Eurographics Symposium on Parallel Graphics and Visualization (2011)
T. Kuhlen, R. Pajarola, and K. Zhou (Editors)

Parallel Gradient Domain Processing of Massive Images

S. Philip[1], B. Summa[1], P.-T. Bremer[1,2] and V. Pascucci[1]

[1]Scientific Computing and Imaging Institute, University of Utah
[2]Lawrence Livermore National Labratory

Abstract

Gradient domain processing remains a particularly computationally expensive technique even for relatively small images. When images become massive in size, giga or terapixel, these problems become particularly troublesome and the best serial techniques take on the order of hours or days to compute a solution. In this paper, we provide a simple framework for the parallel gradient domain processing. Specifically, we provide a parallel out-of-core method for the seamless stitching of gigapixel panoramas in a parallel MPI environment. Unlike existing techniques, the framework provides both a straightforward implementation, maintains strict control over the required/allocated resources, and makes no assumptions on the speed of convergence to an acceptable image. Furthermore, the approach shows good weak/strong scaling from several to hundreds of cores and runs on a variety of hardware.

Categories and Subject Descriptors (according to ACM CCS): I.3.2 [Computer Graphics]: Graphics Systems—Distributed/network graphics I.3.3 [Computer Graphics]: Picture/Image Generation—

1. Introduction

Recently, cheap high-resolution cameras and inexpensive robots to automatically capture image collections [Gigin], have become increasingly accessible. As a result, mosaics of hundreds of individual images are readily available online captured by professionals and nonprofessionals alike. Larger images, many gigapixels in size, are freely distributed online such as satellite imagery from the United States Geological Survey (USGS) [USGin] and planetary images from the High Resolution Imaging Science Experiment (HiRISE) [HiRin]. Furthermore, all indications point to a continuing increase in the interest toward such large images. Thus, research into processing and handling of these large datasets is necessary as the size of current images already stresses the capabilities of modern methods.

Images acquired with inexpensive robots and consumer cameras pose an interesting challenge for the image processing community. Often, panorama robots can take seconds between each photograph, causing gigapixel-sized images to be taken over the course of hours. Due to this delay, images can vary significantly in lighting conditions and/or exposure and when registered can form an unappealing patchwork. Images acquired by air or satellite also suffer from an extreme version of this problem, where the time of acquisition can vary from hours to days for a single composite. Creating a single seamless image from this mosaic has been the subject of a large body of work, for which gradient-domain techniques currently provide the best solution.

Two methods exist to operate on the gradient-domain of massive images: the streaming multigrid [KH08] and progressive Poisson [SSJ*10] techniques. Although efficient in their implementations, each can still take on the order of hours to compute the solution for a gigapixel image on a single system. One option to improve these timings is to see if similar schemes can be designed to run in a distributed environment. Consequently, there has been recent work to extend the multigrid solver [KSH10] to a parallel implementation, reducing the time to compute a gigapixel solution to mere minutes for the first time. However, this approach is primarily a proof-of-concept since It does not supply the classic tests of scalability (weak or strong) nor is it tested significantly. The implementation was optimized for a single distributed system and therefore is unlikely to port well to other environments. Furthermore, like many out-of-core methods, proliferation of disk storage requirements is a major drawback. For example, testing was only possible with a full 16-node cluster for some of the paper's test data due to excessive storage demands. Finally, the technique assumes a small number of predetermined iterations is sufficient to achieve a

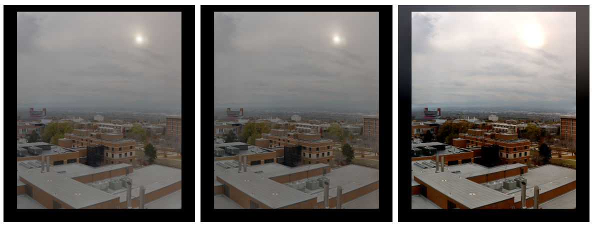


Figure 1: *A comparison of a full analytical solution (left), the progressive Poisson [SSJ*10] (middle), the streaming multigrid method [KH08] (right) on a portion of the Fall panorama,* 21201×24001 *pixels, 485-megapixel. Note how the progressive Poisson method's solution is almost indistinguishable from the full analytical solution.*

solution which may not always be the case as shown in Figure 1 (right).

In this paper we introduce a framework for parallel gradient-domain processing inspired by Summa et al.'s [SSJ*10] progressive Poisson solver. Although an inherently sequential, windowed, out-of-core scheme, we show how a novel reformulation can provide an efficient parallel distributed algorithm. This new framework has both a straightforward implementation and shows both strong and weak parallel scalability. When implemented in standard MPI (Message Passing Interface), the same code base ports well to multiple distributed systems. The parallel progressive Poisson solver also inherits all of the benefits from the previous work. In particular, it allows strict control over required resources which can be especially important when using mixed hardware or modestly provisioned distributed systems. It also makes no assumptions about the number of iterations necessary for an adequate solution. In the following sections, we will detail the algorithm and its MPI implementation as well as show the method both weak and strong scaling from few to many cores on multiple distributed systems.

2. Related Work

Poisson Image Processing. Gradient-based methods, though computationally expensive, have become a fundamental part of any advanced image editing application. Given a guiding gradient field constructed from one or multiple source images, these methods attempt to find a smooth image that is closest in a least squares sense to this guiding gradient. This concept has been adapted for seamless cloning [PGB03], drag-and-drop pasting [JSTS06] as well as matting [SJTS04]. Furthermore, gradient-based techniques can reduce the range of HDR (High Dynamic Range) images for display on standard monitors [FLW02] or hide the seams in panoramas [PGB03, LZPW04, ADA*04, KH08, KSH10]. Other applications include detecting lighting [Hor74] or shapes from images [Wei01], shadow removal [FHD02] or reflections [ARNL05], and artistic editing in the gradient domain [MP08]. An alternative to gradient based methods using Mean-Value Coordinates (MVC) has been introduced to smoothly interpolate the boundary difference between images in order to mimic Dirichlet boundary conditions [FHL*09]. Currently MVC has not been shown to extend well to distributed systems for applications such as panorama stitching due to the dependency between solved images and their unsolved neighbors.

Poisson Solution. Gradient based image processing typically requires the solution to a 2D Poisson problem. Computing the solution to Poisson equations efficiently, in parallel, or on distributed systems has been the focus of a large body of work; therefore we only offer a cursory review in this paper. Methods exist to find a direct Poisson solution using Fast Fourier Transforms (FFT) [Hoc65, ACR05, ARC06, MP08]. FFT methods are inherently global requiring a two full image transformations which need special formulations for parallel computation. These methods have not yet been shown to work well out-of-core or in parallel for gradient domain image processing when compared to state-of-the-art techniques. Often the Poisson problem is simplified by discretization into a large linear system whose dimension is typically the number of pixels in an image. Methods exist to find a direct solution to this linear system. We refer the reader to Dorr [Dor70] who provides an extensive review on direct methods and Heath et al. [HNP91] who provide a survey on parallel algorithms. Often, especially in distributed systems, it is simpler to implement an iterative method to find a solution. Iterative Krylov subspace methods, such as conjugate gradient, are often used due to their fast convergence. However for larger linear systems, memory consumption is a limiting factor and iterative methods such as Successive Over-Relaxation (SOR) [Axe94] are preferred.

If accuracy is not crucial, a coarse approximation to the Poisson solution may be sufficient to achieve the desired result. Extending a coarse solution to finer resolutions using Bilateral upsampling [KCLU07] has been shown to produce good results for applications such as tonemapping. Such methods have not yet been shown to handle applications such as panorama stitching where the solution is typically not smooth at the seams between images.

Often multigrid methods are used to aid the convergence of an iterative solver. These techniques include preconditioners [GC95, Sze08] and multigrid solvers [Bra77, BHM00]. There exist different variants of multigrid algorithms using either adaptive [BC89, KBH06, BKBH07, Aga07, Ric08] or non-adaptive meshes [Kaz05, KH08, KSH10, SSJ*10]. There has been a significant amount of work in distributed multigrid methods and we refer the reader to [CFH*06] for a survey of current methods.

Recently, it has been shown that combining upsampling and the coarse-to-fine half of a multigrid cycle results in high quality results for imaging [SSJ*10]. In this paper, it was shown that an initial coarse solution, when upsampled and used as the initialization of an iterative solver, produces results visually indistinguishable from a direct solution. See Figure 1 (middle) for an example.

Out-of-Core. Toledo [Tol99] provides a survey of general out-of-core algorithms for linear systems. Most algorithms surveyed assume that at least the solution can be kept in main memory, though this is rarely the case for large images. For out-of-core processing of large images, the streaming multigrid method of Kazhdan and Hoppe [KH08] and the progressive Poisson method [SSJ*10] have so far provided the only solutions. Recently, streaming multigrid has been extended to a distributed environment [KSH10] and has reduced the time to process gigapixel images from hours to minutes. Out-of-core methods often achieve a low memory footprint at the cost of significant disk storage requirements. For example, the multigrid method [KH08] requires auxiliary storage often an order of magnitude greater than the input size, almost half of which is due to gradient storage. The distributed multigrid requires 16 bytes/pixel of disk space in temporary storage for the solver as well as 24 bytes/pixel to store the solution and gradient constraints. For [KSH10]'s terapixel example, the method had a minimum requirement of 16 nodes in order to accommodate the needed disk space for fast local caching. In contrast, the approach outlined in this paper needs no temporary storage and is implemented in standard MPI. Streaming multigrid also assumes a precomputed image gradient, which would add substantial overhead in initialization to transfer the color float or double data. Our approach is initialized using original image data plus an extra byte for image boundary information which equates to 1/3 less data transfer in initialization than the previous method. Data transfers between the two phases, while floating point, only deal with the node boundaries which is substantially smaller than the full image and therefore are rarely required to be cached to disk. The multigrid method [KH08, KSH10] may also be limited by main memory, since the number of iterations of the solver is directly proportional to the memory footprint. For large images, this limits the solver to only a few Gauss-Seidel iterations and therefore may not necessarily converge for challenging cases. Our method's memory usage is independent of the number of iterations and can therefore solve images that have slow convergence.

Often large images are stored as tiles at the highest resolution; therefore methods that exploit this structure would be advantageous. Stookey et al. [SXC*08] use a tiled base approach to compute an over-determined Laplacian PDE. By using tiles that overlap in all dimensions, the method solves the PDE on each tile separately and then blends the solution via a weighted average. Unfortunately this method cannot account for large scale trends beyond a single overlap and therefore can only be used on problems which have no large (global) trends. Figure 2 illustrates this problem and why the coarse upsampling in our method is necessary.

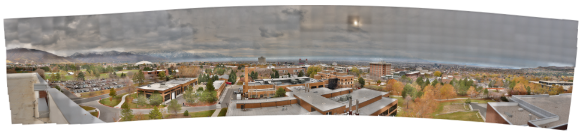


Figure 2: *Although the result is a seamless, smooth image, without coarse upsampling the final image will fail to account for large trends that span beyond a single overlap and can lead to unwanted, unappealing shifts in color.*

3. Gradient Domain Review

Rather than operating directly on the pixel values, gradient based techniques manipulate an image based on the value of a gradient field. Seamless cloning, panorama stitching, and high dynamic range tone mapping are all techniques that belong to this class. Given a gradient field $\vec{G}(x,y)$, defined over a domain $\Omega \subset \Re^2$, we seek to find an image $P(x,y)$ such that its gradient ∇P fits $\vec{G}(x,y)$.

In order to minimize $\|\nabla P - \vec{G}\|$ in a least squares sense, one has to solve the following optimization problem:

$$\min_P \iint_\Omega \|\nabla P - \vec{G}\|^2 \tag{1}$$

Minimizing equation (1) is equivalent to solving the Poisson equation $\Delta P = div\,\vec{G}(x,y)$ where Δ denotes the Laplace operator $\Delta P = \frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2}$ and $div\,\vec{G}(x,y)$ denotes the divergence of $\vec{G}$.

To adapt the equations shown above to discrete images, we apply a standard finite difference approach which approximates the Laplacian as:

$$\begin{aligned}\Delta P(x,y) &= P(x+1,y)+P(x-1,y)+ \\ &\quad P(x,y+1)+P(x,y-1)-4P(x,y)\end{aligned} \tag{2}$$

and the divergence of $\vec{G}(x,y) = (G^x(x,y), G^y(x,y))$ as:

$$\begin{aligned}div\,\vec{G}(x,y) &= G^x(x,y)-G^x(x-1,y)+ \\ &\quad G^y(x,y)-G^y(x,y-1).\end{aligned}$$

For images, we discretize the differential form $\Delta P = div\,\vec{G}(x,y)$ using finite differences into the following sparse linear system: $L\mathbf{p} = \mathbf{b}$.

Each row of the matrix L stores the weights of the standard five point Laplacian stencil, $\mathbf{p}$ is the vector of pixel colors, and $\mathbf{b}$ encodes the guiding gradient field. Both L and $\mathbf{b}$ encode boundary conditions as well. Gradient domain techniques are typically defined by how the guiding gradient field is computed and what boundary conditions are chosen. For instance, seamless cloning uses Dirichlet boundaries set to the color values of the background image and the foreground image's gradient as the guiding field (see Perez et al. [PGB03] for a detailed description). For panorama stitching, Neumann boundary conditions are used and the guiding gradient field is computed as a composite of the gradients from the source images. At image boundaries, where an unwanted large gradient exists (the seams), the gradient between images is averaged or considered to be zero [PGB03, LZPW04, ADA*04, KH08, SSJ*10]. Another interesting example is gradient domain painting [MP08] which uses artistic input as the guiding field.

4. Parallel Gradient Domain Blending

In the following, we provide details of our parallel progressive Poisson algorithm and MPI implementation.

4.1. Parallel Solver

Commonly, large images are stored as tiles, which gives one an underlying structure to divide an image amongst the nodes/processors for a distributed solver. Tile-based distributed solvers have been shown to work well when only local trends are present. Seamless stitching commonly contains large scale trends where a naive tile-based approach will provide poor results. (see Figure 2). The addition of the progressive Poisson method's coarse upsampling, allows for a simple, tile-based parallel solver that can account for large trends. Our algorithm works in two phases: The first phase performs the progressive upsample of a precomputed coarse solution for each tile. The second phase solves for a smooth image on tiles that significantly overlap the solution tiles from the first phase. In this way, the second phase smooths any seams not captured or even introduced by the first phase, producing a complete, seamless image.

Data distribution as tiles. Although a tile-based approach leverages a common image storage format, it is not typically how methods are designed to handle seamless stitching of large panoramas. For instance, methods like streaming multigrid [KH08, KSH10] often assume precomputed gradients for the whole image. Our system is designed to take tiles directly as input and therefore must be able to handle the gradient computation on-the-fly. An important and often undocumented component of panorama stitching is the map or label image. Given an ordered set of images which compose the panorama, the map image gives the correspondence of a pixel location in the overall panorama to the smaller image that supplies the color. This map file is necessary to determine the difference between actual gradients and those due to seams. This map also defines the boundaries of the panorama, which are commonly irregular. This file along with each individual image that composes the mosaic are needed for a traditional, out-of-core scheme [KH08, SSJ*10] for gradient computation. If the gradient across the seams is assumed to be zero, which is a common technique we adopt for this solver, each tile can be composited in advance and the map file is only needed to denote image seams or boundary. As noted above, this composited tile is often already provided if used in a traditional large image system.

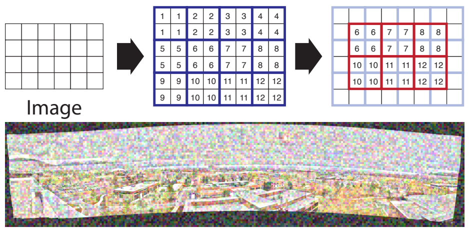


Figure 3: *Our tile-based approach: (top-left) An input image is divided into equally spaced tiles. (top-center) In the first phase after a symbolic padding by a column and row in all dimensions, a solver is run on a window denoted by a collection of 4 labeled tiles. (top-right) Data is sent and collected for the next phase to create new data windows with a 50% overlap. (bottom) An example tile layout for the Fall Panorama example.*

The map file can then be encoded as an extra channel of color information, typically the alpha channel. For mosaics of many hundreds of images, such as the examples in this paper, we cannot encode an index for each image in a byte of data. Though in practice each tile has very little probability of having more than 256 individual images, therefore each image is given a unique 0-255 number on a per tile basis.

We have chosen an overlap of 50% in both dimensions for the second phase windowing scheme of the parallel solver for simplicity in implementation. Each *window* is composed of a 2×2 collections of tiles. To avoid undefined windows in the second phase, we add a symbolic padding of one row/column of tiles to all sides of the image which the solver regards as pure boundary. Figure 3 gives an example of a tile layout. The overlapping window size used for our testing was 1024×1024 pixels (assuming 512×512 tiles) which we found to be a good compromise between a low memory footprint and image coverage. Each node receives a partition of windows equivalent to a contiguous sub-image with no overlap necessary between nodes during the same phase. Data can be distributed evenly across all nodes in the case of a homogeneous distributed system or dependent on weights due to available resources in the case of a heterogeneous hardware. We provide a test case for a heterogeneous system in Section 5.

Coarse Solution. As a first step, the first phase of our solver will upsample via bilinear interpolation a 1-2 megapixel coarse solution. Much like the progressive Poisson method [SSJ*10], each node computes a solution in just a few seconds using a direct FFT solver on a coarsely sampled version of our large image. In tiled hierarchies, this coarse image is typically already present and can be encoded with the map information in much the same way as the tiles.

First phase: Progressive Solution. This phase computes a progressive Poisson solution for each window which are composed of tiles read off of a distributed file system. To progressively solve a window, an image hierarchy is necessary. For our implementation a standard power-of-two image pyramid was used. As a first step, the solver upsamples the solution to a finer resolution in the image pyramid using a coarse solution image and the original pixel values. An iterative solver is then run for several iterations to smooth this upsample using the original pixel gradients as the guiding field. This process is repeating down the image hierarchy until the full resolution is reached. The solver is considered to have converged at this resolution when the L_2 norm falls below 10^{-3} which is based on the range of byte color data. From our testing, we have found that SOR gives both good running times and low memory consumption and therefore is our default solver. As noted above, this window is logically composed of four tiles, which are computed and saved in memory for the next phase as floating point color data. This leads to 12 bytes/pixel (3 floating point color data) to transfer between phases. Given the data distribution, one node may process many windows. If this is the case, only the tiles which border a node's domain are prepared to be transferred to another node, thereby keeping data communication between phases to a relatively small zone.

Second phase: Overlap Solution. The second phase gathers the four tiles (both solution and original) that make up the overlapping window. After the data is gathered, the gradients are computed from the original pixel values and an iterative solver (SOR) is run after being initialized with the solutions from the first phase. The iterative solver is constrained to only work on interior pixels to prevent this phase from introducing new seams at the window boundary. Technically, there may be errors at the pixels around the midpoints of the boundary edges of these windows, though we have not encountered this in practice. Again, this solver is run until convergence given by the L_2 norm. Note that even though the tile gradients are computed in the first phase, we have chosen to recompute them on the fly in the second phase. Passing the gradients would cost at least an additional 12 bytes/pixel overhead. As nodes increase, data transfer and communication becomes a significant bottleneck in most distributed schemes therefore we chose to pay the cost of increased computation and reading the less expensive byte image data from the distributed file system instead of the costly transfer.

Parallel Implementation Details. Each node has one *master* thread which coordinates all processing and communication. The core component of this thread is a priority queue of windows and tiles to be processed. At launch, this queue is initialized by a separate *seeding* thread with the initial domain of windows to be solved in the first phase. Because of the separation of the main thread from the seeding of the queue, the main thread can begin processing windows immediately. Each window is given a first phase id, which is the window's row and column location in the sub-image to be processed by a node. Communication between nodes need only be one-way in our system, therefore we have chosen for communication to be "upstream" between nodes, i.e. the nodes operating on a sub-image with horizontal or vertical location greater than the current node. In order to avoid starvation in the second phase, the queue is loaded with windows in reverse order in terms of the tile id. Figure 4 gives an example of the traversal and communication. All initially seeded windows are given equal low priority in the queue. In essence the initial queue operates much like a first-in-first-out (FIFO) queue. As windows are removed from the queue, the main thread launches a progressive solver thread which is handed off to an intra-node dynamic scheduler. Our implementation uses a HyperFlow [VOS*10] scheduler to execute the solver on all available cores. HyperFlow has been shown to efficiently schedule execution of workflows on multi-core systems and therefore is the perfect solution for our intra-node scheduling. In all there are two distinct sequential stages in each phase: (1) loading of the tile data and the computation of the image gradient and (2) the progressive solution. This flow information allows HyperFlow to exploit data, task, and pipeline parallelism to maximize throughput.

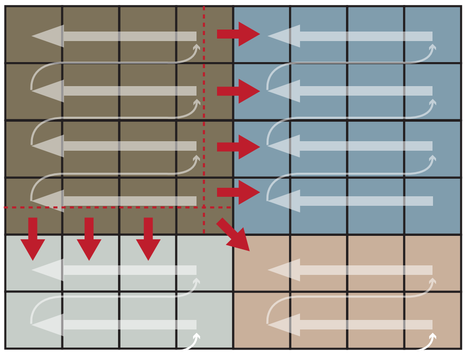


Figure 4: *Windows are distributed as evenly as possible across all nodes in the distributed system. Windows assigned to a specific node are denoted by color above. Given the overlap scheme, data transfer only needs to occur one-way, denoted by the red arrows and boundary above. To avoid starvation between phases and to hide as much data transfer as possible, windows are processed in inverse order (white arrows) and the tiles needed by other nodes are transferred immediately.*

After a solution is computed, the progressive solver thread partitions the window into the tiles that comprise it. This allows the second phase to recombine the tiles needed for the 50% overlap window. All four tiles are loaded into the queue with high priority. If the main thread removes a tile (as opposed to a window) from the queue and the tile is needed by another node, the main thread immediately sends the data asynchronously to the proper node. Otherwise, if the node needs this tile for phase two, the second phase id of the window which needs the tile is computed and hashed with a 2D hash function the same size as the window domain for the second phase. If all four tiles for a given second phase window have been hashed, the main thread now knows a second phase window is ready and immediately passes the window to a solver thread for processing. If the main thread receives a solved tile from another node, this is also immediately hashed.

5. Results

To demonstrate the scalability and adaptability of the approach, we have tested our implementation using two panorama datasets, gigapixels in size. To illustrate the portability of the system, we have also shown its running times and scalability on two distributed systems. Our main system, the *NVIDIA Center of Excellence* cluster in the Scientific Computing and Imaging Institute at the University of Utah, consists of 60 active nodes with 2.67GHz Xeon X5550 Processors (8 cores), 24GB of RAM per node, and 750GB local scratch disk space. The second system, the *Longhorn* visualization cluster in the Texas Advanced Computer Center at the University of Texas at Austin, consists of 256 nodes (of which 128 were available for our tests) with 2.5GHz Nehalem Processors (8 cores), 48GB of RAM per node, and 73GB local scratch disk space. Weak and strong scalability tests were performed on both systems. Given the proven scalability of Hyperflow on one node, we have tested the scalability of the MPI implementation from 2-60 and 2-128 nodes for the *NVIDIA cluster* and *Longhorn cluster* respectively. Timings are taken as best over several runs to discount external effects to the cluster from shared resources such as the distributed file system.

Datasets used:

- **Fall Panorama.** $126,826 \times 29,633$, 3.27-gigapixel. When tiled, this dataset is composed of 124×29 1024^2 sized windows. See Figure 5 for image results from a *NVIDIA cluster* 480 core test run.
- **Winter Panorama.** $92,570 \times 28,600$, 2.65-gigapixel. When tiled, this dataset is composed of 91×28 1024^2 sized windows. See Figure 6 for image results from a *NVIDIA cluster* 480 core test run.

NVIDIA Cluster. To show the MPI scalability of our framework and implementation, strong and weak scaling tests were performed for 2-60 nodes. As shown in Figure 7, both datasets scale close to ideal and with high efficiency for strong scaling. The *Fall Panorama*, due to it's larger size begins to lose efficiency at around 32 nodes when I/O overhead begins to dominate. Even with this overhead, the efficiency remains acceptable. For the *Winter Panorama*, the I/O overhead does not effect performance up to 60 nodes and the implementation maintains efficiency throughout the test. Weak scaling tests were performed using a sub-image of the *Fall Panorama* dataset. See Figure 8 for the weak scaling results. As the number of cores increases so does the image resolution to be solved. The sub-image was expanded from the center of the full image and iterations of the solver for all windows were locked at 1000 for testing to ensure no variation is due to slower converging image areas. As the figure shows, our implementation shows good weak scaling efficiency even for 60 nodes with 480 cores. In all, we have produced a gradient domain solution to a dataset which in previous work the best known methods [KH08,SSJ*10] took hours to compute.

Longhorn Cluster. To show the portability and MPI scalability of our framework and implementation, strong and weak scaling tests were performed on the largest dataset (*Fall Panorama*) on a second cluster. The strong scalling tests were perfromed from 2-128 nodes and the weak scalling tests, limited by the size of the image, were perform from 2-64 nodes. As shown in Figure 10, our implementation maintains very good efficiency and timings for our

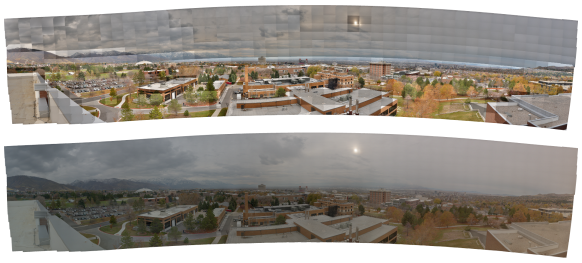


Figure 5: *Fall Panorama - $126,826 \times 29,633$, 3.27-gigapixel. (top) The panorama before seamless blending and (bottom) the result of the parallel Poisson solver run on 480 cores with 124×29 windows and computed in 5.88 minutes.*

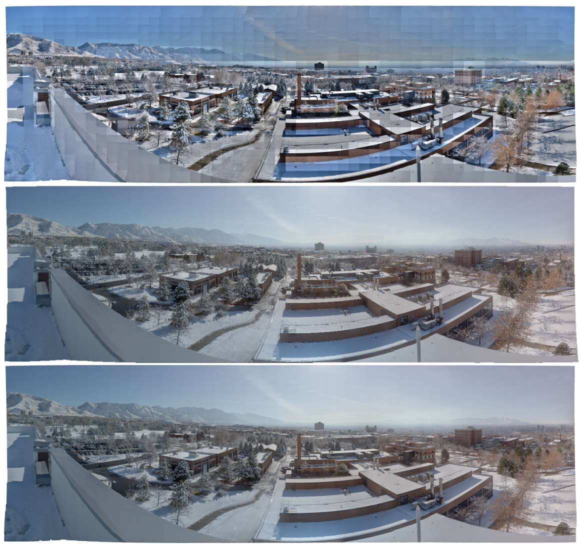
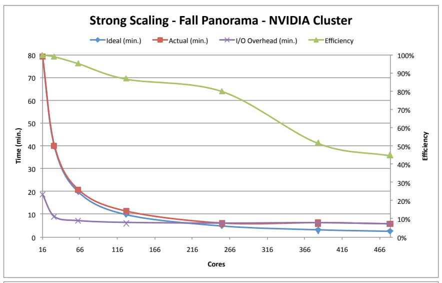
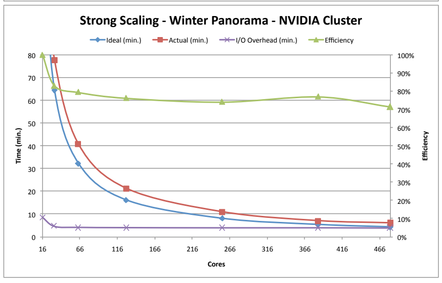


Figure 6: *Winter Panorama - $92,570 \times 28,600$, 2.65-gigapixel. (top) The panorama before seamless blending, (middle) the coarse panorama solution, and (bottom) the result of the parallel Poisson solver run on 480 cores with 91×28 windows and computed in 6.02 minutes.*



Strong Scaling - Fall Panorama - NVIDIA Cluster						
Nodes	Cores	Ideal[†]	Actual[†]	Efficiency	Overhead[†]	% Overhead
2	16	79.35	79.35	100.00%	18.80	23.69%
4	32	39.68	40.08	97.08%	9.05	22.15%
8	64	19.84	20.83	95.22%	7.28	34.96%
16	128	9.92	11.43	78.93%	6.50	51.72%
32	256	4.96	6.20	53.81%	6.20	67.27%
48	384	3.31	6.40	51.66%	6.40	100.00%
60	480	2.65	5.88	44.96%	5.88	100.00%



Strong Scaling - Winter Panorama - NVIDIA Cluster						
Nodes	Cores	Ideal[†]	Actual[†]	Efficiency	Overhead[†]	% Overhead
2	16	128.87	128.87	100.00%	8.63	6.70%
4	32	64.43	77.68	82.94%	4.70	6.05%
8	64	32.22	40.63	79.29%	4.28	10.54%
16	128	16.11	21.17	76.10%	4.17	19.69%
32	256	8.05	10.88	74.00%	4.08	37.52%
48	384	5.37	6.98	76.89%	4.10	58.71%
60	480	4.30	6.02	71.39%	4.00	66.48%

Figure 7: *The strong scaling results for the Fall and Winter Panorama run on the NVIDIA Cluster from 2-60 nodes up to a total of 480 cores. Overhead due to MPI communication and I/O is also provided along with its percentage of actual running time. (top) The Fall Panorama, due to it's larger size begins to lose efficiency at around 32 nodes when I/O overhead begins to dominate. Even with this overhead, the efficiency remains acceptable. (bottom) For the Winter Panorama, the I/O overhead does not effect performance up to 60 nodes and the implementation maintains efficiency throughout all of our runs. († All timings are in minutes.)*

Weak Scaling - NVIDIA Cluster				
Nodes	Cores	Size (MP)	Time (min.)	Efficiency
2	16	100.66	5.55	100.00%
4	32	201.33	5.55	100.00%
8	64	402.65	5.53	100.30%
16	128	805.31	5.68	97.65%
32	256	1610.61	5.77	96.24%
60	480	3019.90	6.57	84.52%

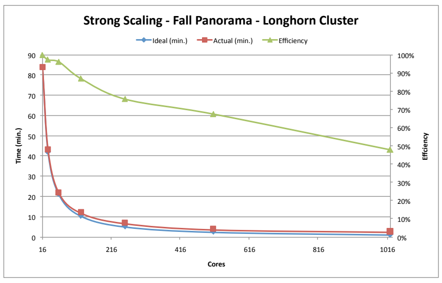
Figure 8: *Weak scaling tests run on the NVIDIA Cluster for the Fall Panorama dataset. As is shown, our implementation shows good efficiency even when running on the maximum number of cores.*

Weak Scaling - Longhorn Cluster				
Nodes	Cores	Size (MP)	Time (min.)	Efficiency
2	16	75.5	5.50	100.00%
4	32	151	6.13	89.67%
8	64	302	6.15	89.43%
16	128	604	6.15	89.43%
32	256	1208	6.13	89.67%
64	512	2416	6.15	89.43%

Figure 9: *Weak scaling tests run on the Longhorn Cluster for the Fall Panorama dataset.*

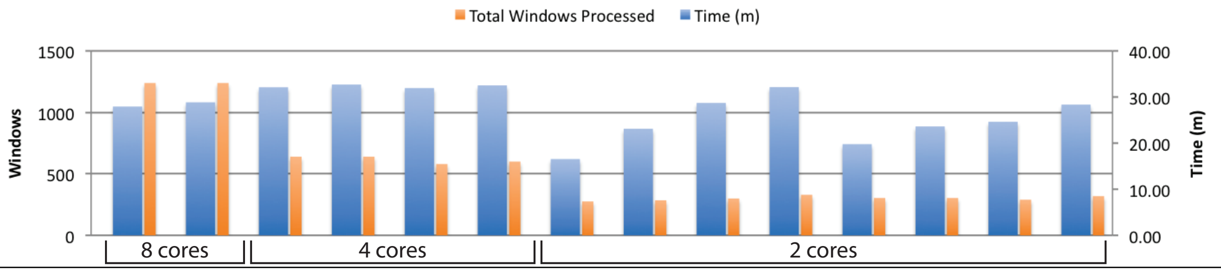
strong scaling test up to the full 1024 cores available on the system. Much like the *NVIDIA Cluster*, weak scaling tests were performed on a portion of the *Fall Panorama* and iterations of the solver were locked at 1000. To ensure that each node got a resonably sized sub-image to solve, the tests were limited to 64 nodes. Figure 9 demonstrates our implementations ability to weak scale on this cluster, maintaining good efficiency for up to 512 cores.

Heterogeneous Cluster. As a final test of portability and adaptability, we presented our implementation with a simulated heterogeneous distributed system. Our parallel framework provides the ability to give weights to nodes which is typically even and therefore results in an even distribution of windows across all nodes. For this example, a simple weighting scheme can easily load-balance this mixed network, giving the nodes with more resources more windows to compute. Figure 11 gives an example mixed system of 2 8-core nodes, 4 4-core nodes, and 8 2-core nodes. In all, this system has 48 available cores. The weights for our framework are simply the number of cores available in each node. This network was simulated using the *NVIDIA Cluster* by overloading Hyperflow's knowledge of available resources with our desired properties. While this is not a perfect simulation since the main thread handling MPI communication would not be limited to reside on the desired cores, as shown in the strong scaling tests even with evenly distributed data on 8-16 nodes the implementation is not yet I/O bound. Therefore we should still have a good approximation to a real, limited system. The figure details the window distribution and timings for the *Fall Panorama* for all nodes in this test. As is shown, we maintain good load balancing given proper node weight-



Strong Scaling - Fall Panorama - Longhorn Cluster				
Nodes	Cores	Ideal (min.)	Actual (min.)	Efficiency
2	16	84.07	84.07	100.00%
4	32	42.03	43.18	97.34%
8	64	21.02	21.85	96.19%
16	128	10.51	12.08	86.97%
32	256	5.25	6.93	75.78%
64	512	2.63	3.89	67.61%
128	1024	1.31	2.73	48.06%

Figure 10: *To demonstrate the portability of our implementation, we have run strong scalability testing for the Fall Panorama on the Longhorn Cluster from 2-128 nodes up to a total of 1024 cores. As the numbers show, we maintain good scalability and efficiency even when running on all available nodes and cores.*



Heterogeneous System - Fall Panorama														
Cores	8		4				2							
Time (m)	27.93	28.85	32.12	32.70	31.93	32.52	16.55	23.12	28.72	32.15	19.78	23.62	24.63	28.35
Windows	1239	1239	640	640	580	600	276	285	300	330	304	304	290	319

Figure 11: *Our simulated heterogeneous system. This test example is a simulated mixed system of 2 8-core nodes, 4 4-core nodes, and 8 2-core nodes. The weights for our framework are the number of cores available in each node. The timings and window distributions are for Fall Panorama dataset. As you can see, with the proper weightings our framework can distribute windows proportionally based on the performance of the system. The max runtime of 32.70 minutes for this 48 core system is on par with timings for the 32 core (40.08 minutes) and 64 core (20.83 minutes) runs from the strong scaling test.*

ing when dealing with heterogenous systems. The max runtime of 32.70 minutes for this 48 core system is on par with run time for the 32 core (40.08 minutes) and 64 core (20.83 minutes) strong scaling results.

6. Conclusions

This paper describes a new framework for parallel gradient domain processing of massive images. The framework provides both a straightforward implementation, maintains strict control over the required/allocated resources, and makes no assumptions on the speed of convergence to an acceptable image. Three important properties not present in previous parallel gradient domain work. This framework also provides a new parallel algorithm for gradient domain processing which is detailed in this paper along with its MPI implementation.

When implemented in standard MPI, this framework is highly portable and scalable across a wide variety of distributed systems, for which this paper has provided two examples. Moreover, this paper has provided both strong and weak scaling tests and has shown that this new framework maintains good scalability and efficiency in both cases and in the case of strong scaling, on multiple datasets. It was also shown that this framework can be configured to handle heterogeneous distributed systems given a proper tile distribution scheme.

Acknowledgements

This work is supported in part by the National Science Foundation awards OCI-0904631, OCI-0906379, IIS-1045032, and CCF-0702817. This work was also performed under the auspices of the U.S. Department of Energy by the University of Utah under contract DE-SC0001922 and DE-FC02-06ER25781 and by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344.

References

[ACR05] AGRAWAL A. K., CHELLAPPA R., RASKAR R.: An algebraic approach to surface reconstruction from gradient fields. In *ICCV* (2005), pp. I: 174–181.

[ADA*04] AGARWALA A., DONTCHEVA M., AGRAWALA M., DRUCKER S., COLBURN A., CURLESS B., SALESIN D., COHEN M.: Interactive digital photomontage. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers* (New York, NY, USA, 2004), ACM.

[Aga07] AGARWALA A.: Efficient gradient-domain compositing using quadtrees. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers* (New York, NY, USA, 2007), ACM, p. 94.

[ARC06] AGRAWAL A. K., RASKAR R., CHELLAPPA R.: What is the range of surface reconstructions from a gradient field? In *ECCV* (2006), pp. I: 578–591.

[ARNL05] AGRAWAL A., RASKAR R., NAYAR S. K., LI Y.: Removing photography artifacts using gradient projection and flash-exposure sampling. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), ACM, pp. 828–835.

[Axe94] AXELSSON O.: *Iterative Solution Methods*. Cambridge Universty Press, New York, NY, 1994.

[BC89] BERGER M. J., COLELLA P.: Local adaptive mesh refinement for shock hydrodynamics. *Journal Computational Physics 82* (1989), 64–84.

[BHM00] BRIGGS W. L., HENSON V. E., MCCORMICK S. F.: *A Multigrid Tutorial*, 2nd ed. SIAM, 2000.

[BKBH07] BOLITHO M., KAZHDAN M., BURNS R., HOPPE H.: Multilevel streaming for out-of-core surface reconstruction. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing* (Aire-la-Ville, Switzerland, Switzerland, 2007), Eurographics Association, pp. 69–78.

[Bra77] BRANDT A.: Multi-level adaptive solutions to boundary-value problems. *Mathematics of Computation 31*, 138 (1977), 333–390.

[CFH*06] CHOW E., FALGOUT R. D., HU J. J., TUMINARO R. S., YANG U. M.: A survey of parallelization techniques for multigrid solvers. In *Parallel Processing for Scientific Computing*, Heroux M. A., Raghavan P., Simon H. D., (Eds.), vol. 20 of *Software, Environments, and Tools*. SIAM, Philadelphia, Nov. 2006, pp. 179–201. ch. 10,.

[Dor70] DORR F. W.: The direct solution of the discrete poisson equation on a rectangle. *SIAM Review 12*, 2 (April 1970), 248–263.

[FHD02] FINLAYSON G. D., HORDLEY S. D., DREW M. S.: Removing shadows from images. In *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part IV* (London, UK, 2002), Springer-Verlag, pp. 823–836.

[FHL*09] FARBMAN Z., HOFFER G., LIPMAN Y., COHEN-OR D., LISCHINSKI D.: Coordinates for instant image cloning. In *SIGGRAPH '09: Proceedings of the 36th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2009), ACM.

[FLW02] FATTAL R., LISCHINSKI D., WERMAN M.: Gradient domain high dynamic range compression. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2002), ACM.

[GC95] GORTLER S., COHEN M.: Variational modeling with wavelets. In *Symposium on Interactive 3D graphics* (1995), pp. 35–42.

[Gigin] GIGAPAN:, http://www.gigapan.org/about.php.

[HiRin] HIRISE:, High Resolution Imaging Science Experiment http://hirise.lpl.arizona.edu/.

[HNP91] HEATH, NG, PEYTON: Parallel algorithms for sparse linear systems. *SIREV: SIAM Review 33* (1991).

[Hoc65] HOCKNEY R. W.: A fast direct solution of Poisson's equation using Fourier analysis. *Journal of the ACM 12*, 1 (Jan. 1965), 95–113.

[Hor74] HORN B. K. P.: Determining lightness from an image. *Comput. Graphics Image Processing 3*, 1 (Dec. 1974), 277–299.

[JSTS06] JIA J., SUN J., TANG C.-K., SHUM H.-Y.: Drag-and-drop pasting. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers* (New York, NY, 2006), ACM, pp. 631–637.

[Kaz05] KAZHDAN M.: Reconstruction of solid models from oriented point sets. In *Eurographics Symposium on Geometry Processing* (2005), pp. 73–82.

[KBH06] KAZHDAN M., BOLITHO M., HOPPE H.: Poisson surface reconstruction. In *Eurographics Symposium on Geometry Processing* (2006), pp. 61–70.

[KCLU07] KOPF J., COHEN M. F., LISCHINSKI D., UYTTENDAELE M.: Joint bilateral upsampling. *ACM ToG 26*, 3 (2007), 96.

[KH08] KAZHDAN M., HOPPE H.: Streaming multigrid for gradient-domain operations on large images. *ACM ToG. 27*, 3 (2008).

[KSH10] KAZHDAN M., SURENDRAN D., HOPPE H.: Distributed gradient-domain processing of planar and spherical images. *ACM ToG. to appear* (2010).

[LZPW04] LEVIN A., ZOMET A., PELEG S., WEISS Y.: Seamless image stitching in the gradient domain. In *In Eighth European Conference on Computer Vision (ECCV 2004* (2004), Springer, pp. 377–389.

[MP08] MCCANN J., POLLARD N. S.: Real-time gradient-domain painting. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers* (New York, NY, USA, 2008), ACM, pp. 1–7.

[PGB03] PÉREZ P., GANGNET M., BLAKE A.: Poisson image editing. *ACM ToG. 22*, 3 (2003), 313–318.

[Ric08] RICKER P. M.: A direct multigrid poisson solver for oct-tree adaptive meshes. *The Astrophysical Journal Supplement Series 176* (2008), 293–300.

[SJTS04] SUN J., JIA J., TANG C.-K., SHUM H.-Y.: Poisson matting. *ACM ToG. 23*, 3 (2004), 315–321.

[SSJ*10] SUMMA B., SCORZELLI G., JIANG M., BREMER P.-T., PASCUCCI V.: Interactive editing of massive imagery made simple: Turning atlanta into atlantis. *ACM Transactions on Graphics (TOG) 29*, 5 (2010).

[SXC*08] STOOKEY J., XIE Z., CUTLER B., FRANKLIN W. R., TRACY D. M., ANDRADE M. V. A.: Parallel ODETLAP for terrain compression and reconstruction. In *GIS* (2008), Aref W. G., Mokbel M. F., Schneider M., (Eds.), ACM, p. 17.

[Sze08] SZELISKI R.: Locally adapted hierarchical basis preconditioning. *ACM ToG. 27*, 3 (2008), 1135–1143.

[Tol99] TOLEDO S.: A survey of out-of-core algorithms in numerical linear algebra. In *External memory algorithms*, Dimacs Series In Discrete Mathematics And Theoretical Computer Science. American Mathematical Society, Boston, MA, 1999, pp. 161–179.

[USGin] USGS:, . United States Geological Survey http://www.usgs.gov/.

[VOS*10] VO H. T., OSMARI D. K., SUMMA B., COMBA J. L. D., PASCUCCI V., SILVA C. T.: Streaming-enabled parallel dataflow architecture for multicore systems. *Comput. Graph. Forum 29*, 3 (2010), 1073–1082.

[Wei01] WEISS Y.: Deriving intrinsic images from image sequences. In *International Conference on Computer Vision* (2001), pp. 68–75.