SAND2014-20041C

Cluster-Based Approach to a Multi-GPU CT Reconstruction Algorithm

Laurel J. Orr, *Member, IEEE,* Edward S. Jimenez, *Member, IEEE,* and Kyle R. Thompson, *Member, IEEE,*

***Abstract*—Conventional CPU-based algorithms for Computed Tomography reconstruction lack the computational efficiency necessary to process large, industrial datasets in a reasonable amount of time. Specifically, processing time for a single-pass, trillion volumetric pixel (voxel) reconstruction requires months to reconstruct using a high performance CPU-based workstation. An optimized, single workstation multi-GPU approach has shown performance increases by 2–3 orders-of-magnitude; however, reconstruction of future-size, trillion voxel datasets can still take an entire day to complete. This paper details an approach that further decreases runtime and allows for more diverse workstation environments by using a cluster of GPU-capable workstations. Due to the irregularity of the reconstruction tasks throughout the volume, using a cluster of multi-GPU nodes requires inventive topological structuring and data partitioning to avoid network bottlenecks and achieve optimal GPU utilization. This paper covers the cluster layout and non-linear weighting scheme used in this high-performance multi-GPU CT reconstruction algorithm and presents experimental results from reconstructing two large-scale datasets to evaluate this approach's performance and applicability to future-size datasets. Specifically, our approach yields up to a 20 percent improvement for large-scale data.**

I. Introduction

Computed Tomography (CT) uses a series of X-ray images taken at various angles around an object to generate cross-sectional slices, also called image planes, to approximate the object's interior and exterior structure. These slices are then combined to form a reconstructed, 3-dimensional volume of the imaged object [1]. Although CT is typically associated with the medical field, it is commonly used in industry for internal component inspection and other nondestructive evaluation applications. It has been shown that utilizing the highly parallel architecture of the Graphics Processor Unit (GPU) on a single workstation can decrease the runtime by 2–3 orders-of-magnitude [2]. Industrial-size volumes, which are up to trillions of volumetric-pixels (voxels) in size, however, can be anywhere from 10 to 10,000 times the size of medical datasets. This means even when using a GPU-based approach, reconstruction can still require days to months to complete. Additionally, with the new advancements in high performance imaging technology [3]–[6], big data, Teravoxel volumes may soon be possible. However, with the current techniques, it would not be possible to reconstruct that massive a volume in a reasonable amount of time.

Laurel J. Orr and Edward S. Jimenez are with Sandia National Laboratories Software Systems R&D, Albuquerque, NM, {ljorr, esjimen}@sandia.gov.

Kyle R. Thompson is with Sandia National Laboratories Structural Dynamics & X-Ray Non-Destructive Evaluation, Albuquerque, NM, krthomp@sandia.gov.

Previous work by Orr and Jimenez [7], [8] made progress towards a solution by implementing a modularized, GPU-based, FDK reconstruction algorithm to run on a single node. The novelty of their approach was to separate the GPU computation from the data I/O to increase the GPU utilization and improve overall performance. By utilizing a high performance workstation, they were able to reconstruct a synthetic Teravoxel dataset in under 24 hours. However, their approach was limited to a single, high performance workstation, which is not a realistic setup for many users and is unable to scale. Since their computation was reaching the limits of one node's capabilities, further improvements to runtime were not possible.

Our work builds upon this single node approach and implements a novel, cluster-based big data reconstruction algorithm. We use the FDK reconstruction algorithm, although we could use any algorithm with slight modifications [9]. The rest of this paper will first give an overview of our approach in Section II and then give details on the critical components in Section III. We evaluate our implementation in Section IV and finally give concluding remarks and propose future work in Section V.

II. Approach

To improve performance, bring runtimes into a reasonable range, and allow for more flexible computing environments, this approach performs reconstruction across a cluster of computers where each computer has access to one or more GPUs. Note that we will use the term GPU to refer to both the chip and the entire device (graphics card), depending on the context. Implementing a high-performance yet flexible clustered, multi-GPU approach is a doubly irregular problem due to the variety of possible GPU-capable workstation environments and the irregular nature of large-scale reconstruction on GPUs [8], [10]. In particular, the ratio of the size of the input to the size of the output for a GPU is not constant throughout the volume due to magnification artifacts, which means the data cannot be linearly partitioned across the GPUs. Additionally, the implementation needs to ameliorate the potential network bottlenecks arising from transferring Gigabytes to Terabytes of data across the cluster.

At a high level, this approach declares one node to be a master, one to be an output node, and the rest to be slaves where each slave must have at least one available GPU. The master dynamically partitions the reconstruction volume across the cluster and instructs each slave to run a single, multi-GPU reconstruction task on its respective sub-volume following the

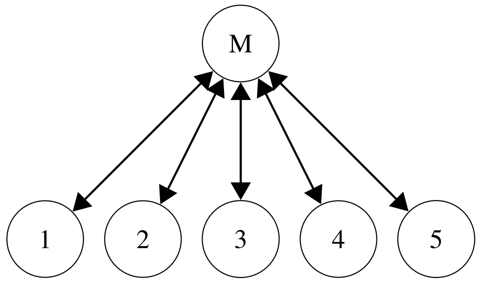

Fig. 1. One Node I/O

method presented in [7], [8]. Each slave requests a subset of preprocessed X-ray sinogram images (input to GPUs) from the master to minimize the number of data transfers while maintaining that all sinogram images and reconstructed image planes (output of GPUs) fit in the host's memory (host refers to the node/workstation where the GPU resides). The slave iteratively runs the reconstruction on a block of sinograms, transfers the reconstructed image planes to the output node once reconstruction is finished, and requests a new block of sinogram images. This process is repeated until the entire reconstruction task is completed.

III. Implementation

A. Cluster Structure

In traditional reconstruction algorithms, there is typically a single node with a large amount of storage space that receives and stores all the raw X-ray data from the imaging system. Since it is not guaranteed that the entire dataset can be transferred to each workstation, cluster-based reconstruction algorithms utilize a single storage device to handle all data I/O. For a cluster, the master is typically the node responsible for providing all I/O operations [11]. Due to not only the massive amount of data to be transferred across the cluster (all sinograms and image planes) but also the fact that GPUs process the X-ray images and output image planes faster than the host can transfer data, network bottlenecks will likely occur at the master when slaves start requesting input data while simultaneously transferring reconstructed image planes. Due to limited bandwidth, slaves will experience delays when transferring data, resulting in suboptimal GPU utilization and performance.

To alleviate network congestion, this implementation allows another node other than the master to handle writing data to storage media while the master still handles reading sinogram images (see Figure 2). With two nodes handling data I/O, although the bottleneck is not completely annihilated, it is significantly reduced due to the modularized nature of the reconstruction algorithm. Additionally, the net data transfer is the same as when there is only one I/O node, so we are not unnecessarily sending data across the network.

B. Load Balancing

To allow for versatile, heterogeneous clusters, we must accommodate nodes having varying GPU compute capabilities (architecture and supported features). If the volume was

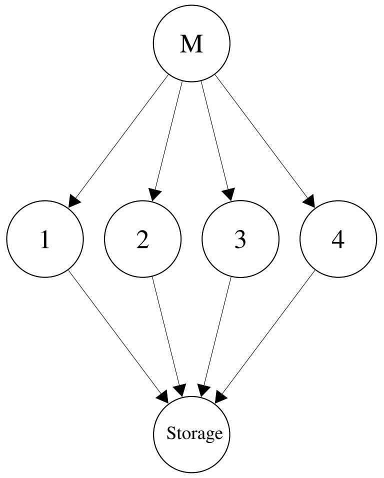

Fig. 2. Two Node I/O

naïvely equally split between the nodes, performance would be limited by how fast the weakest node could process. For example, given three nodes all with the same model of GPU where two nodes had 8 GPUs and the third only had one, then the two nodes with 8 GPUs would sit idle waiting for the single GPU node to complete its task. If a larger portion of the volume went to the two 8 GPU nodes, the overall runtime and GPU utilization would decrease. However, as mentioned in Section I, the amount of input needed to reconstruct a single image plane varies depending on the overall size of the reconstructed volume and where the image plane resides in the volume. This indicates that a non-uniform, dynamic partitioning scheme must be implemented to achieve optimal performance. Additionally, since not all GPUs have the same capabilities or memory configurations, the partitioning scheme cannot be linear in the number of GPUs on the node.

To maximize performance and reduce the amount of network communication between nodes, the master divides the volume prior to computation depending on the overall size of the reconstruction (number of reconstructed image planes) and on the following parameters of each slave:

- **Number of GPUs:** The more GPUs on a node, the greater that node's processing power.
- **Host-to-Device Memory Ratio:** This is total free system memory divided by the sum of the total free device memory across all devices contained within a node. As this ratio increases, so does the total amount of data transfers from host to device prior to the host requiring additional data from the network. Since the transfers across the network are slower than those between GPU and host, we want this ratio maximized. Additionally, when this ratio is too small, the host may not be able to hold enough data to fill the GPU's memory, which hurts GPU utilization and performance. However, this parameter has diminishing returns after a certain point. Specifically, when, for one sub-volume, the time spent transferring data from host to GPU is longer than the time to send data across the network, then an equivalent host system with more GPU memory (smaller ratio) would

perform faster on the same amount of data. In this case, the system with the smaller memory ratio should receive a larger sub-volume. In other words, we do not want systems with an extremely large ratio to dominate the partition.

- **Total Number of GPU Multiprocessors:** More multiprocessors implies there are more available CUDA cores for computing; however, multiple GPUs with a moderate number of multiprocessors is preferred to one GPU with a large number of multiprocessors.
- **Lowest GPU Compute Capability:** An increase in compute capability typically represents an incremental increase in performance. This is the least significant factor in determining the partition.

While there are more parameters we could have used in determining the partition, we isolated what we believed were key, easily accessible factors. Additionally, while this scheme does depend on the total number of image planes being reconstructed, that parameter could be set to a constant, allowing for the cluster to be configured once for multiple reconstructions.

To explain how the variables above are used, assume there are n slave nodes, and let $\vec{p_i}$ be the parameter vector of node i such that

$$\vec{p_i} = \begin{bmatrix} g_i \\ r_i \\ m_i \\ c_i \end{bmatrix}$$

where

- g_i is the number of GPUs.
- r_i is the the ratio of host memory to total GPU memory.
- m_i is the total GPU multiprocessor count.
- c_i is the lowest GPU compute capability.

The master then calculates

$$s = \mathcal{W}P$$

where

- s is a $1 \times n$ row vector such that $s[i]$ is the amount of work assigned to node i.
- $\mathcal{W}$ is a 1×4 user-defined linear or non-linear operator that acts on P.
- $P = \begin{bmatrix} \vec{p_i} & \cdots & \vec{p_n} \end{bmatrix}$.

More specifically, the operator $\mathcal{W}$ is defined as the composition of two user-defined operators,

$$\mathcal{W} = \mathcal{W}_{\mathcal{F}}\mathcal{F}.$$

$\mathcal{F}$ applied to matrix P is a column-wise functional operation to each $\vec{p_i}$ so that

$$\mathcal{F}\vec{p_i} = \begin{bmatrix} f_1\left(g_i\right) \\ f_2\left(r_i\right) \\ f_3\left(m_i\right) \\ f_4\left(c_i\right) \end{bmatrix}.$$

Note that an operator in $\mathcal{F}$ can use information on the total number of reconstructed image plans or information from other nodes. For example, an operator could be a normalization function that normalizes the value with respect to all other nodes.

$\mathcal{W}_{\mathcal{F}}$ applied to $\mathcal{F}\vec{p_i}$ is a 1×4 weighting operator so that

$$\mathcal{W}\vec{p_i} = \mathcal{W}_{\mathcal{F}}\mathcal{F}\vec{p_i} = \begin{bmatrix} w_{\mathcal{F},1} * f_1\left(g_i\right) \\ w_{\mathcal{F},2} * f_2\left(r_i\right) \\ w_{\mathcal{F},3} * f_3\left(m_i\right) \\ w_{\mathcal{F},4} * f_4\left(c_i\right) \end{bmatrix}.$$

Although we could achieve that same weighting scheme without $\mathcal{W}_{\mathcal{F}}$ by incorporating the linear weights into the functions, by separating the two operators, we allow for better understanding and control over the linear and nonlinear aspects of the weighting scheme.

Lastly, we represent the work allocated to node i as a percentage of the total volume. To transform s accordingly, we compute

$$s = \frac{\mathcal{W}P}{\text{rowsum}\left(\mathcal{W}P\right)}.$$

For example, suppose you want to exponentiate the number of multiprocessors, cap the maximum memory ratio at 2, and weight the compute capability as ten times more important than the other parameters. The resulting operators are

$$\mathcal{F} = \begin{bmatrix} f_1 : g_i \mapsto g_i \\ f_2 : r_i \mapsto \max(r_i, 2) \\ f_3 : m_i \mapsto \exp m_i \\ f_4 : c_i \mapsto c_i \end{bmatrix}$$

and

$$\mathcal{W}_{\mathcal{F}} = \begin{bmatrix} 1 & 1 & 1 & 10 \end{bmatrix}.$$

Again, we could allow f_4 to be $c_i \mapsto 10c_i$, but we want to keep the linear and nonlinear components disjoint.

IV. Experimental Evaluation

A. Weighting Setup

We tried three $\mathcal{F}$ operators to determine the optimal partitioning scheme by testing the operators on synthetic cluster configurations. After applying the partitioning algorithm to a variety of configurations, we empirically determined the schemes that were best at balancing the data [12]. For both schemes, the weighting operator $\mathcal{W}_{\mathcal{F}}$, which will to be tuned by the user, made the number of GPUs the most important parameter, followed by the memory ratio, then the number of multiprocessors, and lastly the lowest compute capability. The operator also normalized each parameter. Additionally, since all parameters except the host-to-device memory ratio are expected to linearly affect the processing power of node i, we defined f_1, f_3, and f_4 to be the identity function. f_2, on the other hand, had to be defined differently.

As explained previously, systems with a large amount of host memory but few GPUs give diminishing returns on performance because even though their memory ratio is very large, their processing is constrained by having few GPUs. To account for this and limit the impact of the host-to-device memory ratio on the volume allocation, f_2 was defined as a different non-linear function in each scheme.

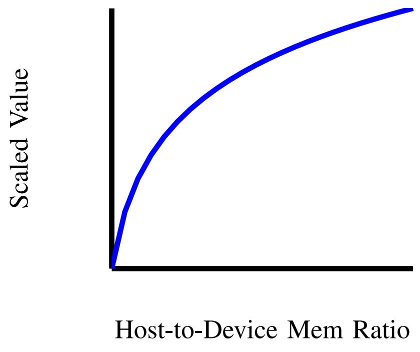

Fig. 3. Log Curve for Scheme 1

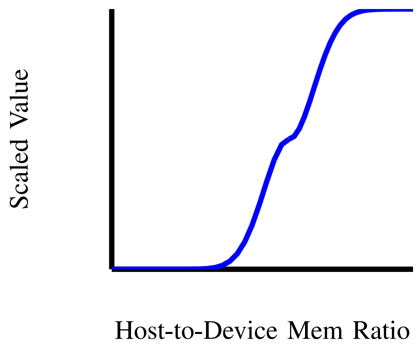

Fig. 4. Double Logistic Curve for Scheme 2

Scheme 1 Used a log function (see Figure 3).

$$f_2(r_i) = a * \log(r_i + b) + c$$

where a, b, and d are user-defined, constant parameters tuned to optimize the scaled value for a given cluster.

Scheme 2 Used a double logistic function (see Figure 4).

$$f_2(r_i) = \text{sign}\,(r_i - d)\left(1 - e^{-\frac{(r_i-d)^2}{s^2}}\right) - a$$

where d, s, and a are user-defined, constant parameters tuned to optimize the scaled value for a given cluster.

Scheme 3 Used a double logistic function like Scheme 2 except we set s to be dependent the total number of image planes reconstructed, denoted as $\#IP$. Specifically, the function is

$$f_2(r_i) = \text{sign}\,(r_i - d)\left(1 - e^{-\frac{(r_i-d)^2}{s(\#IP)^2}}\right) - a$$

where d and a are user-defined parameters. As more image planes are reconstructed, the curve becomes more shallow (see Figure 5 compared to Figure 4). The reason we tried this scheme is because when there are fewer image planes to reconstruct, this ratio becomes less important more quickly. In other words, since the network traffic is less with fewer image planes, a small change in the memory ratio has a more dramatic impact on performance.

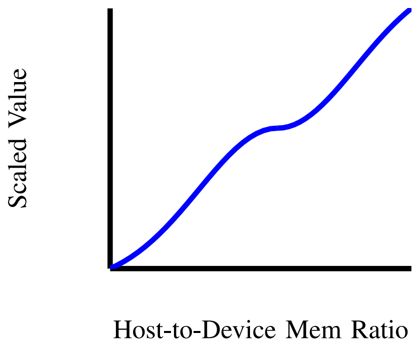

Fig. 5. Double Logistic Curve for Scheme 3

	Node 1	Node 2	Node 3	Node 4
Num GPUs	3	3	4	2
Mem Ratio	12	4	12	8
TotalMP	32	32	52	12
CC	2.0	2.0	2.0	2.0
Scheme 1	26.2	22.7	34.2	16.9
Scheme 2	27.7	21.1	35.7	15.6
Scheme 3 (2k)	26.6	20.6	34.6	18.1
Scheme 3 (10k)	28.3	20.8	36.3	14.6

TABLE I
SYNTHETIC CLUSTER 1 RESULTS

B. *Weighting Results*

Tables 1-4 list the properties of the 4 synthetic, 4 node cluster configurations we used to test our weighting schemes. We used 4 node clusters with each node having compute capability 2.0 to keep the results consistent with those from our experimental evaluation where we also had 4 nodes each with compute capability 2.0 (it remains future work to try GPUs with a variety of compute capabilities). The last four rows of of each table contain the volume percentage allocated to each node. The last two rows are both the volume percentage allocated by scheme 3 with different $\#IP$ values. The first row of scheme 3 has $\#IP = 2,000$ and the second row of scheme 3 has $\#IP = 10,000$.

Across all cluster configurations, the partitioning results are similar, but slight differences can have a large impact on runtime, especially for big data reconstructions. On these Terabyte-sized inputs, reconstruction of a single image plane can take up to a minute. This means a one or two percent volume allocation difference could result in a runtime difference of a few hours.

	Node 1	Node 2	Node 3	Node 4
Num GPUs	4	4	4	4
Mem Ratio	32	16	8	4
TotalMP	52	52	52	52
CC	2.0	2.0	2.0	2.0
Scheme 1	27.9	26.0	24.0	22.1
Scheme 2	28.0	28.0	23.2	20.8
Scheme 3 (2k)	26.6	26.6	26.1	20.6
Scheme 3 (10k)	31.9	28.2	20.6	19.4

TABLE II
SYNTHETIC CLUSTER 2 RESULTS

	Node 1	Node 2	Node 3	Node 4
Num GPUs	1	2	3	4
Mem Ratio	16	4	4	16
TotalMP	52	32	52	64
CC	2.0	2.0	2.0	2.0
Scheme 1	22.4	18.1	24.8	34.6
Scheme 2	24.1	16.5	23.2	36.2
Scheme 3 (2k)	23.9	16.7	23.4	36.0
Scheme 3 (10k)	25.4	15.2	21.9	37.6

TABLE III
SYNTHETIC CLUSTER 3 RESULTS

	Node 1	Node 2	Node 3	Node 4
Num GPUs	3	3	1	8
Mem Ratio	64	16	4	8
TotalMP	32	64	16	128
CC	2.0	2.0	2.0	2.0
Scheme 1	23.6	24.3	9.5	42.6
Scheme 2	22.5	26.8	8.6	42.2
Scheme 3 (2k)	21.1	25.5	8.3	45.1
Scheme 3 (10k)	26.3	27.0	7.1	39.5

TABLE IV
SYNTHETIC CLUSTER 4 RESULTS

Table 1 shows a 3, 3, 4, 2 GPU configuration where Node 4 is the weakest while Node 3 is the strongest with the most GPUs and multiprocessors. Node 1 and 2 are similar with Node 1 having three times the host-to-device memory ratio as Node 2. However, Node 1 does not receive three times the volume as Node 2 since the memory ratio is less significant than the number of GPUs. Node 1 is not going to be able to compute three times the number of image planes as Node 2 because only having 3 GPUs is the bottleneck in its computation. Lastly, the difference between scheme 3 (2k) and scheme 3 (10k) is most notable in Node 4. With 2k image planes, Node 4 receives 18.1 percent of the volume, while with 10k image planes, it receives 14.6 percent. With only 2k image planes to reconstruct, the relative slowness of transferring data across the network is more noticeable, making Node 4's larger memory ratio more beneficial that with 10k image planes.

Table 2 shows a configuration where all nodes have 4 GPUs and the same number of multiprocessors. The only factor differentiating them is the host-to-device memory ratio (Node 1 has the greatest and Node 4 the least). Again, even though Node 3 has double the memory ratio as Node 4, the number of GPUs is the dominating factor; so, Node 3 does not receive double the volume as Node 4. Scheme 3 (10k) has the widest range of allocations with Node 4 receiving 19.4 percent and Node 1 receiving 31.9 percent, which makes sense given the large number of image planes to reconstruct. Scheme 1 has the narrowest allocation range with Node 4 receiving 22.1 percent and Node 1 receiving 27.9 percent, making it a poor choice for large reconstructions where Node 4 would lag behind.

Table 3 shows a 1, 2, 3, 4 GPU configuration. Scheme 2 and 3 (2k) are nearly identical in their volume allocations. Scheme 3 (10k) allocates less volume to Node 2 and 3 and more to Node 1 and 4 than the other schemes because Node 1 and 4's high memory ratios are becoming more significant with the larger number of image planes. Although schemes 1, 2, and 3 are slightly more uniform in their allocation, with a large number of image planes, scheme 3 (10k) will likely be more efficient.

Table 4 shows a 3, 3, 1, 8 GPU configuration where Node 3 has 1 GPU and a memory ratio of only 4 while Node 4 has 8 GPUs with a memory ratio of 8. We also see Node 1 and 2 both have 3 GPUs with a memory ratio of 64 and 16 respectively. Even though Node 1 has four times the memory ratio as Node 2, it actually receives less volume than Node 2. This is because both of their memory ratios are extremely large; so, we see little to no performance benefit from giving more volume to Node 1. In fact, Node 1's lower number of multiprocessors, 32 compared to Node 2's 64, will hurt its performance more. We see that Node 4 receives the most volume in all schemes, which is as expected given its high number of GPUs.

Overall, these schemes better partition the data than a uniform partition, but scheme 3 is the one we think best takes into account each node's capabilities. Scheme 3 also varies the partition depending on the number of image planes to reconstruct, which will improve performance and increase the partitioning scheme's ability to balance the work for a variety of heterogeneous clusters. Therefore, we used scheme 3 to partition our volume in our experimental evaluations.

C. *Cluster Setup*

We used 5 nodes (1 master and 4 slaves) with a total of 14 Nvidia Tesla GPUs. The 14 Tesla GPUs consisted of 8 M2090 devices (512 CUDA cores/6GB RAM), 4 C2070 devices (448 CUDA cores/ 6GB RAM), and 2 C2050 devices (448 CUDA cores/ 3GB RAM). As mentioned previously, all of these graphic cards had compute capability 2.0; so, we were unable to test its impact on performance.

For our one node I/O implementations, (one node was doing all the I/O as in Figure 1), we used an 8x HDD RAID0 array. For our two node implementation (separate nodes for reading input and writing output as in Figure 2), we used an 8x HDD RAID0 array for input I/O and a 3x HDD RAID0 array for output. All HDDs used in the RAID arrays were 3TB. The total cluster memory was approximately 1TB, unevenly distributed between the master and slave nodes.

We used CUDA version 5.0, MPICH [13], and OpenMP version 2.0. Two of the nodes used Microsoft Server 2012 and the remaining three used Microsoft Server 2008. We used a synthetic Teravoxel dataset made up of 10,000 100 Megapixel projections (1 trillion voxels). We ran our reconstruction using a uniform partitioning scheme where all nodes get the same percentage of image planes (25 percent) and using scheme 3 as described above with one modification. Due to previous work showing the reconstruction algorithm does not scale well beyond 4 GPUs [7], we set $f_1 = \max(g_i, 4)$ (g_i is the number of GPus on node i).

D. *Experimental Results*

To test our implementation, we created two clusters. The cluster shown in Table 5 had a 5, 1, 3, 4 GPU configuration with Node 2 and 4 having the highest host-to-device memory ratio. Node 1 had the largest number of GPUs, 5, but the

	Node 1	Node 2	Node 3	Node 4
Num GPUs	5	1	3	4
Memory Ratio	5.8	21.6	7.9	29.9
TotalMultiProc	74	14	46	64
Compute Capability	2.0	2.0	2.0	2.0
Uniform Partitioning	25.0	25.0	25.0	25.0
Weighted Partitioning	32.1	10.2	24.3	33.3

TABLE V
EXPERIMENTAL CLUSTER 1

	Node 1	Node 2	Node 3	Node 4
Num GPUs	4	3	3	4
Memory Ratio	8.4	7.4	7.9	29.9
TotalMultiProc	56	46	46	64
Compute Capability	2.0	2.0	2.0	2.0
Uniform Partitioning	25.0	25.0	25.0	25.0
Weighted Partitioning	25.4	21.4	20.7	32.3

TABLE VI
EXPERIMENTAL CLUSTER 2

lowest memory ratio, 5.8. Our dynamic partitioning scheme allocated the smallest volume to Node 2, 10.2 percent and the largest to Node 4, 33.3 percent. Even though Node 4 had 4 GPUs compared to Node 1's 5, it had a significantly higher memory ratio, meaning it should get more of the volume.

The cluster shown in Table 6 had a 4, 3, 3, 4 GPU configuration with Node 1, 2, and 3 all having a memory ratio of around 8. Node 4 had a ratio of 29.9, meaning it received the most volume since it also had the largest number of GPUs. Node 1 had a slightly higher memory ratio and number of multiprocessors than Node 2 and 3; so, it received a little more volume to reconstruct.

Our experimental results are shown in Table 7. We tested our one node I/O implementation on both clusters and our two node implementation on the second cluster. We wanted to compare the one and two node I/O implementations on a cluster that stressed the network, and the second cluster, being slightly more uniform in volume allocation, does this more than the first cluster.

Our results show that we more than double performance from using a uniform partitioning scheme to our weighted partitioning scheme. In all cases, the reconstruction took over 30 hours for the uniform partitioning. Due to time limitations, we chose not to let the reconstruction run until completion. Cluster 1 and cluster 2 both had similar runtimes for the one node I/O, weighted implementation, further showing the weighting scheme's adaptability. There was a 9 percent increase in performance using one node for I/O versus two nodes. This shows that while reducing the I/O bottleneck at the master node does improve performance, the benefit is not as significant as using a better load balancing scheme.

	Uniform	Weighted
Cluster 1 with 1 Node I/O	>30hrs	14.1hrs
Cluster 2 with 1 Node I/O	>30hrs	14.2hrs
Cluster 2 with 2 Node I/O	>30hrs	12.9hrs

TABLE VII
RUNTIME RESULTS

V. CONCLUSION

This work has shown that future-sized, big data CT reconstruction can be done in a feasible amount of time. By implementing a modularized algorithm for reconstruction, taking advantage of the highly parallel architecture of the GPU, and utilizing the compute power of a cluster, we were able to significantly reduce the runtime for reconstruction. Further, by making the cluster consist of heterogeneous, GPU-capable workstations, we have allowed a variety of users and NDE labs with different workstations to run CT reconstruction without needing expensive and specialized hardware. This means CT can potentially be used as a standard quality control measure due to the improved reconstruction performance.

Although our work has better prepared the scientific community to handle future-sized CT reconstruction, there are many problems that still need to be addressed. Can we take the energy efficiency results from [12], where they showed a single node, GPU-based reconstruction to be more energy efficient than one without GPUs, and show them to be true to our approach? For other future work, we want to test our load balancing scheme on a wider variety of clusters where nodes have differing compute capabilities. We would also research making the cluster configuration automated. Instead of the parameters being hand-tuned, which takes time and skill, we would devise an algorithm to determine the parameters which work best for a particular cluster.

We also want to look at distributing the data using a distributed file system to alleviate the network congestion caused by the massive amount of input and output data being transfered across the network. By using a distributed file system, we would also pave the way to compute a reconstruction on "the cloud", which eliminates the need to purchase hardware and GPUs.

We believe this work has a significant impact on the industrial imaging community and is a first step towards near-real-time CT reconstruction.

VI. ACKNOWLEDGMENTS

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] H. H. Barrett and K. J. Myers, *Foundations of Image Science*. Wiley-Interscience, 2004.

[2] W. mei W. Hwu, Ed., *GPU Computing Gems - Emerald Edition*. Morgan Kaufmann, 2011.

[3] O. Cossairt, D. Miau, and S. Nayar, "Gigapixel computational imaging," in *Computational Photography (ICCP), 2011 IEEE International Conference on*, 2011, pp. 1–8.

[4] R. Kohley, P. Gaé, C. Vétel, D. Marchais, and F. Chassat, "Gaia's fpa: Sampling the sky in silicon," in *Proceedings of SPIE, Space Telescopes and Instrumentation: Optical, Infrared, and Millimeter Wave*, vol. 8442. SPIE, 2012.

[5] N. K. Dhar and R. Dat, "Advanced imaging research and development at darpa," in *Proceedings of SPIE, Infrared Technology and Applications XXXVIII*, vol. 8353. SPIE, 2012.

[6] D. J. Brady, M. E. Gehm, R. A. Stack, D. L. Marks, D. S. Kittle, D. R. Golish, E. M. Vera, and S. D. Feller, “Multiscale gigapixel photography,” *Nature*, vol. 486, pp. 386–389, 2012.

[7] L. J. Orr and E. S. Jimenez, “Preparing for the 100-megapixel detector: Reconstructing a multi-terabyte computed-tomography dataset,” in *SPIE Optical Engineering+Applications*. International Society for Optics and Photonics, 2013.

[8] E. S. Jimenez, L. J. Orr, K. R. Thompson, and R. Park, “A high-performance and energy-efficient ct reconstruction algorithm to reconstruct multi-terabyte datasets,” in *Conference Proceedings for the IEEE Nuclear Science Symposium and Medical Imaging Conference, NSS-MIC 2013*. IEEE Computer Society, 2013.

[9] L. Feldkamp, L. Davis, and J. Kress, “Practical cone-beam algorithm,” *Journal of the Optical Society of America A*, vol. 1, 1984.

[10] E. S. Jimenez, L. J. Orr, and K. R. Thompson, “An irregular approach to large-scale computed tomography on multiple graphics processors improves voxel processing throughput,” in *Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. IEEE Computer Society, 2012, pp. 254–260.

[11] J. Deng, “Parallel computing techniques for computed tomography,” Ph.D. dissertation, University of Iowa, 2011.

[12] E. S. Jimenez, E. L. Goodman, R. Park, L. J. Orr, and K. R. Thompson, “Irregular large-scale computed tomography on multiple graphics processors improves energy-efficiency metrics for industrial applications,” in *SPIE Optical Engineering+ Applications*. International Society for Optics and Photonics, 2014.

[13] “MPICH: High-Performance Portable MPI,” http://www.mpich.org/.