Iterative Helical CT Reconstruction in the Cloud for Ten Dollars in Five Minutes

Jeffrey M. Rosen, Junjie Wu, Jeffrey A. Fessler, Thomas F. Wenisch
Department of EECS, University of Michigan

***Abstract*—Iterative statistical X-ray CT reconstruction algorithms can improve image quality for low dose scans. Unfortunately, their clinical utility has been hampered by their enormous computational requirements; typical low-dose reconstructions require about an hour on commercial systems. Most existing parallel implementations use a shared memory programming model, limiting available parallelism. We investigate using a large compute cluster for a penalized weighted least-squares algorithm using ordered subsets (PWLS-OS), scaled to hundreds of cores to accelerate a single helical CT reconstruction problem. Using Amazon's Elastic Cloud Compute (EC2) service, our experimental results show that a typical helical chest scan can be reconstructed in under five minutes at a cost under $10.**

I. Introduction

Model-based iterative reconstruction for X-ray CT can improve image quality and promises to enable X-ray dose reductions compared to conventional filtered back-projection [3]. Such methods use statistical models and imaging system models, improving image quality. The primary drawback of statistical reconstruction methods is their massive computational requirement. Current commercial model-based reconstruction methods can require about an hour to reconstruct a typical helical chest scan. Improving reconstruction times is essential to enable ubiquitous use of low-dose CT.

Researchers are developing reconstruction algorithms that reduce computational requirements and/or converge more quickly. For example, one recent ordered-subsets algorithm reaches its limit cycle in about 20 iterations [1]. Nevertheless, compute-times-per-iteration remain high (several minutes for helical chest CT scans), so matching scanner and reconstruction throughput requires further improvements.

Parallelization can reduce time-per-iteration, leveraging the multiple cores present in modern processors by partitioning computation into multiple simultaneous sub-problems [4]. Most existing parallel implementations share image and sinogram data in a global main memory accessible to all processor cores. Shared memory simplifies parallelization—each core computes a subset of the image/sinogram, and can read from any part of the image/sinogram space with only infrequent synchronization at coarse steps of each iteration. However, shared memory approaches are limited by the number of cores that can be provisioned in a single system—at most a few tens of cores in conventional commodity systems.

In this work, we investigate the alternative of leveraging the scalability of massive compute clusters to apply distributed computing power to image reconstruction. We demonstrate distributed image reconstruction using leased resources from a commercial cloud computing provider. Cloud services provide low-cost, on-demand, commodity computing resources. They are relatively cheap when compared to purchasing a cluster, and can either be used on-demand for flexibility or reserved for exclusive use and further discounted costs. They also provide a low-overhead mechanism for expanding computational power. To increase the number of nodes working on a problem, one need only purchase more cloud compute time (only seconds of setup time).

Although we use the cloud to demonstrate the performance potential of distributed reconstruction, our methods are applicable more generally to all distributed systems. For example, researchers have used accelerated reconstruction using the parallelism in graphics processing unit (GPU) accelerators [5]. However, ganging multiple GPUs for greater parallelism presents a significant challenge because they do not share a single global address space, hence data dependencies are problematic. Our methods could be applied to such a system to sub-divide processing across distributed GPUs. Similarly, emerging devices such as the Xeon Phi coprocessor can be ganged together to achieve greater performance and scalability using our approach.

To use cloud resources, one must parallelize reconstruction algorithms across compute nodes that do not share a single global memory. We follow the paradigm of many large-scale scientific applications by using explicit *message passing* to exchange updates to sinogram and image data between compute nodes at appropriate synchronization points in the reconstruction algorithm. As we will show, we can easily scale the number of nodes collaborating on a single reconstruction problem until performance is limited by available communication bandwidth; further speedups will require either faster (and more expensive) interconnection networks or innovations to reduce data communication.

Prior efforts to parallelize filtered back-projection over a cluster have used the MapReduce programming model [6, 7], wherein the computation is translated into simple "map" and "reduce" tasks and a runtime system orchestrates communication among these tasks. Though they ease programming, publicly available MapReduce frameworks, such as Hadoop, store intermediate results on disk when communicating, which is extremely inefficient for iterative reconstruction algorithms. To our knowledge, there has been no prior reports of iterative reconstruction of clinical helical CT scans (with thousands of projection views) using hundreds of cores on a commodity cloud computing service. The closest related work is the

JAF supported in part by NIH grant R01 HL 098686 and by equipment donations from Intel.

The authors thank Donghwan Kim for assistance with the algorithms in [1, 2].

investigation by Gregor of an unregularized SIRT algorithm on four 8-core nodes for axial micro CT with 360 views [8].

This paper investigates a penalized weighted least-squares with ordered subsets (PWLS-OS) reconstruction algorithm [2] that distributes computation across several multi-core nodes that communicate via explicit message passing. This approach scales beyond the limits of a single node, so the number of cores working in parallel is limited only by available hardware, communication bandwidth, and cost. We use Amazon's Elastic Compute Cloud (EC2) service to demonstrate the potential of cloud computing environments and show that a 20-iteration reconstruction of a 320-slice helical chest CT scan using 50 nodes (800 cores) requires less than five minutes at a total computing cost under $10. A "private" cloud computing environment (e.g., operated under contract for a large hospital) might approach similar costs.

II. Methods

A. Background

We compute a reconstructed image $\hat{\mathbf{x}}$ by minimizing a PWLS cost function [3]:

$$\hat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmax}} \frac{1}{2} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|^2_{\mathbf{W}} + R(\mathbf{x}), \tag{1}$$

where $\mathbf{y}$ denotes the observed X-ray CT sinogram data, $\mathbf{W}$ denotes a diagonal statistical weighting matrix, $\mathbf{A}$ is the system matrix [9], and $R(\mathbf{x})$ is an edge-preserving regularizer that balances noise and image resolution. We use an $R(\mathbf{x})$ with first-order finite differences between a voxel and its closest 26 neighbors and a Fair edge-preserving potential [10], but our methods can be extended to other regularizers.

The PWLS-OS iterative algorithm [4] involves four steps: forward projection, back projection, regularization, and image update. Figure 1 depicts the algorithm graphically. The first step involves forward projecting the estimate $\mathbf{x}_n$ at the nth iteration and calculating the weighted sinogram residual:

$$\mathbf{r}_n = \mathbf{W}(\mathbf{A}\mathbf{x}_n - \mathbf{y}). \tag{2}$$

The calculations for each residual are independent, so they can be arbitrarily reordered or parallelized [9]. Iterating along the scan axis in the innermost loop enables reuse of beam geometry calculations. Only the projection views within a given subset are projected in a given sub-iteration, so methods that use fewer subsets, and hence more views per subset, *e.g.*, [1], provide more opportunity for parallelism.

Back projection applies the transpose of the system matrix

$$\mathbf{b}_n = \mathbf{A}'\mathbf{r}_n. \tag{3}$$

Back projection must occur after forward projection because of its dependency on $\mathbf{r}_n$. In principle, one can backproject the residual into every voxel independently, allowing massive parallelism over voxel space. In practice it is again more efficient to have an inner loop along the axial direction for each thread [9]. This strategy leads to parallelization across the $\approx 512^2$ voxels in a single transaxial slice, which still allows for tens of thousands of threads.

The regularization step calculates the gradient ∇ of the regularizer at the current image $\mathbf{x}_n$:

$$\mathbf{g}_n = \nabla R(\mathbf{x}_n). \tag{4}$$

Because it depends only on $\mathbf{x}_n$, one can perform regularization in parallel with back projection and in any voxel order.

Finally, the image is updated as follows:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \mathbf{D}(\mathbf{b}_n + \mathbf{g}_n), \tag{5}$$

where $\mathbf{D}$ denotes a diagonal matrix that is precomputed prior to iterating using optimization transfer principles [2]. We also enforce non-negativity in this step.

Parallelism is readily available in each of these steps; in principle, one could launch individual execution threads to calculate each element in $\mathbf{x}_n$, $\mathbf{r}_n$, $\mathbf{b}_n$, and $\mathbf{g}_n$. In practice, it is more efficient to group calculations into threads that allow common sub-expressions to be factored out of inner-most loops. Most existing statistical reconstruction implementations [2, 5] use programming interfaces such as POSIX threads, which allow concurrent operation on a single copy of $\mathbf{x}_n$, $\mathbf{r}_n$, $\mathbf{b}_n$, and $\mathbf{g}_n$ stored in a shared main memory. As each thread updates a disjoint subset of each matrix, the threads can proceed without synchronization, except for a global barrier between each step.

Though shared main memory provides a simple abstraction, it limits performance scalability. Far greater performance can be achieved by scaling a workload to execute on a large cluster. Distributed computation in a cluster is particularly cost-effective in cloud computing environments, where clusters can be time-shared and compute time leased by the hour at low cost.

B. Parallelizing over a cluster

A large cluster can bring far more compute cores to bear on a problem, but data updates must be transmitted explicitly between compute nodes. The central challenge of implementing statistical reconstruction on such a cluster lies in orchestrating this communication, requiring a fundamentally different implementation approach than conventional parallel statistical reconstruction.

Unfortunately, data structures are not easily partitioned and distributed among nodes in statistical reconstruction algorithms for X-ray CT. Both forward and back projection require essentially all data in a particular transaxial slice. Data can be somewhat partitioned along the scan axis, particularly for small cone angles that limit the axial interaction distance between image data and views. For a helical CT chest scan with 320 slices, we partition the image volume into, say, 5 "slabs" of 64 slices each and reconstruct each of those slabs by an independent set of compute nodes. Because of the "long object problem" in helical CT, to reconstruct 64 slices of interest, we also reconstruct 32 padding slices on each end of the slab. These padding slices are discarded after reconstruction, and thus represent a somewhat undesirable overhead of the slab-partition approach. This overhead means that it is inefficient to partition the volume into many more slabs with fewer slices per slab.

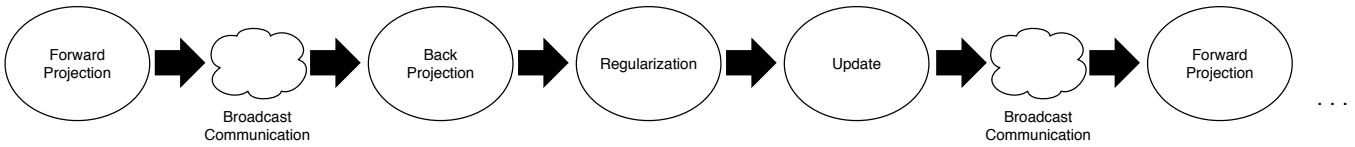

Fig. 1: Visual representation of computation phases. Arrows represent global barriers between steps.

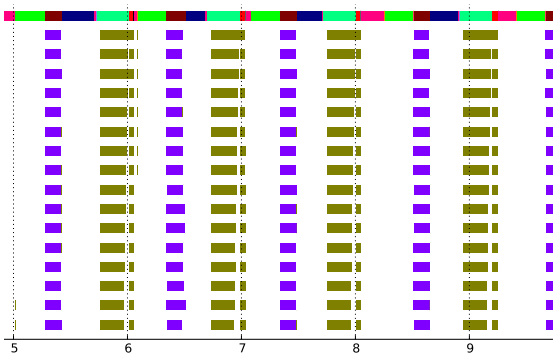

Fig. 2: Sample execution timeline illustrating phases of computation and communication. The color of Line 1 indicates the current algorithmic step. Dark red indicates forward projection, dark blue indicates communication after forward projection, light green indicates back projection, light red indicates regularization, purple indicates update, neon green indicates communication after the update step, and pink indicates barrier synchronization. The remaining lines indicate forward progress of individual threads on each core.

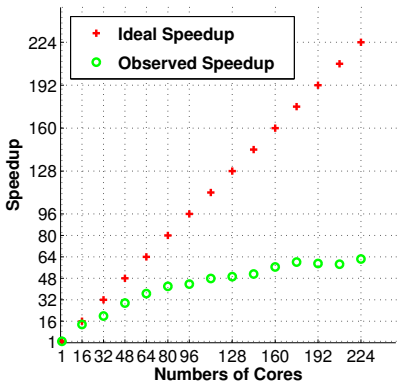

Fig. 3: Speedup of a single iteration of one slab for varying node configurations.

After each algorithm step, image/sinogram updates are broadcast and merged with results from all other nodes participating in the slab computation. We use the Message Passing Interface (MPI) for inter-node communication. MPI provides a means of sending and receiving data both synchronously (blocking) and asynchronously (non-blocking), as well as creating global barriers that prevent any node from proceeding past the point of the barrier until all nodes have reached it.

A straightforward communication approach places a significant burden on the interconnection network between servers. At the end of any given step, updating each node's copy of $\mathbf{x}$ requires each node to broadcast a copy of its portion of the data to $N-1$ other nodes, where N is the number of nodes participating in a given slab's reconstruction. An entire X-ray CT image volume for a helical scan can occupy about a gigabyte of memory (in single float precision). With a network bandwidth of 10 gigabits per second (as in Amazon's EC2 system), transferring several copies of the entire image volume incurs considerable delay. Communication is needed multiple times in each subset and iteration, so the total time spent sending and receiving over the network ultimately limits performance scalability. Hence, optimizing communication is critical. Partitioning the problem into smaller-sized slabs is a first step towards reducing communication. We also broadcast only those arrays that must be synchronized, which are the residual $\mathbf{r}_n$ and the updated image $\mathbf{x}_{n+1}$.

Figure 1 illustrates the reconstruction steps for a single subset. The arrows represent global barriers between each step, and the clouds represent the two necessary broadcast steps.

Figure 2 illustrates the computation timeline over about four (out of 12) subset updates, derived from measured results for execution on 10 nodes of 16 cores each. Each colored segment of the top bar represents an individual task, and the segment length indicates the amount of time taken for that task (in seconds). The 16 bottom bars show execution time for each core in the first node to perform the task identified by the color in the topmost bar. Synchronization and communication time (dark blue, pink, and neon green in the top bar; large blank regions in the remaining bars) occupy a significant fraction of overall execution time.

III. Experimental Results

We report on our distributed version of PWLS-OS implemented in the C99 programming language using the openMPI and POSIX thread libraries. Our implementation produces identical output to that of a previously existing multithreaded version, confirming our methods do not sacrifice image quality.

Our test environment consisted of Amazon EC2 HPC cc2.8xlarge nodes, each having dual eight-core 2.6 GHz Xeon processors, 60.5 GB of memory, and 10 gigabit ethernet. We used Amazon's group placement policy for all experiments to ensure nodes were located physically close to each other.

We used simulated helical CT data where the image volume for a single slab is $512 \times 512 \times 128$ slices with a 70 cm transaxial field of view (FOV) and 0.625 mm slice thickness. Out of an entire 9-turn helical scan, with pitch = 63/64, we used 3 turns (2952 views) for reconstructing each slab. The views were each 64 rows by 888 channels. We use the separable footprint projector [9]. Voxels within the 70 cm FOV are reconstructed using [2]. We used 12 subsets because that is suitable for our latest accelerated OS algorithm [1].

Figure 3 shows the computational speedup for a single iteration plotted for a single core up to 224 cores (each node contains 16 cores) for our implementation compared to ideal (linear) speedup. Speedup [11] for a system with n cores is

$$\frac{\text{Time taken for 1 core}}{\text{Time taken for } n \text{ cores}}. \tag{6}$$

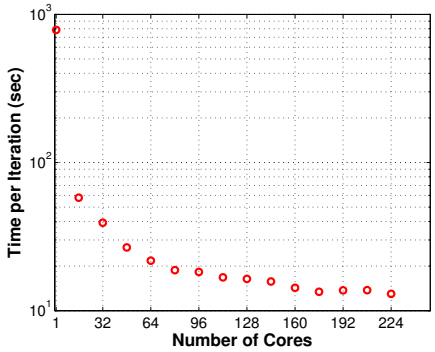

Fig. 4: Timing results for a single iteration on one slab for varying node configurations.

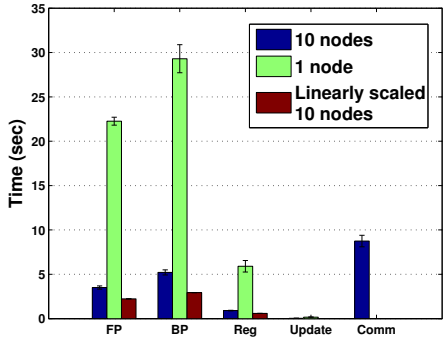

Fig. 5: Time (per iteration) spent in each step for one slab in the 10-node and 1-node cases compared to ideal linear scaling.

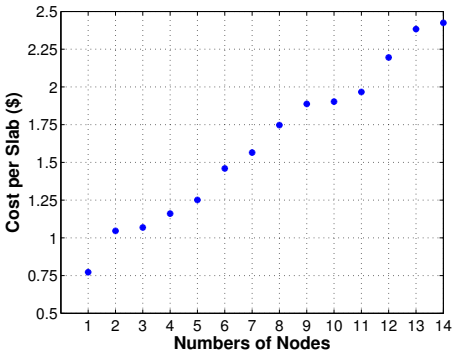

Fig. 6: Cost for a single slab of a 20-iteration scan for varying node configurations.

Figure 3 shows that the observed speedup growth slows as nodes are added due to communication time and bandwidth constraints. This behavior results in a knee in the speedup curve as it approaches a maximum speedup of about 64.

Figure 4 depicts the time taken to complete a single iteration for 1 core to 224 cores in 1-node increments. As expected, the time curve decreases more and more slowly as the number of nodes increases, asymptotically approaching a minimum of about 12 seconds. Communication begins to dominate around 160 cores (10 nodes), at which point the benefit of using more nodes becomes insignificant.

Figure 5 shows computation times of individual steps for the observed 10-node, 1-node, and ideal linear scaling 10-node cases. Linear scaling for n nodes is defined as achieving a speedup of n, so a linearly scaled 10 node system would take one tenth of the time of the 1 node case per iteration. The actual time taken in each compute step (blue) is reasonably close to the ideal linearly scaling case (red). Thus individual steps scale well even though total iteration time exhibits much less than linear speedup due to communication time. The significance of communication time is evident; it accounts for nearly half the time per iteration in the 10-node case.

Figure 6 plots the cost of running 20 PWLS-OS iterations to reconstruct a single 128-slice slab versus the number of nodes used. The cost increases monotonically because communication is such a significant factor in performance. The total cost, however, is still inexpensive.

Based on diminishing returns when using more than 10 nodes, we focus on the 10-node case as a reasonable configuration. Using 10 nodes (160 cores) per slab, the total time for reconstructing a 5-slab scan (320 usable slices) is $\frac{15 \text{ sec}}{\text{iteration}} \times 20 \text{ iterations} = 300$ seconds. Likewise, the cost of performing a reconstruction is $\frac{10 \text{ nodes}}{\text{slab}} \times 5 \text{ slabs} \times \frac{\$0.00067}{\text{sec}} \times \frac{15 \text{ sec}}{\text{iteration}} \times 20 \text{ iterations} = \$10$. For a longer helical scan with 640 slices the cost would scale to \$20 (by using more nodes for the additional slabs) but the 5-minute reconstruction time would remain unchanged.

IV. Summary and Conclusions

Use of statistical reconstruction methods is impeded by their computation time. We have investigated using commercial cloud computing to improve the speed of MBIR through parallel computing. Our results, generated using Amazon's EC2 service, show that even with significant communication overhead, attractive reconstruction times (5 minutes) can be achieved at a low price (\$10). If high resolution targeting of a region of interest (ROI) is needed, then a two-stage reconstruction will be needed [12] that would increase the time and cost accordingly. As expected, node-to-node communication is a limiting factor on performance, even for a relatively small number of nodes. Future work includes reducing communication by using data compression techniques, and by devising iterative algorithms that do not require full synchronization after every update.

References

[1] D. Kim, S. Ramani, and J. A. Fessler. Ordered subsets with momentum for accelerated X-ray CT image reconstruction. In *Proc. IEEE Conf. Acoust. Speech Sig. Proc.*, 2013. Submitted as 4595.

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

[3] 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.*, 34(11):4526–44, November 2007.

[4] D. Kim and J. A. Fessler. Parallelizable algorithms for X-ray CT image reconstruction with spatially non-uniform updates. In *Proc. 2nd Intl. Mtg. on image formation in X-ray CT*, pages 33–6, 2012.

[5] M. Wu and J. A. Fessler. GPU acceleration of 3D forward and backward projection using separable footprints for X-ray CT image reconstruction. In *Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med*, pages 56–9, 2011.

[6] B. Meng, G. Pratx, and L. Xing. Ultrafast and scalable cone-beam CT reconstruction using MapReduce in a cloud computing environment. *Med. Phys.*, 38(12):6603–9, December 2011.

[7] S. Srivastava, A. R. Rao, and V. Sheinin. Accelerating statistical image reconstruction algorithms for fan-beam x-ray CT using cloud computing. In *Proc. SPIE 7961 Medical Imaging 2011: Phys. Med. Im.*, page 796134, 2011.

[8] J. Gregor. Distributed multi-core implementation of SIRT with vectorized matrix kernel for micro-CT. In *Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med*, pages 64–7, 2011.

[9] Y. Long, J. A. Fessler, and J. M. Balter. 3D forward and back-projection for X-ray CT using separable footprints. *IEEE Trans. Med. Imag.*, 29(11):1839–50, November 2010.

[10] R. C. Fair. On the robust estimation of econometric models. *Ann. Econ. Social Measurement*, 2:667–77, October 1974.

[11] J. L. Hennessy and D. A. Patterson. *Computer architecture: A quantitative approach*. Morgan Kaufmann, San Francisco, 4 edition, 2006.

[12] A. Ziegler, T. Nielsen, and M. Grass. Iterative reconstruction of a region of interest for transmission tomography. In *Proc. SPIE 6142 Medical Imaging 2006: Phys. Med. Im.*, page 614223, 2006.