Reproducibility: Performance Evaluation of MemXCT on Azure CycleCloud Platform

Yuchen Liu, Yixuan Meng, Kaiyuan Xu, Zijun Xu, Tianyuan Wu, Yiwei Yang, and Shu Yin

Abstract—Memory-Centric X-ray Computational Tomography(CT) is an iterative reconstruction technique that trades compute simplifications with higher memory accesses. MemXCT implements a sparse matrix-vector multiplication(SpMV) with multi-stage buffering and two-level pseudo-Hilbert ordering for optimization. Motivated by the need to validate conclusions from previous work, we reproduce the numerical results, the algorithm's performance, and the scaling behavior of the algorithms as the number of MPI processes increases on Azure. Digital artifacts from these experiments are available at: 10.5281/zenodo.5598108

Index Terms—SpStudent cluster challenge, reproducibility, scalability, MemXCT

1 Introduction

As computational models and software play increasingly important roles in scientific research, reproducibility is of utmost importance to validate and verify the results of articles. MemXCT proposes a memory-centric algorithm for X-ray computed tomography, avoiding redundant computations in the conventional method. The MemXCT system optimizes data communication, partitioning, and accesses with an improved implementation of sparse matrix-vector multiplication, and demonstrated the changes in performance metrics with multi-stage buffering and two-level pseudo-Hilbert ordering.

Using a limited budget, we aimed to compare the performances on different hardware, such as single CPU and GPU, and its scalability using multiple datasets.

The remainder of this paper is organized as follows: Section 2 summarizes the contributions of the MemXCT paper and numeric results for the evaluation of the MemXCT system's performance. Section 3 describes our experimental methodology. Section 4 demonstrates the planned experimental run. Section 5 discusses our strategy of tuning buffer sizes and block sizes. Section 6 compares the single CPU and single GPU performance with given datasets. Section 7 discusses the results and scalability of MemXCT with strong scaling. Section 7 contains our conclusion.

2 Background

Memory-Centric X-ray Computational Tomography(CT) Reconstruction [1] is a 3D imaging technique to reconstruct tomograms in high resolution. MemXCT utilizes multi-stage buffering and two-level pseudo-Hilbert ordering to implement an optimized sparse matrix-vector multiplication(SpMV). With the efficient SpMV, the performance of computations is improved, making memory the performance bottleneck. As two compute-Intensive kernels in iterative reconstruction, forward and backward projections are sped up using MPI parallelization in the MemXCT system. The forward and backward projections can be split into three steps: multiplication costs A_p, the communication C, and reduction R.

- *The authors are with the School of Information Science and Technology, ShanghaiTech University, Shanghai 201210, China. E-mail: {liuyc1, mengyx, xuky, xuzj, wuty, yangyw, yinshu}@shanghaitech.edu.cn.*

Manuscript received 16 Mar. 2021; revised 13 July 2021; accepted 8 Nov. 2021 Date of publication 11 Nov. 2021; date of current version 24 Jan. 2022.
(Corresponding author: Shu Yin.)
Recommended for acceptance by S.L. Harrell and S.A. Michael.
Digital Object Identifier no. 10.1109/TPDS.2021.3127450

The MemXCT paper carried out an experimental demonstration of the performance gain due to the optimizations above by reporting the GFLOPS metrics in single devices. The author also performed weak and strong scaling experiments and gave the A_p, C, and R kernel times as well as the total reconstruction times.

3 Experimental Setup

To validate the performance of this system, we utilized a cluster with the specification provided in Table 1.

3.1 Hardware Selection

Our experimental cluster was comprised of eight nodes. Each 60 virtual core AMD EPYC 7551 processors. With these processors, we can use up to 126 GB/s memory bandwidth. Azure launched the cloud product with the latest Zen-2 AMD Rome series processors [2], which is cost-efficient and can endure strenuous calculation. We still need high bandwidth of message passing by choosing SR-IOV and RDMA hardware. To enable them, we added a custom image and modified cloud-init to enable the fast allocation of instances. Each node hosts an installation of CentOS 8.0. For node-to-node connectivity, we use Mellanox Infiniband HDR to ensure efficient MPI communication between nodes. For the HB60rs model, the first four cores on one socket are hypervisor reserved, and Infinity Fabrics[3] transmit the data between the CPU sockets and the GPU ucx[4]. for GPU. In comparison, the original paper[1] utilized 4096 KNL nodes with 256K cores.

As for the storage, Azure provides three hierarchies: standard, premium, and ultra, which stands for basic HDDs, 900MB/s SSD and 1.45GB/s Nvme. For the storage node, we choose the XFS filesystem over EXT4 because of its consistency and high scalability.

3.2 Software Selection

As listed in Table 2, the cluster installs a complete development environment, including the version of our compiler and the visualization tool. We choose the intel-parallel-studio@cluster.2019.5 [5] to compile the code because GNU gcc and AOCC(AMD high-performance compiler) do not provide the peak performance that Intel does. In comparison, the Intel compiler has better knowledge of X86 architecture and do optimizations such as more aggressive Loop Unrolling. The option of the compiler makes a significant difference in speed up [6]. We edited the MakeFile script `MakeFile.theta` to adapt to our environment. We replaced the flag `-xmic-avx512` to `-xCORE-AVX2`, as the former flag is for Intel®KNL architecture[7]. In terms of compiler and linker, we prefer Intel®compiler suite (ICC) as it provides advanced optimizations towards the x86 architecture. Options including `-qopt-prefetch=0`, `-qopt-report=5`, and `-qopt-report-file=$@.optrpt` are removed as they are developed for Intel®Xeon®processors. Besides, the prefetch and branch prediction techniques in the Profiling Guided Optimization(PGO) do not perform as efficiently as the option `-O3`, we prefer `-O3` in the experiments. We then wrote scripts to run and analyze the scalability based on scaling results.

4 Description of Experimental Run

We organized experiments as follows. The MemXCT paper uses 62.5 cores per node on average, and we have 60 cores for each node. The KNL processors have a bottleneck introduced by shared memory. Therefore when the size of datasets exceeds the capacity of shared memory, there is a dramatic drop of GFLOPS with large buffer size and block size. However, such a bottleneck does not exist in our CPU architecture since our CPUs do not have shared memory, and the provided datasets can fit in CPU memory. Such

TABLE 1
Parameters of Hardware Components

Node	Computing node		Storage
Type	HB60rs	Nc24r_v2	DS4_v2
Processor	AMD Epyc 7551	Intel Xeon E5-2690	
vCPU spec	60 cores 2.35 GHz	24 cores 2.60 GHz	4 cores
Memory	384 GiB	224 GiB	28 GiB
Storage	768 GiB SSD	2.9 TiB SSD	1.0 TiB XFS
GPU Card	/	4 P100	/
SR-IOV Support	no	no	no
RDMA Support	100Gbps	56Gbps	/
Cost per hour	$2.51	$10.74	$0.51

TABLE 2
Software Specification

Host Compilers	Intel parallel studio 2019.05 icc 19.0.5.281 GNU RedHat gcc 8.3.1
Device Compilers	Cuda and CuDNN 11.1
Other Specs	Fiji 2.1.0

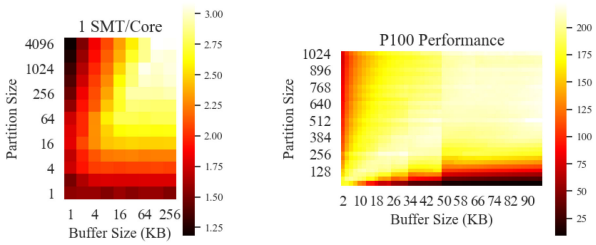

Fig. 1. Tune the parameters of buffer and block size.

property of the architecture predicts different scalability for strong scaling experiments.

Due to limited budget, we used Tesla P100 GPU, while the MemXCT paper used V100. The single-precision performance is 9.3 TeraFLOPS for Tesla P100, and 16.4 TeraFLOPS for V100. The GFLOPS and effective memory bandwidth of a single CPU and GPU are then tested and compared on different datasets. A single Tesla P100 is unsurprisingly much better than the performance on a single AMD EPYC 7551 processor.

5 Tuning Multi-Stage Input Buffering

The MemXCT paper improves first-level cache utilization using multi-stage buffering, thus performs an optimized sparse matrix-vector multiplication. Input buffer enables MemXCT to reuse buffered data and save memory bandwidth. With a larger block size, the per-block number of buffer stagings increases and brings overhead. Meanwhile, the memory footprint per block also increases and boosts the data reuse from the buffer. We tune the block size and the buffer size to improve the performance of the CPU and GPU.

We tuned the input buffering parameters on both the EPYC and GPU architectures to find out which block size and buffer size would provide good single EPYC and single GPU performance. We write scripts to parse buffer size and block size. As shown in Fig. 1, the performance is improved when the block size is set to 1024 for ADS1 to ADS2. The committee didn't require to tune the ADS3 for a single GPU setup. We made our attempts to optimize the parameters on ADS3 but failed, majorly because the size of ADS3 is too large to fit into a single GPU memory.

For instance, according to the original paper [1], the performance gets worse when the buffer size or the partition is set too small. On AMD EPYC 7551, the GFLOPS performance reaches the peak for ADS1 with 1 SMT/core and buffer size of 64. Similarly, we apply the same tuning strategy for GPU, and a high GPU performance of GFLOPS is observed when the block size is 256.

TABLE 3
Comparison of GFLOPS and Memory B/W Performance on Single Device

	CPU	GPU
GFLOPS on ADS1	36.6470	126.5434
Mem B/W on ADS1	7.3655 GB/s	487.2815 GB/s
GFLOPS on ADS2	33.6689	119.3763
Mem B/W on ADS2	6.8672 GB/s	491.2889 GB/s

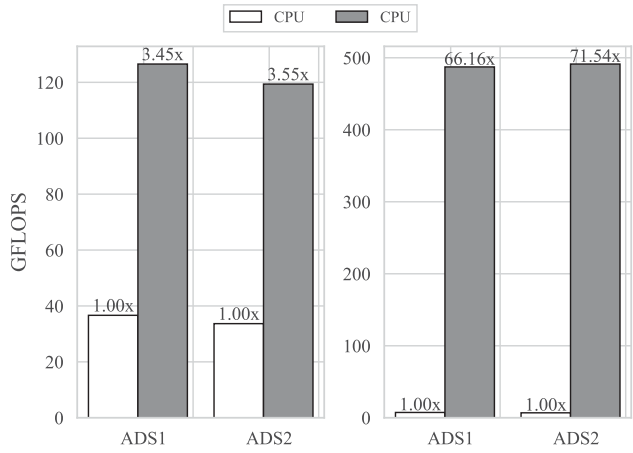

Fig. 2. Test and compare GFLOPS and effective memory bandwidth on single CPU and GPU.

6 Single CPU Versus Single GPU Performance Comparison

To test the GFLOPS performance and effective memory bandwidth on single devices (CPU and GPU), we ran ADS1 and ADS2 on all 60 cores of an AMD EPYC 7551 and a Tesla P100. We choose the block size and buffer size according to tuned parameters to obtain the best performance of them. As for the GFLOPS and memory bandwidth of a single CPU and GPU, the metrics of performance are summarized in Table 3. The magnifications of GFLOPS and memory bandwidth on GPU comparing to CPU are shown in Fig. 2, suggesting a great improvement on both metrics on GPU rather than CPU.

7 Strong Scaling on CPUs

We reproduced the strong scaling experiment by reconstructing ADS1 and ADS2 samples. Fig. 3 shows the strong scaling property of the application while running the ADS1 and ADS2 datasets. During the experiment, the sizes of the datasets remain the same as we increase the number of processors. The maximum number of processors we used to run the datasets is 60 to let the dataset fit within the system. Our cluster showed good scaling with up to 30 processors for ADS1 and ADS2. However, the performance of the MemXCT system drops with 48 processors for both the ADS1 and the ADS2 datasets. Additionally, the MemXCT paper pointed out a

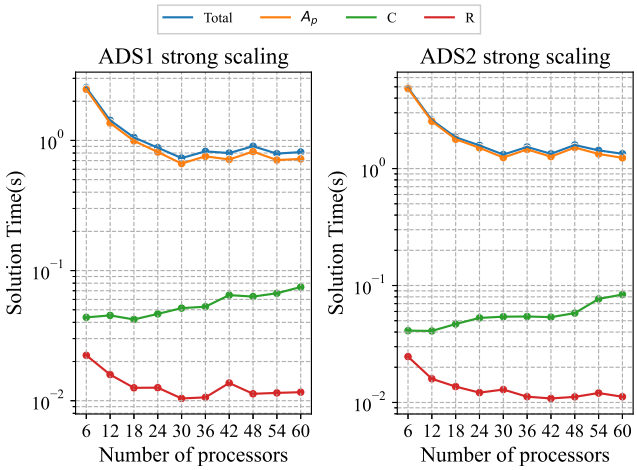

Fig. 3. Strong scaling on ADS1 and ADS2 datasets,increasing number of processors.

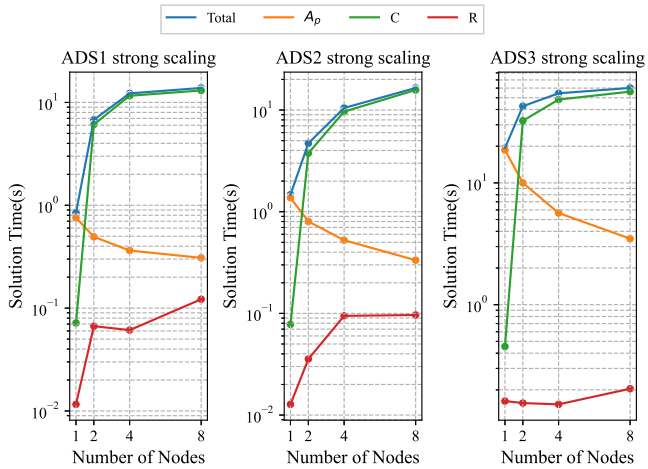

Fig. 4. Strong scaling on ADS1, ADS2 and ADS3 datasets, increasing number of nodes.

super-linear speedup of A_p kernel. This trend is not significant in our cluster after the number of processors increased to 30, probably due to the increasing communication overhead as the number of processors becomes larger.

We also performed strong scaling experiments with one, two, four, and eight nodes, each node is equipped with an AMD EPYC 7551 processor, while the MemXCT paper used up to 4096 nodes. The result for all three datasets is shown in Fig. 4. Fig. 4 demonstrates that the communication overhead grows dramatically, causing the increasing trend for the total solution time when the number of nodes changes from one to eight. Comparing to our strong scaling experiments on the single node, the super-linear speedup for the A_p kernel is more significant, especially in the ADS3 case.

8 Conclusion

In this work, we reproduced both single CPU and single GPU performance in the MemXCT paper. We also validated the scalability by conducting strong scaling experiments on single and multiple nodes, then compared the output on different datasets. Our experimental results present a similar scaling trend to that in the original MemXCT paper, hence confirm its conclusions.

Not only did we compare the GFLOPS and effective memory bandwidth between a single CPU and GPU, but also suggested a scaling property for all given datasets in a different environment with a limited computing budget, confirming the scalability of the MemXCT system.

Acknowledgments

The authors would like to thank our shepherds and the SC-21 reviewers for their constructive feedback, support from ShanghaiTech University, and our co-advisor Mr. Yingdong Zhang from the Library and Information Technology Center, ShanghaiTech University. This reproducibility experimental result was performed during the Student Cluster Competition SC-20 by team GeekPie_HPC at ShanghaiTech University. Yuchen Liu, Yixuan Meng, Kaiyuan Xu, Zijun Xu, Tianyuan Wu, Yiwei Yang contributed equally.

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, pp. 1–56.
[2] "AMD EPYC rome CPU spec," Accessed: Oct. 20, 2020. [Online]. Available: https://www.slideshare.net/AMD/isscc-2018-zeppelin-an-soc-for-multichip-architectures
[3] N. Beck, S. White, M. Paraschou, and S. Naffziger, "'Zeppelin': An SoC for multichip architectures," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2018, pp. 40–42.
[4] P. Shamis *et al.*, "UCX: An open source framework for HPC network APIs and beyond," in *Proc. IEEE 23rd Annu. Symp. High-Perform. Interconnects*, 2015, pp. 40–43.
[5] "Intel parallel studio XE *reg*; documentation," Accessed: Mar. 7, 2021. [Online]. Available: https://software.intel.com/content/www/us/en/develop/articles/get-started-guides-for-intel-parallel-studio-xe-2019.html
[6] C. Curtsinger and E. D. Berger, "Stabilizer: Statistically sound performance evaluation," *ACM SIGARCH Comput. Archit. News*, vol. 41, no. 1, pp. 219–228, 2013.