A High-Performance and Energy-Efficient CT Reconstruction Algorithm For Multi-Terabyte Datasets

Edward S. Jimenez[1], Laurel J. Orr[1], Kyle R. Thompson[1], and Ryeojin Park[2]

[1]*Sandia National Laboratories*
PO Box 5800
Albuquerque, NM 87185, USA
{esjimen,ljorr,krthomp}@sandia.gov

[2]*University of Arizona*
1401 E. University Blvd
Tucson, AZ 85721, USA
rpark@email.arizona.edu

***Abstract*—There has been much work done in implementing various GPU-based Computed Tomography reconstruction algorithms for medical applications showing tremendous improvement in computational performance. While many of these reconstruction algorithms could also be applied to industrial-scale datasets, the performance gains may be modest to non-existent due to a combination of algorithmic, hardware, or scalability limitations. Previous work presented showed an irregular dynamic approach to GPU-Reconstruction kernel execution for industrial-scale reconstructions that dramatically improved voxel processing throughput. However, the improved kernel execution magnified other system bottlenecks such as host memory bandwidth and storage read/write bandwidth, thus hindering performance gains. This paper presents a multi-GPU-based reconstruction algorithm capable of efficiently reconstructing large volumes (between 64 gigavoxels and 1 teravoxel volumes) not only faster than traditional CPU- and GPU-based reconstruction algorithms but also while consuming significantly less energy. The reconstruction algorithm exploits the irregular kernel approach from previous work as well as a modularized MIMD-like environment, heterogeneous parallelism, as well as macro- and micro-scale dynamic task allocation. The result is a portable and flexible reconstruction algorithm capable of executing on a wide range of architectures including mobile computers, workstations, supercomputers, and modestly-sized hetero or homogeneous clusters with any number of graphics processors.**

I. Introduction

Industrial Computed Tomography (CT) is an indirect 3D imaging technique that typically consists of datasets that are orders-of-magnitude larger compared to medical-scale datasets due to detectors with many pixels (usually 6-16 million pixels), many more projections, larger reconstruction volumes, high energy x-rays, or any combination thereof [1]–[3].

The reconstruction algorithm, frequently Feldkamp-Davis-Dress [4] for large-scale industrial applications, is computationally complex at $o(n^4)$ and can require an unreasonable amount of time to complete in a traditional computing environment. The computational complexity of reconstruction has let to the investigation of graphics processing units (GPUs) to be utilized as coprocessors to complete the heavy computational task of the reconstruction algorithm while the central processing unit (CPU) manages storage and memory I/O tasks. For medical datasets, GPU-based reconstruction algorithms have seen tremendous improvement in computation time while achieving comparable numerical stability [5], [6].

Many of these GPU-based approaches to medical-scale CT reconstruction could easily be applied to industrial-scale datasets; unfortunately, the time required to reconstruct could still be unacceptable due to bottlenecks created by the larger input, increased computational requirements, host and/or device bandwidth limitations, and irregular memory access patterns [3], [7]. Furthermore, these bottlenecks could become much more exaggerated by the GPU kernel design (i.e instruction ordering) as well as the host I/O capability in both host memory and storage. The large-scale datasets may range from a few gigabytes to several terabytes for both the input and the output reconstructed volume; this results in non-trivial storage tasks as well as severe bandwidth pressure on the entire system.

This work presents a flexible and portable multi-GPU reconstruction algorithm that exhibits high-performance and energy efficiency on a wide range of reconstruction tasks applicable to industrial applications. This work will present performance results on two large-scale datasets; the first, is a 64 gigavoxel reconstruction from 1800 16 megapixel projections; the second, a synthetic ”future-sized” dataset consisting of a teravoxel (1 trillion voxels) reconstructed from 10,000 100 megapixel projections. There are two high-performance multi-GPU algorithms that will be presented as well as a ”naïve” multi-GPU implementation and a traditional CPU-based approach for comparison.

II. Approach

The approach presented in this work utilizes an irregular kernel design developed by Jimenez et. al. [3]. An irregular approach for the kernel design is crucial as the scan geometry for many industrial applications can vary widely due to object size, magnification requirements, or hardware system limitations. Any combination of these leads to an unpredictable memory access pattern requiring a variable amount of data for any given neighborhood within the reconstruction volume as shown in figure 1. A wide memory access pattern combined with the massive input dataset and massively multithreaded environment of the GPU device results in an irregular computation for reconstruction tasks. Work done by Jimenez and Orr [7] demonstrated the impact of kernel design for irregular reconstructions and showed that computational times improved by up to three-fold when accounting for the irregular nature of the computation. This improvement could be the difference between a large reconstruction requiring many days to complete to accomplishing the same computation in less than a day.

Thread-based parallelization of many reconstruction algorithms is done by assigning a voxel (or set of voxels) to a thread and executing parallel threads to simultaneously update voxel information. Thus, for a cluster of threads updating voxel information (regardless of proximity with respect to voxel position), the back-projection nature of reconstruction could potentially result in threads accessing data from the input in non-localized regions; in the case of GPUs, this results in a computational performance degradation due to severe memory latencies of accessing disparate memory addresses.

The approach described by Jimenez et. al. showed that the key to improving kernel computational performance was dependent on not only massive multi-threading and fast memory, but also:

- *Memory uploads* - Data uploads are maximized instead of minimized to accommodate more slices simultaneously.
- *Host Pinned Memory* - Pinned host memory will allow for faster data upload to the devices.
- *GPU Cache hit-rate maximization* - An irregular approach dramatically improves the GPU cache hit-rate thus maximizing computational performance and a reduction of wasted GPU clock cycles.
- *Dynamic Task Allocation* - Varying GPU tasks with respect to location in the volume will improve load-balancing between the GPUs as well as benefit the irregular computation.
- *Instruction Ordering* - Kernel design was implemented such that any register latency from write backs are amortized by assigning instructions that are independent of the inaccessible register.
- *Resource Maximization* - The kernel is designed independent of the GPU model and specifications, this will allow the kernel to execute optimally across a wide variety of GPUs. Resource maximization will query the resources available on the GPU and ensure that all compute cores are utilized as well as all device memory.
- *Cache Structure* - The cache structure of GPUs is distinctly unique compared to traditional CPUs. Namely, texture and constant (”scratch pad” cache) [8] cache. To maximize performance, these unique caches must be exploited.

Addressing the computational bottleneck of reconstruction only addresses a portion of the problem. Once all voxels of a given image plane have been reconstructed, the image plane is typically written to storage media. For industrial applications, this could range from several dozen gigabytes to several terabytes for the entire reconstructed volume. Since most computing systems cannot store the entire reconstructed volume in system memory, the reconstruction algorithm will intermittently pause to write the completed subvolume to storage media. For large-scale industrial applications, this results in tremendous performance degradation as the consequences are two-fold: The I/O storage tasks require nontrivial time to complete the task and all GPUs sit idle, not contributing to the progress of the reconstruction all while consuming energy waiting for the completion of the pending storage tasks.

This issue was addressed by Orr and Jimenez [9] by implementing a modularized approach to improve architectural support to a multi-GPU algorithm to reduce GPU idle time by overlapping storage write tasks during GPU computation via an input read-ahead approach. This approach dynamically queries the entire system to determine the available host memory, number of devices, and specifications of each device to determine the largest subvolume that can reside in host memory while still allowing for relevant projection image input data to reside in host memory as well. Next, the algorithm utilizes an MIMD-like approach by assigning varying assignments to each available host thread on the system. Each thread is assigned to either manage kernel launches and memory transfers between host and a specific device or manage post-processing and storage writes of the reconstructed image planes. The algorithm will read the input to host memory and begin GPU-based computations on a given subvolume of the volume determined to fit in host memory. Upon completion, the reconstructed subvolume is transfered to host memory by its assigned thread and computation of the next subvolume is immediately started. While computation of the next subvolume is performed, the host threads assigned to storage writing tasks receive a message that write tasks are pending and then begin writing pending image planes to storage and continue to do so until all pending planes are written. The original intent of Orr and Jimenez was to demonstrate an approach to reconstruct ”future-sized” datasets efficiently. However, this work contends that this approach is also beneficial to current large-scale reconstructions.

III. Implementation

The algorithms presented were developed using Visual Studio 2008 and were written using the C++ programming language. The GPU kernels were written using Nvidiaś CUDA programming environment version 5.0. All implementations will execute the FDK reconstruction algorithm [4].

A. Modular Approach

The modular approach (MA) utilizes the kernel design optimizations of Jimenez et. al. and the storage I/O optimizations described by Orr and Jimenez and the storage I/O optimizations described by Orr and Jimenez. This approach, as described in the previous section, requires parallel host threads to either supply input data to a particular device or apply post-processing on completed reconstructed image planes and write them to storage media. Parallel host code was achieved using OpenMP 2.0.

B. Serial Approach

The serial approach (SA) also utilizes the kernel design of Jimenez et. al. but does not exploit the I/O optimizations of Orr and Jimenez. This approach still requires host parallelization, all available host threads are evenly distributed to all available devices and are responsible for providing input data to its assigned device as well as post-processing and storage writing tasks. This approach is presented to illustrate the impact of the approach taken by Orr and Jimenez. Parallelization of the host code is achieved by utilizing OpenMP 2.0.

C. Naïve Approach

For comparison purposes, a ”naïve” multi-GPU-based implementation will also be presented. This naïve approach is a port of a CPU-based reconstruction algorithm that is modified sufficiently to function on a GPU, but does not exploit any GPU specific characteristics such as hardware interpolation, cache optimizations or subvolume processing. The main feature of the approach is minimal and includes exploiting the multi-threaded nature of the GPU. Each kernel launch updates a single image plane with data from a single projection serially until all projection data

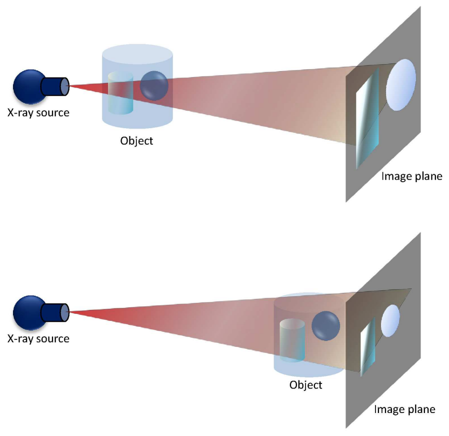

Fig. 1. Example of irregular memory access due to variable geometry

has been applied to the image plane. Additionally, none of the optimizations identified by Jimenez et. al. are exploited. The naïve approach is similar to the one presented by work done by Jimenez and Orr [7]. Similar to the above implementations, parallelization of the host threads is done by using OpenMP 2.0.

D. CPU Implementation

For baseline comparison, we present a CPU-based design that is widely used in the industrial non-destructive testing community. The CPU-design is a hybrid design that parallelizes the source code using OpenMP 2.0 and MPICH2. This approach is optimized for maximal CPU utilization.

IV. Evaluation

This work will focus on the performance of each implementation on a single system. The system consists of a Supermicro Server with dual Intel Xeon Processors clocked at 2.0 GHz (Octo-core with Hyper-Threading), 512 GB RAM, 8 disk RAID0 array connected via an Intel Controller, and 8 Tesla M2090 GPUs. Each M2090 device is a Fermi-class device with 6GB of GDDR5 memory and 512 streaming processors.

Performance of each implementation will be measured against two datasets. The first is a 4000^3 voxel volume reconstruction from 1800 16 megapixel projections which is representative of the larger-end of current reconstructions. The second is a teravoxel (1 trillion voxels) volume reconstruction from 10,000 100 megapixel projections, which was selected as a stress-test as well as a potential future-sized dataset. Performance metrics to be measured include computation time, kilowatt-hours consumed (measured using Kill-A-Watt meters connected directly to the system), and performance scalability with respect to the number of GPUs.

V. Results

A. 64 Gigavoxel Dataset

Figure 2 presents the computational performance of each GPU-based implementation. Both optimized approaches far outperform the naïve approach; in fact, performance of the naïve approach degrades significantly when more GPUs are added. The figure also seems to show very little improvement with respect to GPUs; this seems to imply that the computation task in the 64-billion voxel volume is not the main bottleneck, but in fact the storage I/O is the main hindrance, this was further supported by the observance of each GPU reconstructing image planes at a rate greater than one per second. I/O burden seems to be slightly alleviated by exploiting the I/O optimization of Orr and Jimenez, but with the lack of significantly overlapping tasks (compute and write) due to quick computations, the performance gains are very modest. The CPU-based approach required over 34 hours to complete the same reconstruction; many works have shown that GPU-based CT reconstruction significantly outperform CPU-based implementations, so this is to be expected [6], [10], [11].

Figure 3 presents energy consumption for each implementation. The modular approach performs between 19 and 48 percent more efficiently than the serial approach, and may be attributed to the reduced GPU downtime and slightly improved computational performance. However, both optimized approaches dramatically outperform the naïve and CPU-based implementations by an order-of-magnitude for 8 GPUs.

Figure 4 shows the scalability with respect to devices for each GPU-based implementation. Since the computation task is not the main bottleneck for the optimized approaches, it is not surprising that there is minimal scalability improvement. However, as seen in figures 2 and 3, scalability is absent for the naïve approach.

B. Teravoxel Dataset

The synthetic teravoxel dataset was large enough to adequately stress all implementations so that true performance comparisons could be made. It should be noted that for the CPU-based reconstruction, only a subvolume was reconstructed and the values for the entire volume were extrapolated. The subvolume reconstructed consisted of 10 billion voxels located in the center image planes of the volume as this is typically the region that

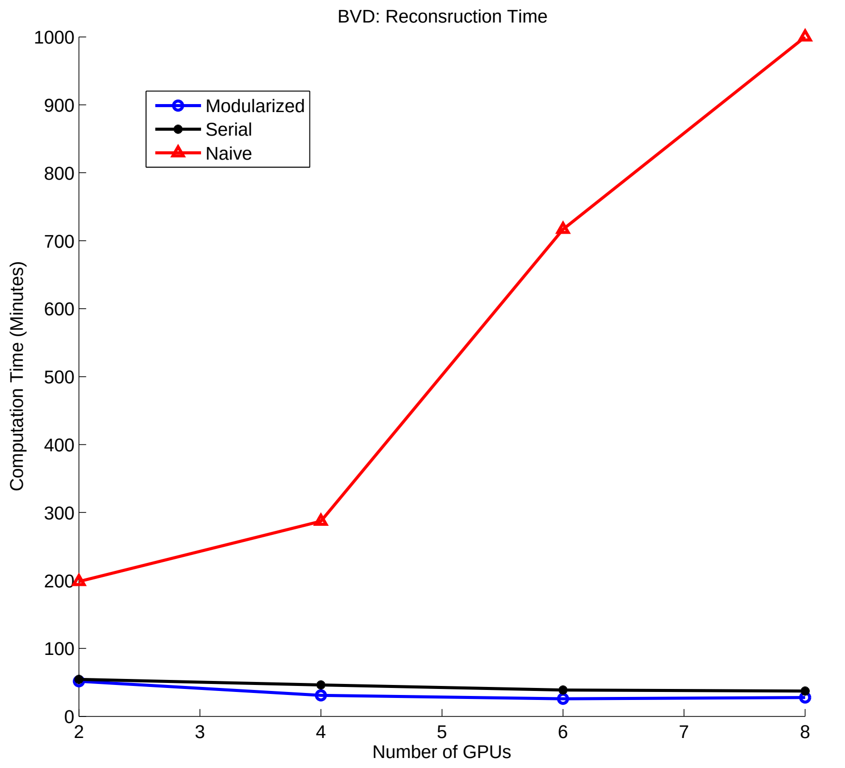

Fig. 2. Reconstruction time performance for the 64 gigavoxel volume with respect to number of GPUs

the algorithm performs best due to coalesced memory reads and reduced necessary data for a full reconstruction.

Figure 5 shows a tremendous improvement in computational performance for both optimized implementations over the naïve approach. Additionally, performance is measurably improved when increasing the number of devices. the I/O optimization in the modular approach has a measurable impact resulting in an improvement in computation time of approximately 12 hours for a single GPU and over 3 hours for 8 GPUs. The CPU-based implementation required over 2500 hours.

Figure 6 shows that the optimized implementations are about two orders-of-magnitude more energy efficient than the CPU-based method and an order-of-magnitude more energy efficient than the naïve approach. The trend for the optimized approaches is similar to the trend seen in figure 5 implying that the reduced GPU downtime is the main source of energy savings for the modular approach when compared to the serial approach.

Figure 7 shows that the optimized approaches scale nearly identically with a slight deviation at 8 GPUs. The naïve approach exhibits better scalability compared to figure 4; however, the scalability performance is very poor compared to the optimized implementations.

VI. Conclusion

This work has shown that kernel design of GPU-based reconstruction that is focused on the irregular nature of large-scale reconstruction tasks will dramatically improve performance when compared to CPU- and other GPU-based methods. Additionally, a modular approach was shown to improve performance on current large-scale datasets and not just future-scale datasets as was shown by Orr and Jimenez [9].

Intelligent algorithm design in GPU kernels has shown that significant energy savings can be achieved, again, not only compared to CPU-based methods, but also against Naïve approaches to GPU kernel design. No previous work could be found on energy consumption metrics of reconstruction algorithms. The initial tests presented in this work show that large-scale industrial reconstructions could realize tremendous energy savings, even for the cases in which trillions of voxels need to be processed. In the case of the 64 billion voxel dataset, a 23.5x energy consumtion improvement was realized and a 35.2x consumption improvement for the teravoxel dataset. Future work will study other important aspects of efficiency such as the energy-delay product [12], amperage, voxel throughput and global reconstruction throughput per megajoule, etc. Many green computing efforts focus on the hardware design to improve energy efficiency, but clearly there is work to be done on the software design aspects of efficiency as well.

VII. Acknowledgements

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[1] S. Izumi, S. Kamata, K. Satoh, and H. Miyai, "High energy x-ray computed tomography for industrial applications," *Nuclear Science, IEEE Transactions on*, vol. 40, no. 2, pp. 158 –161, apr 1993.
[2] H. H. Barrett and K. J. Myers, *Foundations of Image Science*. Wiley-Interscience, 2004.

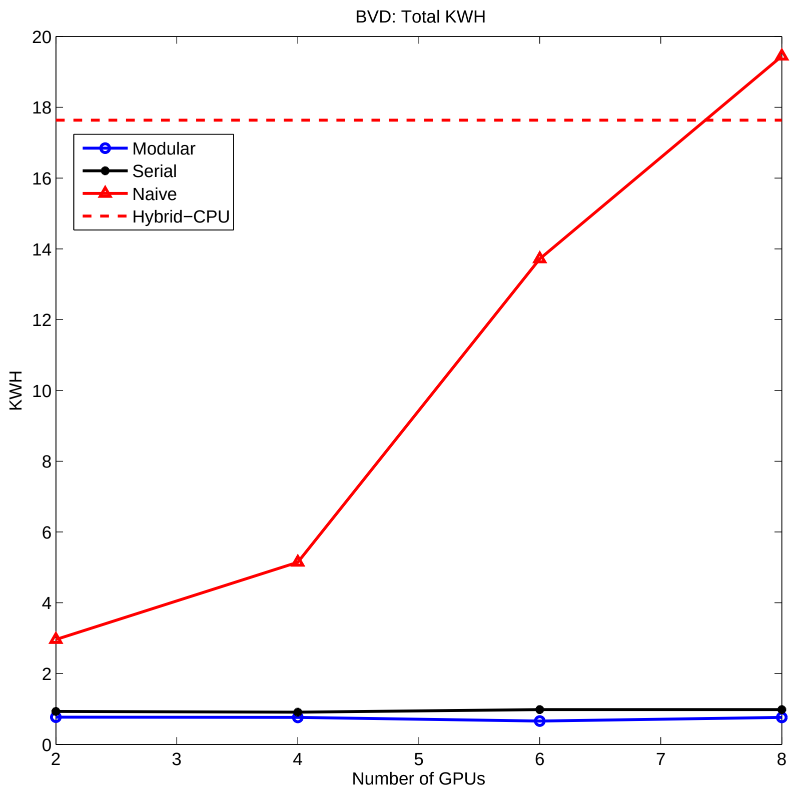

Fig. 3. Energy Consumption the 64 gigavoxel reconstruction with respect to number of GPUs

[3] E. Jimenez, L. Orr, and K. Thompson, "An irregular approach to large-scale computed tomography on multiple graphics processors improves voxel processing throughput," in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, 2012, pp. 254–260.

[4] L. Feldkamp, L. Davis, and J. Kress, "Practical cone-beam algorithm," *Journal of the Optical Society of America A*, vol. 1, no. 6, pp. 612–619, 1984.

[5] S. Xiao, Y. Bresler, and J. Munson, D.C., "Fast feldkamp algorithm for cone-beam computer tomography," in *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on*, vol. 2, sept. 2003, pp. II – 819–22 vol.3.

[6] F. Xu and K. Mueller, "Ultra-fast 3d filtered backprojection on commodity graphics hardware," in *Biomedical Imaging: Nano to Macro, 2004. IEEE International Symposium on*, april 2004, pp. 571 – 574 Vol. 1.

[7] E. S. Jimenez and L. J. Orr, "Rethinking the Union of Computed Tomography Reconstruction and GPGPU Computing," in *Workshop on Penetrating Radiation Systems and Applications XIV*, ser. SPIE Optical Engineering + Applications, Aug. 2013.

[8] J. W., S. K.A., and M. M., "Characterizing and Improving the Use of Demand-Fetched Caches in GPUs," ser. International Conference on Supercomputing 2012, June 2012.

[9] L. J. Orr and E. S. Jimenez, "Preparing for the 100-megapixel Detector: Reconstructing a Multi-Terabyte Computed-Tomography Dataset," in *Workshop on Penetrating Radiation Systems and Applications XIV*, ser. SPIE Optical Engineering + Applications, Aug. 2013.

[10] F. Xu and K. Mueller, "Accelerating popular tomographic reconstruction algorithms on commodity pc graphics hardware," *Nuclear Science, IEEE Transactions on*, vol. 52, no. 3, pp. 654 – 663, june 2005.

[11] K. Mueller, F. Xu, and N. Neophytou, "Why do commodity graphics hardware boards (GPUs) work so well for acceleration of computed tomography?" in *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, ser. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, vol. 6498, Feb. 2007.

[12] R. Gonzalez and M. Horowitz, "Energy dissipation in general purpose microprocessors," *Solid-State Circuits, IEEE Journal of*, vol. 31, no. 9, pp. 1277–1284, 1996.

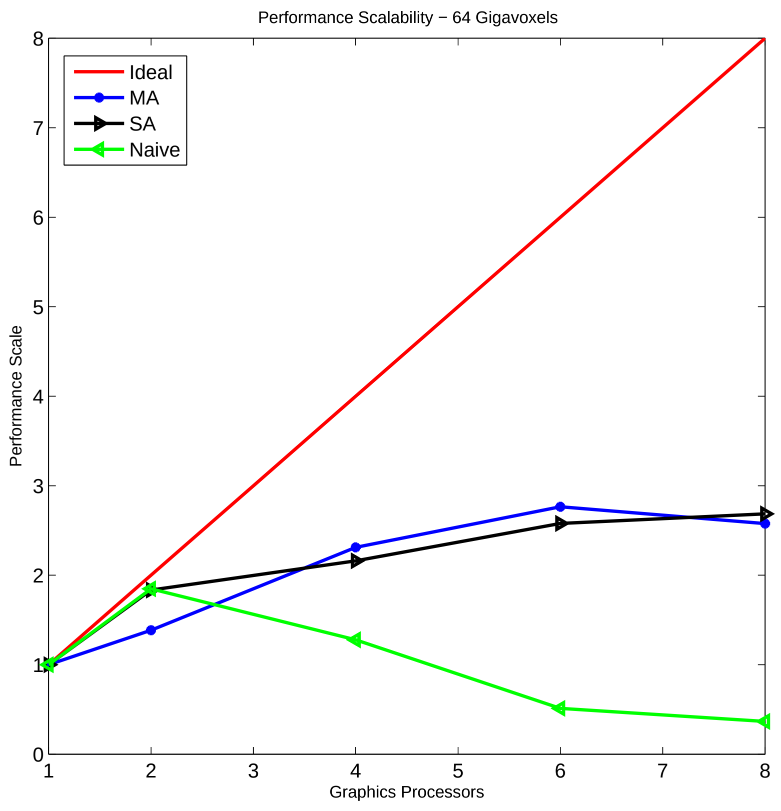

Fig. 4. Scalability with respect to GPU count for the 64 gigavoxel volume

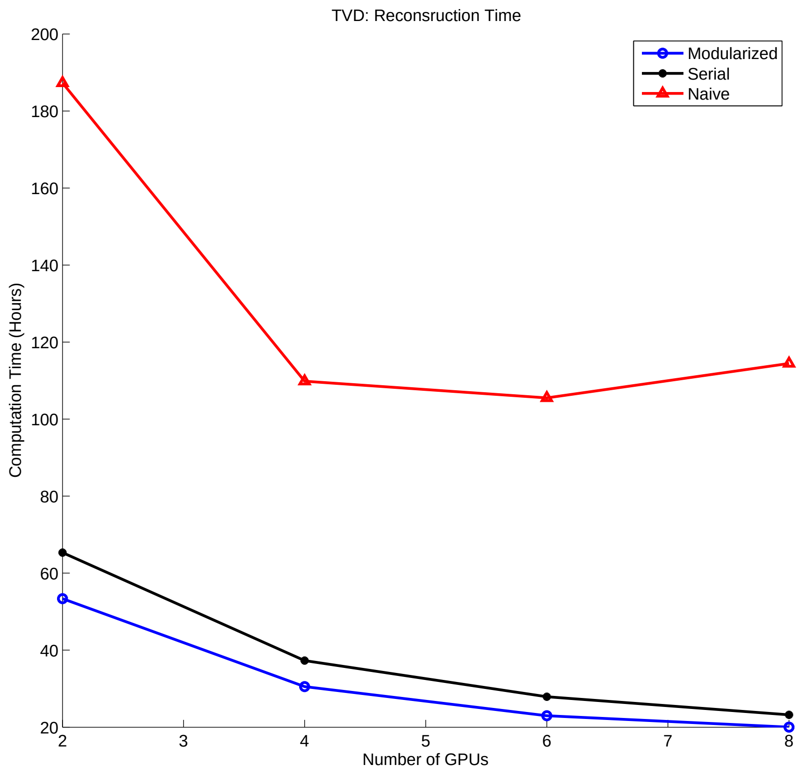

Fig. 5. Reconstruction time performance for the teravoxel volume with respect to number of GPUs

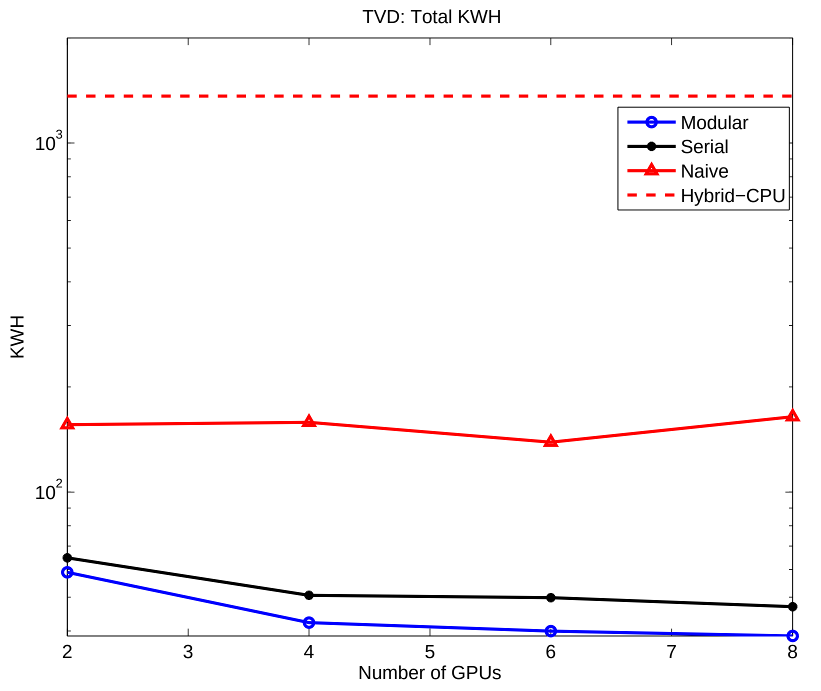

Fig. 6. Energy Consumption the teravoxel reconstruction with respect to number of GPUs

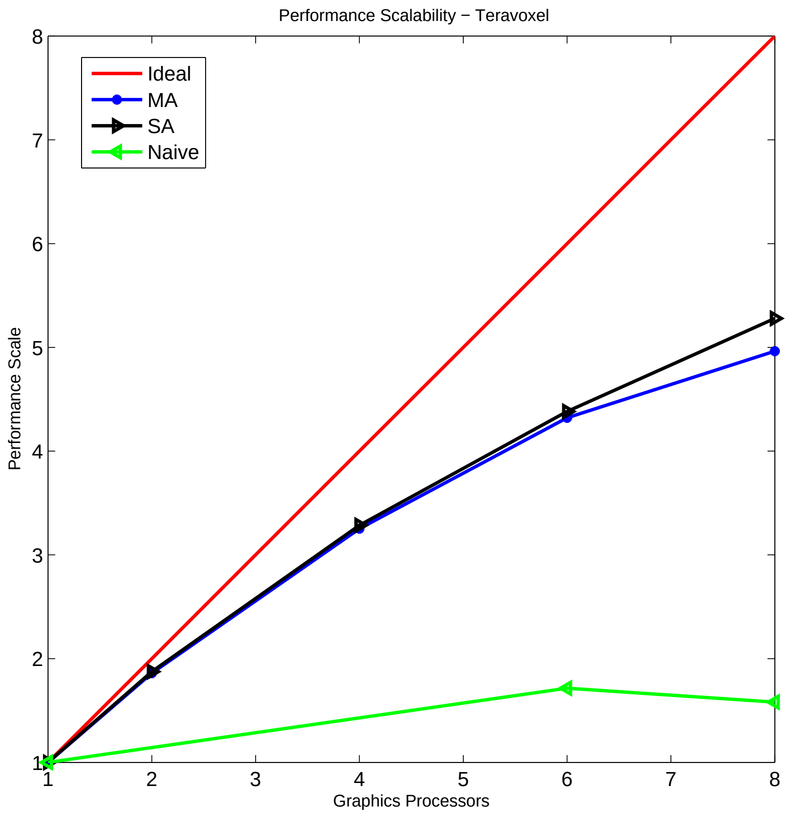

Fig. 7. Scalability with respect to GPU count for the teravoxel volume