 51, no. 15, 2006, Art. no. R541.
[46] C.-Y. Chou, Y.-Y. Chuo, Y. Hung, and W. Wang, "A fast forward projection using multithreads for multirays on GPUs in medical image reconstruction," *Med. Phys.*, vol. 38, no. 7, pp. 4052–4065, 2011.
[47] G. Pratx, G. Chinn, P. D. Olcott, and C. S. Levin, "Fast, accurate and shift-varying line projections for iterative reconstruction using the GPU," *IEEE Trans. Med. Imag.*, vol. 28, no. 3, pp. 435–445, Mar. 2009.
[48] D. Lee, I. Dinov, B. Dong, B. Gutman, I. Yanovsky, and A. W. Toga, "CUDA optimization strategies for compute-and memory-bound neuroimaging algorithms," *Comput. Methods Programs Biomed.*, vol. 106, no. 3, pp. 175–187, 2012.
[49] B. Jang, D. Kaeli, S. Do, and H. Pien, "Multi GPU implementation of iterative tomographic reconstruction algorithms," in *Proc. IEEE Int. Symp. Biomed. Imag.: Nano Macro*, 2009, pp. 185–188.
[50] A. Sabne, X. Wang, S. J. Kisner, C. A. Bouman, A. Raghunathan, and S. P. Midkiff, "Model-based iterative CT image reconstruction on GPUs," in *Proc. 22nd ACM SIGPLAN Symp. Princ. Practice Parallel Program.*, 2017, pp. 207–220.
[51] S. S. Stone *et al.*, "Accelerating advanced MRI reconstructions on GPUs," *J. Parallel Distrib. Comput.*, vol. 68, no. 10, pp. 1307–1318, 2008.
[52] F. Xu and K. Mueller, "Accelerating popular tomographic reconstruction algorithms on commodity PC graphics hardware," *IEEE Trans. Nucl. Sci.*, vol. 52, no. 3, pp. 654–663, Jun. 2005.
[53] T. B. Jablin, P. Prabhu, J. A. Jablin, N. P. Johnson, S. R. Beard, and D. I. August, "Automatic CPU-GPU communication management and optimization," *ACM SIGPLAN Notices*, vol. 46, pp. 142–151, 2011.
[54] M. Hidayetoğlu *et al.*, "Petascale XCT: 3D image reconstruction with hierarchical communications on multi-GPU nodes," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2020, pp. 1–13.
[55] G. Peano, "Sur une courbe, qui remplit toute une aire plane," *Mathematische Annalen*, vol. 36, no. 1, pp. 157–160, 1890.
[56] M. Bader, *Space-Filling Curves: An Introduction With Applications in Scientific Computing*, vol. 9. Berlin, Germany: Springer, 2012.
[57] F. Günther, M. Mehl, M. Pögl, and C. Zenger, "A cache-aware algorithm for PDEs on hierarchical data structures based on space-filling curves," *SIAM J. Sci. Comput.*, vol. 28, no. 5, pp. 1634–1650, 2006.
[58] H. Sagan, *Space-Filling Curves*. Berlin, Germany: Springer, 2012.
[59] M. Bader and C. Zenger, "Cache oblivious matrix multiplication using an element ordering based on a Peano curve," *Linear Algebra Appl.*, vol. 417, no. 2/3, pp. 301–313, 2006.
[60] Witten and Neal, "Using Peano curves for bilevel display of continuous-tone images," *IEEE Comput. Graphics Appl.*, vol. 2, no. 3, pp. 47–52, May 1982.
[61] J. K. Lawder and P. J. H. King, "Querying multi-dimensional data indexed using the hilbert space-filling curve," *ACM SIGMOD Rec.*, vol. 30, no. 1, pp. 19–24, 2001.
[62] N. Reissman, J. C. Meyer, and M. Jahre, "A study of energy and locality effects using space-filling curves," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops*, 2014, pp. 815–822.
[63] J. Mellor-Crummey, D. Whalley, and K. Kennedy, "Improving memory hierarchy performance for irregular applications using data and computation reorderings," *Int. J. Parallel Program.*, vol. 29, no. 3, pp. 217–247, 2001.
[64] D. Hilbert, *Über die Stetige Abbildung einer Linie auf ein Flächenstück*. Berlin, Germany: Springer, 1935, pp. 1–2.
[65] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz, "Analysis of the clustering properties of the hilbert space-filling curve," *IEEE Trans. Knowl. Data Eng.*, vol. 13, no. 1, pp. 124–141, Jan./Feb. 2001.
[66] G. M. Morton, "A computer oriented geodetic data base and a new technique in file sequencing," 1966. [Online]. Available: https://dominoweb.draco.res.ibm.com/reports/Morton1966.pdf
[67] M. A. Heroux and C. K. Garrett, "Special issue on SC16 student cluster competition reproducibility initiative," *Parallel Comput.*, vol. 70, pp. 3–4, 2017. [Online]. Available: https://doi.org/10.1016/j.parco.2017.10.002
[68] J. Schindelin *et al.*, "Fiji: An open-source platform for biological-image analysis," *Nat. Methods*, vol. 9, no. 7, pp. 676–682, 2012.

Mert Hidayetoğlu is currently working toward the PhD degree at the University of Illinois at Urbana-Champaign, Champaign, Illinois with research at the intersection of large-scale applications, high-performance computing, and software systems. His research interests include unstructured data accesses, communications, and computations on supercomputers with multi-GPU node architecture. He was a Givens Fellow with Argonne National Laboratory and is currently a member of IBM-Illinois Center for Cognitive Computing Research. He is a recipient of the SC20 Best Paper Award and HPEC'20 Graph Challenge Champion Award, and 2021 ACM-IEEE CS George Michael Memorial HPC Fellowship.

Tekin Biçer (Senior Member, IEEE) received the the master's and PhD degrees from Computer Science and Engineering Department, Ohio State University, Columbus, Ohio. He is currently a computer scientist with Data Science and Learning Division, Argonne National Laboratory. He also holds joint appointments in the X-Ray Science Division at Advanced Photon Source and Consortium for Advanced Science and Engineering, University of Chicago. His research interests include large-scale, high-performance, parallel and distributed computing systems. He is a senior member of the ACM.

Simon Garcia de Gonzalo received the PhD degree from Computer Science Department, University of Illinois at Urbana-Champaign, Champaign, Illinois. He is currently a postdoctoral researcher and STARS 2020 Fellow with Barcelona Supercomputing Center (BSC), Computer Sciences Department. He works within the Accelerators and Communications for HPC Team. His research interests include area of performance portability, runtime systems, and programming models for high-performance computing.



Bin Ren received the PhD degree from the Department of Computer Science and Engineering, Ohio State University, Columbus, Ohio. He is currently an assistant professor with the Department of Computer Science, College of William & Mary. His research interests include compilers, parallel computing, software systems, real-time machine learning, and machine learning systems. He has published more than 50 peer-reviewed papers in leading venues related to these areas, including ASPLOS, PLDI, PPoPP, SC, ICML, AAAI, and *IEEE Transactions on Parallel and Distributed Systems*. He is the recipient of NSF CAREER Award, Jeffress Trust Award, CGO'13 and SC'20 Best Paper Awards, and the first place in Design Contest of ISLPED'20.

Doğa Gürsoy received the BS and MS degrees from Middle East Technical University, Ankara, Turkey, in 2004 and 2007, respectively, and the PhD degree in electrical engineering from the Graz University of Technology, Graz, Austria, in 2010. He is currently a computational scientist with the Advanced Photon Source at Argonne National Laboratory. He is also an adjunct associate professor with the Department of Electrical Engineering and Computer Science, Northwestern University, and a fellow of Northwestern Argonne Institute of Science and Engineering. His research interests include developing computational methods for solving inverse problems in imaging and applying them to interesting problems in biological and materials sciences.

Rajkumar Kettimuthu (Senior Member, IEEE) is currently a computer scientist with Argonne National Laboratory, a senior scientist with the University of Chicago and a senior fellow with the Northwestern-Argonne Institute of Science and Engineering. His research interests include intelligent compute and data infrastructure for science, advanced wired and wireless communications for science, and quantum networks. He has coauthored more than 125 peer-reviewed articles, most of which appeared in premier journals and conferences in computer science, and several of which won best paper awards. His work has featured in more than ten news articles. He has participated in more than 25 DOE and NSF workshops to set strategy and direction for networking and distributed computing research. He is a recipient of the prestigious R&D 100 award. He is a distinguished member of the ACM.

Ian T. Foster (Fellow, IEEE) received the BSc (hons I) degree in computer science from the University of Canterbury, Christchurch, New Zealand, and the PhD degree in computer science from Imperial College, London, U.K. He is currently the director of Argonne's Data Science and Learning Division, Argonne senior scientist and distinguished fellow and the Arthur Holly Compton distinguished service professor of computer science with the University of Chicago. His research interests include high-performance computing, distributed systems, and data-driven discovery. He has published hundreds of scientific papers and eight books on these and other topics. He is an elected fellow of the AAAS and ACM.

Wen-Mei W. Hwu (Fellow, IEEE) is currently a senior distinguished research scientist with NVIDIA. Prior to NVIDIA, he was a professor, Sanders-AMD endowed chair, acting department head and chief scientist of the Parallel Computing Institute, University of Illinois at Urbana-Champaign. He and his Illinois team developed the superblock compiler scheduling and optimization framework that has been adopted by virtually all modern vendor and open-source compilers today. For his research contributions, he received the ACM SigArch Maurice Wilkes Award, ACM Grace Murray Hopper Award, IEEE Computer Society Charles Babbage Award, ISCA Influential Paper Award, MICRO Test-of-Time Award, IEEE Computer Society B. R. Rau Award, CGO Test-of-Time Award and Distinguished Alumni Award in CS of the University of California, Berkeley. He has also won numerous best paper awards for major conferences. He is a fellow of the ACM.