Improving GPU Scaling for X-Ray CT

Harini Muthukrishnan, Thomas F. Wenisch, Jeffrey A. Fessler

***Abstract*—Model-based iterative reconstruction (MBIR) for X-Ray CT is computationally expensive, yet highly parallelizable, making it amenable to multi-GPU implementation. However, reconstruction time does not improve linearly with the number of GPUs, mainly due to high inter-GPU copying delays at the end of computation phases. Overlapping copies with computation—by copying incrementally as data are produced—can mitigate copy overhead and improve performance scalability. This paper demonstrates how to perform asynchronous copies using GPU threads initiated via dynamic kernel launch. Our technique enables 90% of copy time to overlap with compute, achieving a speedup of 1.24× (nearing the theoretical bound of 1.31× with instantaneous copies) over conventional cudaMemcpy at the end of compute phases on four Tesla K40m GPUs. Relative to a baseline implementation on a single GPU, our accelerated approach achieves a speedup of 3.46× on four GPUs. We project even higher impact from our technique with more GPUs.**

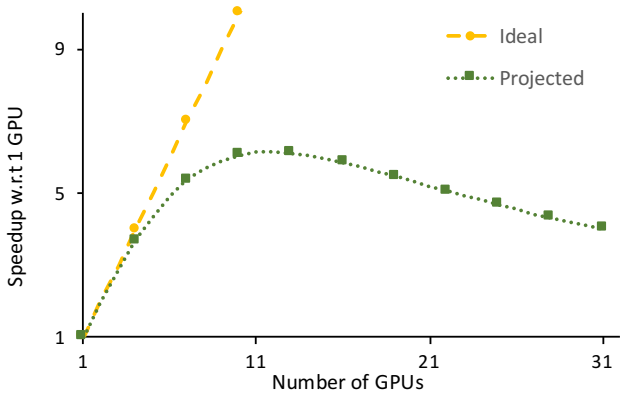

Fig. 1: Ideal and Projected speedups achievable with GPU scaling with conventional multi-GPU implementation.

I. Introduction

Model-based iterative reconstruction (MBIR) for X-Ray CT offers improved image quality at lower radiation doses than Filtered Back Projection (FBP) [1], but at higher computational costs. Although researchers have explored various acceleration techniques, such as using SIMD instructions on CPUs [2, 3] and cloud computing [4, 5], the compute times remain undesirably high for MBIR to be ubiquitous clinically.

Further MBIR acceleration requires increasing both computational resources and memory bandwidth, making a case for employing multiple GPUs [6, 7]. But GPU scaling does not always result in linear speedup. Fig. 1 shows projected speedups as we parallelize a state-of-the-art MBIR algorithm [8] over more GPUs. Using more GPUs initially provides near-linear speedup up to about eight GPUs, as the computational phases of MBIR can be readily partitioned across the GPUs. However, beyond eight GPUS, speedup saturates and then begins to decrease. This reversal arises due to the time spent copying data among GPUs between computational phases. Sinogram and image data must be exchanged all-to-all among the GPUs between phases, yet current systems offer no mechanism to broadcast data, requiring pairwise copies. As a result, even though computation time shrinks, copy time grows and ultimately dominates as the number of GPUs increases.

High copy time can be mitigated by overlapping copying with computation—by "streaming" data from producer GPUs as soon they become ready. By initiating some copies while computation is ongoing, the next computational phase must wait only for straggling data produced at the end of a phase.

While overlapping communication with compute between CPU and GPU has been studied for FBP[9], hiding copies underneath compute in multi-GPU systems entails several challenges: (1) State-of-the-art CPU–GPU systems provide an astonishing diversity of mechanisms to move data from one GPU's memory space to another. Data may be moved by CPU loads/stores through a variety of addressing mechanisms, by GPU loads/stores, by DMA engines integrated on the GPU, and potentially even by other devices on the PCIe bus. These alternatives trade off bandwidth, initiation latency, and disruption to other GPU threads in non-trivial ways; the best approach for our purpose is not obvious. (2) Every CT phase must indicate when enough data has been generated for a copy to start. It is neither clear how to trigger copies, nor at what granularity they should be performed. (3) The CUDA programming model allows enormous freedom in ordering the execution of individual threads. No existing programming interface allows GPU programs to efficiently track production of output data, initiate copies, and await copy completion.

This work focuses on accelerating a penalized weighted least-squares with ordered subsets (PWLS-OS) reconstruction algorithm [8] on a multi-GPU system by hiding copies under computation phases that generate data. We discuss performing copies using GPU threads initiated though dynamic kernel launch as the mechanism best suited for our purpose. We identify the best granularity at which to perform such copies and describe techniques to track data production and copy completion. We also consider how the subdivision of MBIR phases into individual GPU kernels affects the order data is generated and how to orchestrate these to maximize gains from our technique.

Using four Tesla K40m GPUs, we show that a 20-iteration helical CT reconstruction of a 512×512×512 image from 7256 views of size 888×64 takes 11.25 minutes using our copy mechanism. We also demonstrate, using a simple mathematical model, how our technique makes MBIR CT more amenable to further GPU scaling.

Supported in part by NIH Grant U01 EB018753. The authors are with the EECS Department of the University of Michigan, Ann Arbor, MI 48019 USA (email: {harinim, twenisch, fessler}@umich.edu).

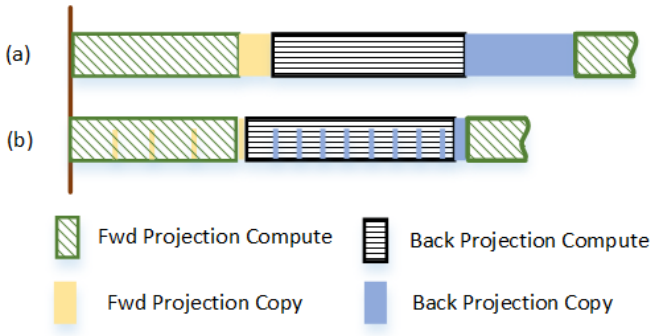

Fig. 2: (a) Conventional: computation and copies in series. (b) Our goal: incremental copying overlapping computation

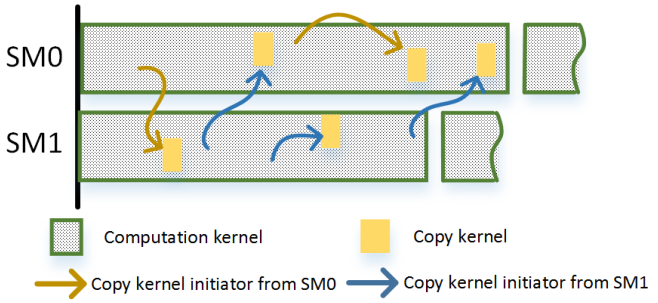

Fig. 3: GPU SM resource utilization and copy initiation. The launch of the initiated copy kernels depends entirely on the hardware scheduler.

II. Methods

A. Background

We reconstruct the image $\widehat{\mathbf{x}}$ by iteratively minimizing the PWLS cost function [5]:

$$\widehat{\mathbf{x}} = \arg\min_{\mathbf{x}\geq 0} \Psi(\mathbf{x}), \qquad \Psi(\mathbf{x}) = \frac{1}{2}\|\mathbf{A}\mathbf{x}-\mathbf{y}\|_{\mathbf{W}}^2 + \mathbf{R}(\mathbf{x}), \qquad (1)$$

where $\mathbf{A}$ is the system matrix, $\mathbf{y}$ is the sinogram measurements, $\mathbf{W}$ is the statistical weighting and $\mathbf{R}$ is the regularizer.

Each iteration updates the current image estimate $(\mathbf{x}^{(n)})$ using the following gradient of Ψ:

$$\nabla\Psi(\mathbf{x}^{(n)}) = \mathbf{A}^T\mathbf{W}(\mathbf{A}\mathbf{x}^{(n)}-\mathbf{y}) + \nabla\mathbf{R}(\mathbf{x}^{(n)}). \qquad (2)$$

Each iteration comprises four phases as shown in Figure 4. Each phase can be formulated to admit considerable parallelism over image voxels or detector values in the sinogram, making it well-suited to using the enormous compute capabilities of modern GPUs [10].

GPUs have multiple Streaming Multiprocessors that concurrently execute many threads [11]. These hardware elements execute a GPU kernel (programmed in CUDA) organized as blocks, warps and threads [12]. Kernel ordering is programmer-controlled, but ordering of blocks and warps is left to the hardware scheduler.

In our GPU implementation of PWLS-OS, we parallelize forward projection by partitioning views across multiple GPUs, employing one GPU thread to compute the value of one detector residual. Every GPU broadcasts its generated residual values to every other GPU, since back projection uses all of them. We then parallelize computation of the other phases by partitioning the y plane across multiple GPUs. Each thread performs the back projection, regularization and update of one image voxel. The GPUs then copy the corresponding partial image to other GPUs before the next iteration commences.

B. Overlapping copy with compute

As shown in Figure 1, although PWLS-OS is highly amenable to parallelization, GPU scaling does not yield linear speedup due to time taken to perform all-to-all copies of the detector residual at the end of forward projection and of the image voxels at the end of the update phase. The total amount of data copied increases linearly with number of GPUs; although each GPU produces fewer values, they must copy to more destinations. Since using more GPUs also reduces compute time per GPU, scaling beyond 12 GPUs results in copy time that exceeds compute time.

The available copy bandwidth is bound by the interconnect (PCIe3.0 and PCIe4.0 pose theoretical limits of 16GB/s and 32GB/s, respectively, though substantially lower sustained throughput is achievable in practice) and cannot be increased without hardware enhancements. However, a careful study of the hardware utilization pattern during the reconstruction process indicates that the interconnect remains idle during the computation phases and is utilized to its practical limit only during the ensuing copy phase, when no compute takes place. Hence, one way to decrease the reconstruction time is to initiate copies of data generated early in the computation phase while the rest is still being computed, overlapping computation and copying, as depicted in Figure 2.

To overlap compute and copy, the copy mechanism must impose minimal overhead and interference on the computation. Our concurrent work [13] identifies GPU thread-based copy using dynamic kernel launch as the mechanism best suited for our purpose. In this technique, the GPU threads that perform computation also trigger copies after generating a data chunk (e.g., a set of sinogram bins) of the desired granularity. The copies are issued via *dynamic kernel launch*, wherein a copy kernel is launched from within the main CT computation kernels—a capability introduced in CUDA 5.0 [14]. The copy kernel is scheduled by the GPU hardware scheduler and copies the data chunk to the other GPUs as shown in Figure 3. As computation proceeds, copy kernels are triggered, scheduled by hardware, and executed.

Although our technique incurs minimal copy initiation overhead, the copy kernels nevertheless use GPU resources that might otherwise have been used by CT computation kernels, and hence indirectly delay computation. Hence, invoking the copy kernel at an appropriate frequency while ensuring that the maximum amount of copy time is hidden behind compute becomes crucial. We address this challenge by carefully tuning the granularity at which copies are initiated. Since this granularity depends only on the CT geometry and not on patient features, it can be determined in advance via a parameter-sweep over a sample image.

Choosing the right granularity to track data production is also important, as tracking at too low granularity (such as thread granularity) would increase the overhead of tracking,

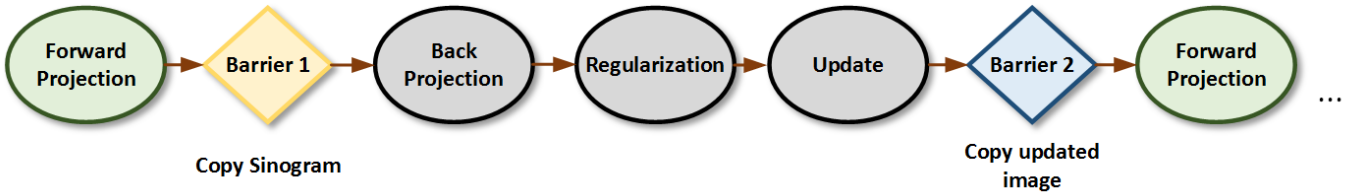

Fig. 4: Phases of X-Ray CT MBIR. After forward projection, the GPUs exchange their generated sinogram data, followed by three computational phases and an image update. The GPUs then exchange image data and the algorithm repeats until convergence.

slowing computation, while tracking at too high granularity might not ensure enough compute-copy overlap. We perform the tracking at the granularity of GPU thread blocks. Once consecutive thread blocks produce a chunk of data of the desired granularity, a dynamic copy kernel initiates the copies.

Our final design decision entails selecting the best implementation to track data generation. Proper design of this mechanism is crucial because the CUDA programming model offers enormous freedom to the scheduler with respect to block ordering, so blocks may complete in any order. We employ an atomic counter-based approach, wherein each data chunk is assigned a corresponding atomic counter, initialized to the number of blocks that contribute data to the chunk. The first thread of each block waits until all its sibling threads complete, then decrements the counter using an atomic decrement instruction. When the counter reaches zero, it indicates that the corresponding group of consecutive blocks is complete. The thread then initiates a dynamic copy kernel for the corresponding data chunk. Although the counter-based approach uses atomic accesses that are inherently slower than normal reads and writes, it performed better than alternatives (e.g., dedicated threads that poll for chunk completion).

C. Sequencing data generation

To ensure that the reconstruction method can effectively use the copy strategy discussed above, it is important to structure the kernels to perform all the computation corresponding to a particular data element in quick succession, producing data elements incrementally rather than performing multiple updates to all data elements during kernel execution. The goal is to ensure that data chunks are available as early as possible to maximize copy-compute overlap. While the original forward projection code computed the detector residuals in succession, the back projection code updated the relevant voxels using one view before updating the voxels using the next view, i.e., an outer loop over views. This led to voxels being ready for copy only during the processing of the last view, leaving very little room for compute-copy overlap. We restructured the code to generate the voxels in succession by having each voxel loop over the relevant views that contribute to it, thus ensuring voxel values are produced incrementally.

III. Experimental Results

We report on our multi-GPU PWLS-OS implementation. We validate the GPU implementation against a CPU baseline, which it matches to within 0.0289HU (Hounsfield units). Our test system comprises four Tesla K40m GPUs, each having

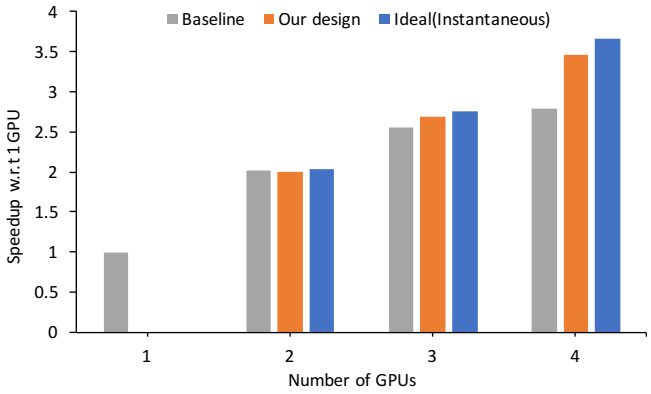

Fig. 5: Measured speedups achieved through GPU scaling

2880 CUDA cores, 11.9GB global memory and 4KB shared memory per block. Each GPU is capable of performing a peer access to the other GPUs. They reside on a PCIe3.0 bus that also interfaces them with the host.

We simulated 9-turn helical CT data with pitch 63/64 and 7256 views of size 64 rows by 888 channels, and reconstructed a 512×512×512 voxel image volume over a 512 mm transaxial field of view (FOV) with 0.625 mm slice thickness using the separable footprint projector [15] and [8] with 24 subsets.

In our design, the GPU threads initiated through dynamic kernel launch perform copies, as explained in Section II-B. The copies are performed for 8kB data chunks for detector residuals and 16kB for image voxels. We compare our approach against the baseline wherein data is copied after compute phases using cudaMemcpy Peer-to-Peer [16].

Figure 5 shows speedup achieved with respect to a single GPU for three cases: (1) Baseline, (2) Our design, and (3) Instantaneous copies (copies are performed in zero time). The blue (Instantaneous) bars indicate the theoretical limit on the performance gains achievable via compute-copy overlap. With only two GPUs, copy overhead is negligible and there is little performance difference between the baseline and the ideal. However, the potential and realized gains grow rapidly with further scaling. Our approach realizes 94% of the theoretical opportunity for four GPUs, achieving a speedup of 1.24× over the baseline and 3.46× over a single GPU.

Our design falls short of the opportunity available with instantaneous copies for two reasons: (1) The dynamic copy kernels require some GPU execution bandwidth, slightly delaying execution of CT kernel threads. (2) A 100% copy-compute overlap is not possible because the data generated by the final blocks is copied after computation is complete.

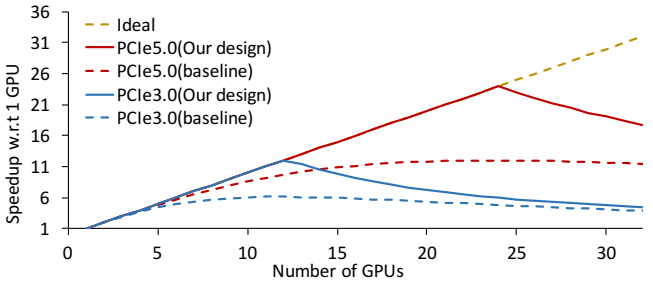

Fig. 6: Effect of GPU scaling on speedup for different interconnect generations

IV. Scalability Analysis

Our approach holds the potential to unlock even higher performance scalability on future, larger multi-GPU systems. To analyze this potential, we develop a simple analytic model that predicts the impact of our approach with more GPUs and faster GPU interconnects. We base our scalability model on the following observations: (1) As previously explained, the four phases of X-ray CT are amenable to GPU scaling. For simplicity, we assume that the compute time of individual algorithmic phases scales linearly with the number of GPUs, although practical implementations typically fall a bit short of ideal linear scaling. (2) An all-to-all broadcast must occur at the end of the forward projection and update phases. Thus, as the number of GPUs increases, the total data to be copied, and hence the time required for copy, increases, since the data must be copied to additional GPUs.

A. Scalability Model

At the end of forward projection, the GPUs must exchange the portions of the sinogram each generated. The total bytes copied is the product of the sinogram size ($\text{Size}_{\text{sino}}$), and the number of destination GPUs ($n_{\text{gpu}} - 1$). The total time for the copy depends upon the interconnect technology, which we model simply as a 'copy time per byte (t_{perbyte})' bandwidth, as expressed in the following equation:

$$\text{Copytime}_1 = (n_{\text{gpu}} - 1) \ * \ \text{Size}_{\text{sino}} \ * \ t_{\text{perbyte}}.$$

After the update phase, the GPUs must exchange the image portions each generated. Hence the total bytes copied is the product of the total image size ($\text{Size}_{\text{image}}$) and the number of destination GPUs ($n_{\text{gpu}} - 1$):

$$\text{Copytime}_2 = (n_{\text{gpu}} - 1) \ * \ \text{Size}_{\text{image}} \ * \ t_{\text{perbyte}}.$$

B. Discussion

Using this simple model, we estimate the impact of our design on CT reconstruction performance. Figure 6 shows the projected speedup of our design and that of the baseline for different GPU counts against a single GPU implementation for two different assumptions on interconnect bandwidth (unidirectional transfer bandwidths of 16GB/sec for PCIe 3.0 and an estimated 64GB/s for PCIe 5.0).

For both assumptions on interconnect bandwidth, our approach enables performance scalability to a much larger number of GPUs than the baseline. For PCIe 3.0, baseline performance saturates at about 6× speedup (over a single GPU) with ten GPUs. Above six GPUs, growth in copy time exceeds reductions in computation time. In contrast, our approach enables near-ideal scaling up to twelve GPUs. Beyond this point, copy time grows to the point where it exceeds compute time and can no longer be hidden.

Higher interconnect bandwidth under PCIe 5.0 reduces copy time, enabling greater performance scalability for both the baseline and our method. However, our technique still drastically increases the scalability potential. Our technique enables near-ideal scaling up to 24 GPUs, while the baseline saturates at about 12× speedup with 20 GPUs. Newer GPUs will further decrease compute time by the time PCIe 5.0 becomes available, making our solution even more relevant.

V. Summary and Conclusion

In this paper, we proposed compute-copy overlap using warp based copies to overcome the communication bottleneck of GPU scaling of CT MBIR. We demonstrated that our technique offers a 3.46× speedup over a single GPU implementation on a 4 GPU system, and makes CT MBIR more amenable to further GPU scaling.

References

[1] J.-B. Thibault, K. Sauer, C. Bouman, and J. Hsieh, "A three-dimensional statistical approach to improved image quality for multi-slice helical CT," *Med. Phys.*, vol. 34, no. 11, pp. 4526–44, Nov. 2007.

[2] K. Zeng, E. Bai, and G. Wang, "A fast CT reconstruction scheme for a general multi-core PC," *Intl. J. Biomedical Im.*, vol. 2007, p. 29160, 2007.

[3] R. Sampson, M. G. McGaffin, T. F. Wenisch, and J. A. Fessler, "Investigating multi-threaded SIMD for helical CT reconstruction on a CPU," in *Proc. 4th Intl. Mtg. on image formation in X-ray CT*, 2016, pp. 275–8.

[4] J. Ni, X. Li, T. He, and G. Wang, "Review of parallel computing techniques for computed tomography image reconstruction," *Current Medical Imaging Reviews*, vol. 2, no. 4, pp. 405–14, Nov. 2006.

[5] J. Rosen, J. Wu, T. Wenisch, and J. Fessler, "Iterative helical CT reconstruction in the cloud for ten dollars in five minutes," in *Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med*, 2013, pp. 241–4.

[6] M. McGaffin and J. A. Fessler, "Alternating dual updates algorithm for X-ray CT reconstruction on the GPU," *IEEE Trans. Computational Imaging*, vol. 1, no. 3, pp. 186–99, Sep. 2015.

[7] B. Jang, D. Kaeli, S. Do, and H. Pien, "Multi GPU implementation of iterative tomographic reconstruction algorithms," in *Proc. IEEE Intl. Symp. Biomed. Imag.*, 2009, pp. 185–8.

[8] D. Kim, D. Pal, J.-B. Thibault, and J. A. Fessler, "Improved ordered subsets algorithm for 3D X-ray CT image reconstruction," *Proc. 2nd Intl. Mtg. on image formation in X-ray CT*, pp. 378–81, 2012.

[9] T. Zinßer and B. Keck, "Systematic performance optimization of cone-beam back-projection on the Kepler architecture," in *Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med*, 2013, pp. 225–8.

[10] K. Mueller, F. Xu, and N. Neophytou, "Why do commodity graphics hardware boards (GPUs) work so well for acceleration of computed tomography?" in *Proc. SPIE 6498 Comp. Imag.*, 2007, p. 64980N.

[11] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "NVIDIA Tesla: A unified graphics and computing architecture," *IEEE micro*, vol. 28, no. 2, 2008.

[12] D. B. Kirk and W. H. Wen-Mei, *Programming massively parallel processors: a hands-on approach*. Morgan Kaufmann, 2016.

[13] H. Muthukrishnan, J. A. Fessler, and T. F. Wenisch, "SUBLIME: Hiding copies in multi-GPU systems," 2018, submitted.

[14] S. Jones, "Introduction to dynamic parallelism," in *GPU Technology Conf. Presentation S*, vol. 338, 2012, p. 2012.

[15] Y. Long, J. A. Fessler, and J. M. Balter, "3D forward and back-projection for X-ray CT using separable footprints," *IEEE Tr. Med. Im.*, vol. 29, no. 11, pp. 1839–1850, 2010.

[16] T. Schroeder, "Peer-to-peer and unified virtual addressing," in *GPU Technology Conference, NVIDIA*, 2011.