Comparing GPU Implementations of Bilateral and Anisotropic Diffusion Filters for 3D Biomedical Datasets

Mark Howison

Computational Research Division
Lawrence Berkeley National Laboratory
One Cyclotron Road, Berkeley, CA 94720, USA
mhowison@lbl.gov

***Abstract*—We compare the performance of hand-tuned CUDA implementations of bilateral and anisotropic diffusion filters for denoising 3D MRI datasets. Our tests sweep comparable parameters for the two filters and measure total runtime, memory bandwidth, computational throughput, and mean squared errors relative to a noiseless reference dataset.**

I. Introduction

Denoising is a important step in many image processing pipelines for brain magnetic resonance imaging (MRI). We focus on two 3D filters, the bilateral filter and the anisotropic diffusion filter, that remove noise and smooth features within MR images while at the same time preserving edges in the image. Our implementation of these stencil-based algorithms are hand-tuned in NVIDIA's CUDA programming language to take advantage of the high computational throughput of GPU coaccelerators.

This work was supported by the Director, Office of Science, Office of Basic Energy Sciences, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

Both filters we investigate offer several tunable parameters, and we perform parameter sweeps that identify both optimal runtime and noise reduction. To measure noise reduction, we use MR images from a simulated brain database called BrainWeb [1], which include a noiseless reference dataset and a dataset with realistic MRI noise added.

One application scenario for our system is rapid turnaround of MRI data smoothing over a range of filtering parameters. The idea is that between data acquisition and physician inspection there is a battery of filtering operations; our highly tuned implemenation can minimize the time required for such a filtering battery, or could enable real-time, interactive exploration of filter parameter choices. Figure 1 shows an example of the output from a filtering battery over many combinations of parameters for the bilateral filter.

II. Background and Related Work

Image smoothing, or denoising, is a fundamental operation in computer vision and image processing. One of the simplest approaches to smoothing is to perform averaging of nearby points to compute an estimate of the noiseless signal. A "box filter" computes an estimate using equal weights for all the nearby sample points. A better estimate of the average would be to afford greater weights to nearby points and smaller weights to more distant points. The Gaussian low-pass filter performs such an averaging using a set of weights defined over a normal distribution such that points nearby the target sample point have a greater contribution to the average than points far away from the sample point. For an image I and sample point p, the new value in the filtered image I' can be computed as

$$I'_p = \sum_{q \in N_p} G_\sigma(\|p-q\|) I_q$$

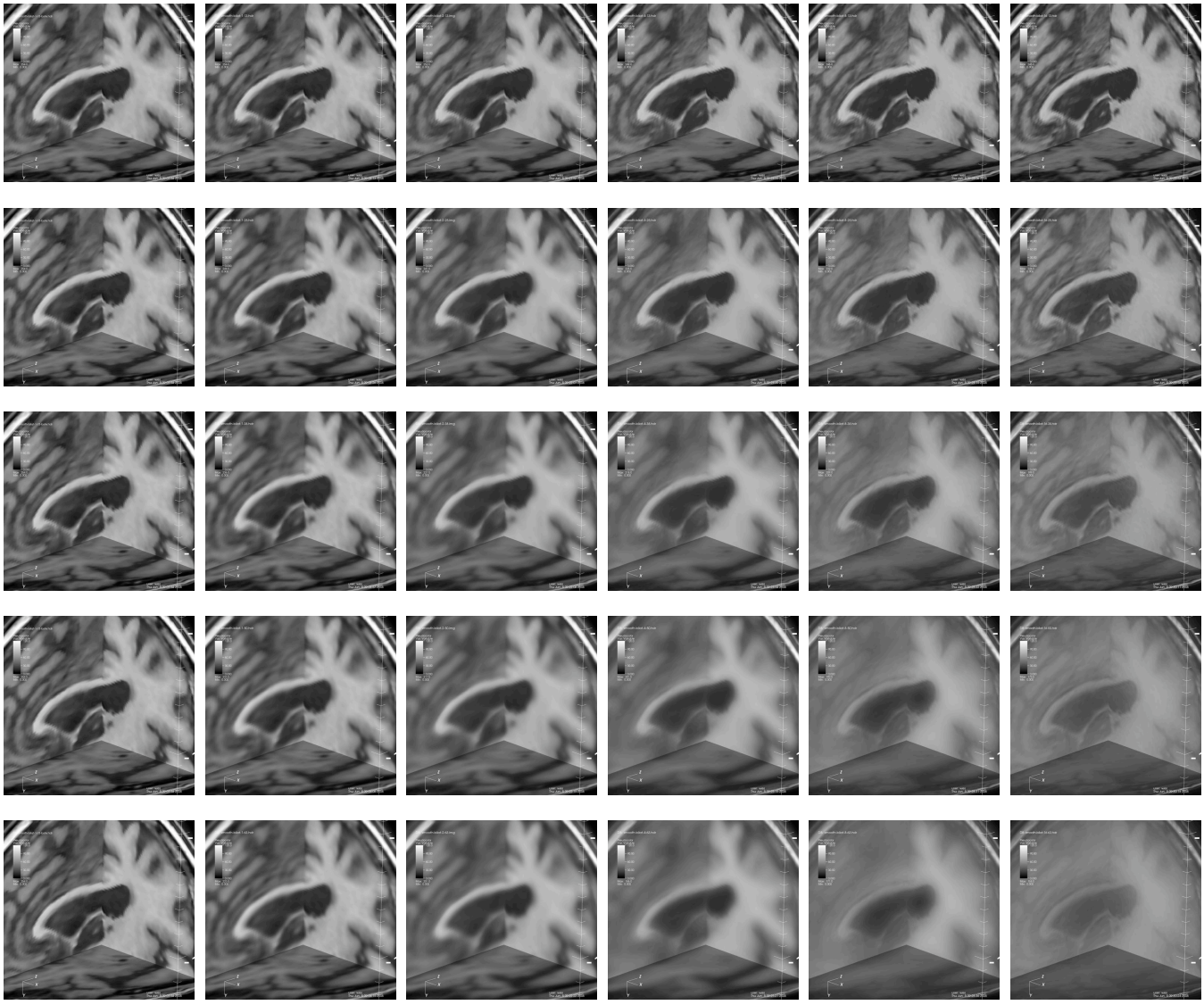

Fig. 1. These images show a zoomed-in view of three orthogonal slices 3D brain MR images smoothed with our bilateral filter implementation. The first column shows the original data, and the subsequent columns show smoothed versions for $r = \{1,2,4,8,16\}$. The rows show varying standard deviations in the photometric range, $\sigma_r = \{5\%,10\%,15\%,20\%,25\%\}$.

where N_s is a neighborhood of points around p and G_σ is a Gaussian kernel with standard deviation σ.

This type of smoothing is isotropic in the sense that the filter application is performed independent of the underlying signal. The result is that it smooths equally in all directions, which has the unfortunate side effect of blurring edges. We have implemented two well-known image denoising filters that try to solve the edge blurring problem: bilateral filtering and anisotropic diffusion.

A. Bilateral Filtering

Bilateral filtering, as defined by Tomasi [2], aims to perform anisotropic image smoothing using a low-cost, non-iterative formulation. The idea is to smooth images by computing the influence of nearby points in a way that removes noise "within regions," and that does not have the undesirable property of smoothing edge features. This formulation uses a straightforward, tunable estimate for region boundaries: a Gaussian-weighted difference in the range of the signal, or photometric space. Wherever a sharp edge exists, there will be a large difference in signal that will correspond to a small Gaussian weight, thereby attenuating the smoothing. The range estimate is combined with a traditional Gaussian-weighted distance function in the spatial domain to lessen the contribution of more distant voxels.

In the bilateral filter, the output at each pixel location p in an image I is the weighted average of the influence

of nearby pixels in the neighborhood N_s with "radius" r (e.g., for $r = 1$ the $3 \times 3 \times 3$ cube centered at p). The "influence" is computed as the product of two Gaussian kernels, one in the spatial domain (G_{σ_d}) and one in the photometric range (G_{σ_r}). These Gaussian kernels attenuate the influence of nearby points such that those nearby in geometric or signal space have greater influence, while those further away in geometric or signal space have less influence. The filtered image I' is computed as

$$I'_p = \frac{1}{W_p} \sum_{q \in N_p} G_{\sigma_d}(\|p-q\|) G_{\sigma_r}(|I_p - I_q) I_q$$

where W_p is a normalization factor that sums all of the Gaussian weights,

$$W_p = \sum_{q \in N_p} G_{\sigma_d}(\|p-q\|) G_{\sigma_r}(|I_p - I_q|).$$

While it is possible to precompute the weights in W_p contributed by G_{σ_d}, the contributions from G_{σ_r} are not known *a priori* as they depend on the photometric values of a particular image.

In earlier work, we extended the original Tomasi bilateral filtering formulation for use on 3D volumetric data and compared its scalability and performance on shared- and distributed-memory platforms using several different parallel programming models and execution frameworks [3].

B. Anisotropic Diffusion

Anisotropic diffusion is a process similar to isotropic diffusion, or Gaussian smoothing, except that it makes use of an additional "conductance" function that regulates how much diffusion takes place at different locations in the data. This is useful for edge-detection and for edge-preserving denoising of images because the conductance function can be chosen to limit the diffusion at voxels with sudden changes in value (i.e. large gradients) and these areas typically correspond to edges.

The PDE for modeling anisotropic diffusion is similar to the heat equation,

$$\frac{\partial u}{\partial t} = \nabla^2 u = \nabla \cdot \nabla u,$$

but scales the gradient by the conductance function g to regulate the amount of diffusion,

$$\frac{\partial u}{\partial t} = \nabla \cdot (g(|\nabla u|) \nabla u).$$

In this formulation, the conductance function is a scalar-valued function whose domain is also scalar (i.e. it operates on the *magnitude* of the gradient $|\nabla u|$). Technically, this makes the Perona-Malik filter *isotropic*: it diffuses the same amount in all directions, but with different magnitudes at different voxels (i.e., *non-homogenous* might be a better term). A truly anisotropic filter would use a conductance *tensor* that operates on the vector-valued gradient. For example, the equation could be rewritten as

$$\frac{\partial u}{\partial t} = \nabla \cdot (D \nabla u),$$

for a matrix D with eigenvalues and eigenvectors that depend on ∇u. Setting an eigenvalue to 1 will cause diffusion to take place in the direction of its corresponding eigenvector. For edge-preserving diffusion in 2D, Weickert [4] suggests using the matrix

$$D = \begin{bmatrix} \vec{v}_1 & \vec{v}_2 \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} \vec{v}_1^T \\ \vec{v}_2^T \end{bmatrix}$$

constructed from the eigenvectors and eigenvalues

$$\vec{v}_1 \| \nabla u_\sigma, \quad \vec{v}_2 \perp \nabla u_\sigma, \quad \lambda_1 = g(|\nabla u_\sigma|), \quad \lambda_2 = 1,$$

defined on a Gaussian smoothed version of the image u_σ. This preliminary smoothing helps regularize the image to prevent artifacts in the numerical solution of the PDE, as proved by Catté et al [5].

Perona and Malik [6] give an inexact discretization of the continuous PDE for anisotropic diffusion by restricting the gradient approximation to a lattice. By inexact, they mean that they are actually solving a PDE that looks (in 2D) like

$$\frac{\partial u}{\partial t} = \nabla \cdot (g(u_x)u_x, g(u_y)u_y).$$

Their stencil operation for diffusing an image value at location p in image I is

$$I_p^{t+1} = I_p^t + \Delta t \sum_{q \in N_p} g(I_q^t - I_p^t)(I_q^t - I_p^t),$$

where Δt is the timestep and N_p is the neighborhood of points that are ± 1 lattice point from p in each direction: it is a 2-point stencil in 1D, 4-point in 2D, 6-point in 3D, etc. Perona and Malik claim that they noticed no difference between this simplified approach and more complicated approximations of the gradient. One of their motives for using a lattice is because it is the "natural choice for analog VLSI implementations." Of course, it is also natural to think of pixels or voxels in an image as nodes of a lattice.

Voci et al. [7] and Gerig et al. [8] suggest enhancing the numerical approximation by sampling from a larger neighborhood, for instance by including all of the diagonal lattice directions. This is an 8-point stencil in 2D or 26-point in 3D, and the additional accuracy comes with

a computational cost. A stencil operation of this type is similar to Perona and Malik's, but adds a normalization factor l_q that corrects for the length of the diagonal between p and q:

$$I_p^{t+1} = I_p^t + \Delta t \sum_{q \in N_p} \frac{g(I_q^t - I_p^t)}{l_q}(I_q^t - I_p^t).$$

The normalization factor also accomodates datasets with non-uniform elements, such as in 3D MR images where the resolution is lower in the z-dimension. However, some implementors warn that images acquired with excessively skewed ratios between resolutions, such as $1:2$ or greater, contain insufficient information to be used as 3D datasets in biomedical settings (see [9] pg. 242 for a discussion).

In contrast to Perona and Malik's "inexact" approach, Weickert and Benhamouda [10] describe a discretization that does calculate $g(|\nabla I|)$ but requires two stencil operations. The first pass computes the gradient approximation (by central difference), applies the conductance function, and stores the value in a temporary buffer image G:

$$G_p = g\left(\sum_{(q_+,q_-)\in N_p}\left(\frac{I_{q_+} - I_{q_-}}{2(l_{q_+} + l_{q_-})}\right)^2\right).$$

The pairs (q_+, q_-) are the locations on either side of p in a given lattice direction. Again, the stencil accomodates non-uniform data elements with the normalization factors $l_.$. The second pass uses these pre-computed conductance values in place of the conductance function in the previous stencils:

$$I_p^{t+1} = I_p^t + \Delta t \sum_{q \in N_p} \frac{(G_q^t - G_p^t)}{l_q}(I_q^t - I_p^t).$$

Weickert and Benhamouda also establish an upper limit on choosing a stable timestep for an N-dimensional image:

$$\Delta t < \frac{1}{\sum_{i=1}^{N} \frac{2}{l_i^2}},$$

where $\{l_1, \ldots, l_N\}$ are the normalization factors for each lattice direction. For an evenly spaced 3D grid, $\Delta t = 1/8$.

The original conductance functions given by Perona and Malik are

$$g_{PM1}(t) = e^{-|t|/\kappa^2},$$

$$g_{PM2}(t) = \frac{1}{1 + (t/\kappa)^2}.$$

The first function preserves high-contrast edges better than low-contrast ones, while the second preserves larger regions better than smaller ones. Voci et al. suggest using the Charbonnier conductance function,

$$g_{Char}(t) = \frac{1}{\sqrt{1 + (t/\kappa)^2}},$$

in place of Perona and Malik's second function.

Black et al. [11] recast the Perona–Malik filter in terms of *robust statistics* in order to derive even better edge-stopping conductance functions. The premise is that an image is made up of regions with fairly constant values separated by edges. Within a region, the difference between a given element and its neighbors is small and normally distributed. If a neighbor lies on the other side of an edge, however, the difference becomes large and is a statistical outlier of the distribution. Black et al. liken anisotropic diffusion to the problem of minimizing an error norm over the image. This connection suggets two other candidates for conductance functions. The first is derived from Tukey's biweight norm,

$$g_{Tukey}(t) = \begin{cases} \frac{1}{2}\left(1 - \frac{t^2}{5\kappa^2}\right)^2, & |t| \le \sqrt{5}\kappa, \\ 0, & \text{otherwise}, \end{cases}$$

and the second from the Huber min-max norm,

$$g_{Huber}(t) = \begin{cases} 1/\kappa, & |t| \le \kappa, \\ 1/|t|, & \text{otherwise}. \end{cases}$$

Black et al. demonstrate an image that after 100 iterations of diffusion has visibly sharper edges with the Tukey function than with the g_{PM2} function.

C. GPU Computing

A GPU implementation of bilateral filtering appears in [12]. Using a combination of vertex and fragments shaders, it performs filtering on a data structure called a "bilateral grid" that contains a reduced-size approximation of the original data, thereby achieving good performance and memory utilization. In contrast, our work is a direct implementation in CUDA of bilateral filtering in 3D, rather than an approximation, which is more appropriate for a clinical medical imaging application.

Jiang et al. [13] study autotuning of a matrix multiplication kernel on GPUs. Their system takes a matrix multiplication kernel written in the BrookGPU language [14], a portable high-level programming language for GPUs, and uses techniques to improve data reuse so as to leverage SIMD GPU instructions. They present an algorithm to search the tuning space that relies on a combination of algorithm- and platform-centric heuristics with adaptive search to find the combination of tuning parameters that results in optimal performance.

More recently, Ryoo presents a set of performance metrics to estimate the performance of a given optimization configuration for CUDA-based code running on a GPU [15]. They compute two metrics, efficiency and utilization, by examining developer-readable assembler and GPU resource utilization maps produced by the Nvidia CUDA compiler. The basic idea is to estimate values for each of these metrics by examining resource utilization maps. Then, to avoid an exhaustive search of the optimization space, they estimate the relative performance change by altering parameters that contribute to both metrics. They prune the size of the search space by examining only those configurations that have only high levels in both metrics.

Sumanweera et al. [16] present optimizations for an FFT kernel on the GPU. The implementation is based on OpenGL fragment and vertex shaders rather than CUDA. They solve a load balance problem, where the aim is to keep both stages of a two-stage algorithm filled with work, by automatically searching for the right balance between workload at each of these two processing stages and choosing the one with the best performance. Our work, in constrast, investigates optimization that might occur well before this type of autotuning. Namely, we evaluate the performance impact of fundamental algorithmic parameter choices.

Stone et al. [17] present results from optimizing a MRI reconstruction algorithm on a Quadro FX 5600. They compute anatomically constrained reconstructions of non-cartesian MRI data. Their CUDA-based algorithm performs a least-squares minimization using conjugate gradient iterations along with FFT, inverse FFT, BLAS and sparse BLAS operations. They experiment with a number of optimizations: register allocation, coalesced memory access, use of constant memory, fast math operations, and finally exhaustive search for optimal tiling, number of threads per block and loop unrolling factors. They report a speedup of $\approx 13\times$ over a CPU implementation for a 128^3 dataset. While we use similar GPU optimizations, we use a different stencil-based algorithm with a different data access pattern. From a MR imaging pipeline perspective, the bilateral or anisotropic diffusion filter could be used as a complementary post-processing technique after the data has been acquired and processed using their non-cartesian acquisition method.

III. Architecture

Our GPU implementations are written in the highly-threaded, data-parallel CUDA [18] language, which enables access to GPU architectural resources via a set of extensions to C. We implement a simple memory tiling scheme originally described by Rivera and Tseng [19] to accelerate 3D stencil computations by reusing data cached in a CUDA thread block's shared memory. In order to support bilateral filtering, our implementation handles stencil widths of arbitrary size. We also implement utility routines to transfer 3D datasets to and from CUDA global memory while maintaining the appropriate padding required by the stencil at boundary locations. Finally, we pad out the dataset in the fastest moving dimension to a multiple of the CUDA "warp" size of 32, which is the number of threads that are executed simaultaneously on each of the GPU's multi processors.

Our test platform is Tesla, a Sun Fire x4600-M2 connected to an NVIDIA QuadroPlex 2200-S4 located at the National Energy Research Scientific Computing Center (NERSC). The QuadroPlex unit contains four NVIDIA FX-5800 Quadro GPUs each with 240 "CUDA cores" across 30 "multi processors," which consist of 8 processors that each execute the same instruction. Instructions are batched into "warps" of 32, which complete every 4th clock cycle, in what NVIDIA terms Single Instruction Multiple Threads (SIMT). The FX 5800 has 4GB of GDDR3 memory with 102GB/s of theoretical memory bandwidth. We used a single GPU for our tests.

One disadvantage of using a GPU coprocessor to accelerate computations is the cost of transferring data between main memory on the host system and the GPU's memory. This takes place over a PCI-Express bus, which on Tesla is an 8 lane PCIe v1 bus with a maximum transfer rate of 2GB/s, a factor of $50\times$ less than the memory bandwidth of the onboard GPU memory. We measured the average time to copy the $181 \times 217 \times 181$ BrainWeb dataset (27.2MB after conversion to 4-byte float values) to GPU memory as 0.04s, and 0.06s to copy the data back after filtering, for a round trip time of 0.1s. This corresponds to a memory bandwidths of 678MB/s to and 452MB/s from the GPU memory. All runtimes we report later do *not* include this constant cost of data transfer, and measure only the runtime of the actual CUDA kernel.

IV. Results

BrainWeb [1], [20] provides a reference MR image and corresponding noisy images that have been artificially generated using a mathematical model of real-world noise. With this reference image, we can calculate the mean-squared error of the noisy image before and after applying our denoising filter.

For the bilateral filter, we chose to sweep three parameters: the filter half-radius r (i.e. spatial support), the standard deviation σ_d in the spatial domain, and the standard deviation σ_r in the image range.

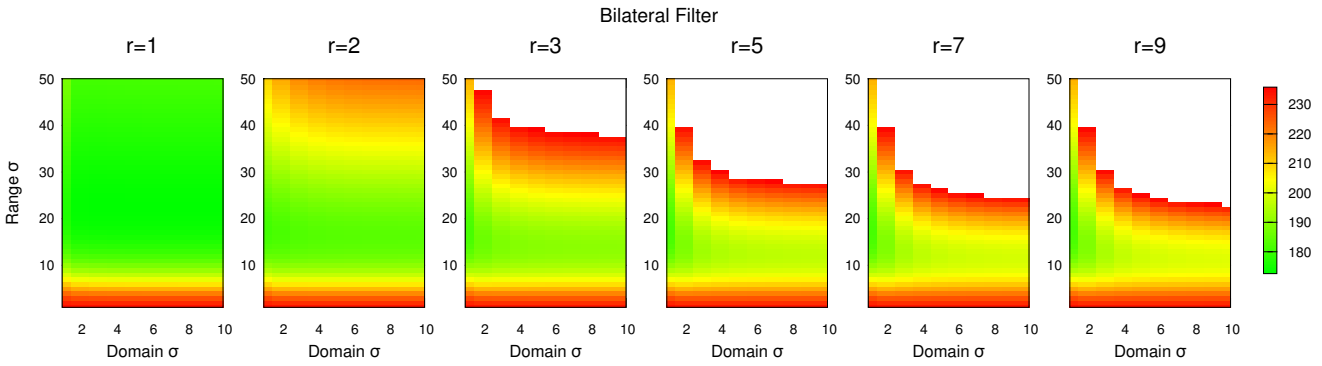

Fig. 2. MSE for the bilateral filter for a parameter sweep over the standard deviations σ_d and σ_r and filter radius r. White areas indicate that the filter increased rather than reduced MSE.

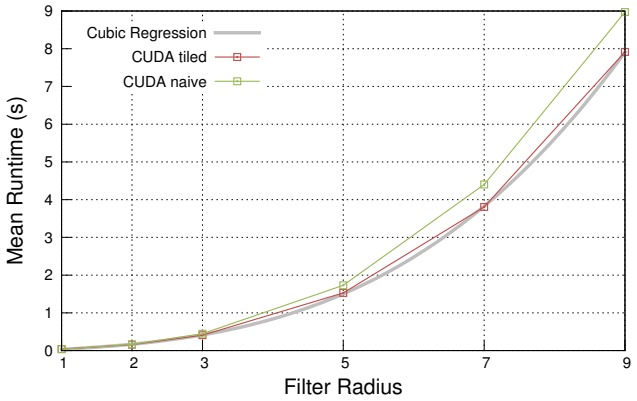

Fig. 3. Mean runtime for the bilateral filter by filter radius. There is only minor variation of runtime across the standard deviation (σ_d and σ_r) parameters.

For the anisotropic diffusion filter, we implemented three of the discrete forms of the PDE: the original form described by Perona and Malik, one with larger spatial support (26-point in 3D), and the two-pass form described by Weickert. Additionally, we implemented four conductance functions: the two given by Perona and Malik (g_{PM1} and g_{PM2}), the Charbonnier function g_{Char}, and the g_{Tukey} function given by Black et. al. For each combination of form and conductance function, we swept over the following parameters: the number of iterations N, the size of discrete timesteps Δt, and the κ value.

For a dataset I with dimensions $X \times Y \times Z$, we calculated MSE as

$$\frac{1}{XYZ}\sum\left(I'_s - I_s\right)^2$$

where I' is the filtered dataset and I is the noiseless reference dataset. Although we performed the filtering on normalized float data on the interval $[0,1]$, we scaled those values by 255 before calculating MSE to better represent the deviation in terms of 8-bit integer values (the original format of our MR images). We measured the MSE of the unfiltered, noisy dataset as 234.868.

In many configurations, the denoising filters *increased* the MSE of the final image relative to the reference image. These configurations appear as white regions in the heatmaps in Figures 2 and 4. In contrast, the configurations that reduced MSE the most appear as green regions, while red regions indicate where the MSE was unchanged by filtering.

The most effective configurations for both filters reduced the MSE of the noisy image by nearly 30%, from 234.868 to about 173 (see Tables I and II). Although the lowest MSE values were very close for the two filters, the bilateral filter ran nearly 5× faster. However, if we instead look for the anisotropic diffusion configuration that yielded the greatest reduction in MSE for the least cost in runtime (i.e. the best "value"), we see that an MSE of about 187 (within 10% of the lowest MSE) requires even less time (0.015 s). For the bilateral filter, the configurations with the best value were identical to those with the lowest MSE.

In general, the runtime of the bilateral filter scaled cubicly with filter radius (see Figure 3), although there was little practical benefit in running at higher radii since the best MSE reduction occurred at $r = 1$. In contrast, the anisotropic diffusion filter scaled linearly with number of iterations (see Figure 5), and achieved the lowest MSE with the largest number of iterations we tested. The choice of discrete scheme and conductance function for the anisotropic diffusion filter added a constant amount of variation in the linear scaling.

We also tested a more aggressive memory tiling optimization in which we preloaded a block of neighboring voxels into the GPU's shared memory to reduce the number of redundant memory loads among neighboring voxels. For larger stencil radii, where there is more

TABLE I
LOWEST MSE FOR BILATERAL FILTER

r	σ_d	σ_r	MSE	Runtime (s)
1	4	23	172.55	0.040
1	3	23	172.56	0.041
1	5	23	172.56	0.041

TABLE II
LOWEST MSE FOR ANISOTROPIC DIFFUSION

N	Δt	κ	Cond.	Scheme	MSE	Runtime (s)
9	0.025	0.01	PM2	Weickert	173.52	0.206
9	0.025	0.1	PM1	Weickert	174.15	0.207
7	0.025	0.01	PM2	Weickert	174.74	0.160

TABLE III
BEST MSE REDUCTION VS. RUNTIME FOR ANISOTROPIC DIFFUSION

N	Δt	κ	Cond.	Scheme	MSE	Runtime (s)
1	0.025	1	PM2	26-Point	188.66	0.015
1	0.025	1	Char	26-Point	188.87	0.015
1	0.025	10	PM2	26-Point	189.12	0.015

opportunity for cache reuse, this netted a small gain. Memory tiling atually led to worse performance with the anisotropic diffusion filter because the smaller stencil size (6-point or 26-point depending on the scheme) offers less opportunity for cache reuse, and the cost of the synchronization required after the shared load leads to a net loss in performance.

V. CONCLUSIONS AND FUTURE WORK

Our analysis of MSE reduction for the bilateral and anisotropic diffusion filters suggest that it is easy to choose parameters that either oversmooth or, especially in the case of the bilateral filter, provide diminishing returns for the cost in runtime.

In future work, we plan to perform an in-depth analysis of the tile dimensions used in our shared-memory cache. Using parameter sweeps similar to those for MSE presented in this paper, we identify the optimal tile dimensions for various GPU architectures, including the forthcoming Fermi architecture.

Additionally, we plan to assess the suitability of these denoising implementations and parameter choices for an image processing pipeline where we also perform segmentation and calculate validation scores.

ACKNOWLEDGMENT

This research used resources of the National Energy Research Scientific Computing Center (NERSC), which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. Sample data for this work was provided by the UC Davis Alzheimer's Disease Center, which is supported by the National Institutes of Health under grant No. P30 AG010129.

REFERENCES

[1] M. N. I. McConnel Brain Imaging Center, http://www.bic.mni.mcgill.ca/brainweb/.

[2] C. Tomasi and R. Manduchi, "Bilateral Filtering for Gray and Color Images," in *ICCV '98: Proceedings of the Sixth International Conference on Computer Vision*. Washington, DC, USA: IEEE Computer Society, 1998, p. 839.

[3] E. W. Bethel, "High Performance, Three-Dimensional Bilateral Filtering," Lawrence Berkeley National Laboratory, Tech. Rep. LBNL-1601E, 2009.

[4] J. Weickert, *Anisotropic Diffusion in Image Processing*. Stuttgart, Germany: B.G. Teubner, 1998.

[5] F. Catté, P.-L. Lions, J.-M. Morel, and T. Coll, "Image Selective Smoothing and Edge Detection by Nonlinear Diffusion," *SIAM Journal on Numerical Analysis*, vol. 29, no. 1, pp. 182–193, 1992.

[6] P. Perona and J. Malik, "Scale-Space and Edge Detection Using Anisotropic Diffusion," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 7, pp. 629–639, July 1990.

[7] F. Voci, S. Eiho, N. Sugimoto, and H. Sekiguchi, "Estimating the Gradient Threshold in the Perona-Malik Equation," *IEEE Signal Processing Magainze*, vol. 21, no. 3, pp. 39–46, May 2004.

[8] G. Gerig, O. Kübler, R. Kikinis, and F. A. Jolesz, "Nonlinear Anisotropic Filtering of MRI Data," *IEE Transactions on Medical Imaging*, vol. 11, no. 2, pp. 221–232, June 1992.

[9] L. Ibanez, W. Schroeder, L. Ng, and J. Cates, *The ITK Software Guide: The Insight Segmentation and Registration Toolkit*. Kitware, 2003.

[10] J. Weickert and B. Benhamouda, "Why the Perona-Malik filter works," Department of Computer Science, University of Copenhagen, Tech. Rep. DKIU-TR-97/22, 1997.

[11] M. J. Black, G. Sapiro, D. H. Marimont, and D. Heeger, "Robust Anisotropic Diffusion," *IEEE Transactions on Image Processing*, vol. 7, no. 3, pp. 421–432, March 1998.

[12] J. Chen, S. Paris, and F. Durand, "Real-time Edge-aware Image Processing with the Bilateral Grid," in *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*. New York, NY, USA: ACM, 2007, p. 103.

[13] C. Jiang and M. Snir, "Automatic Tuning Matrix Multiplication Performance on Graphics Hardware," in *Proceedings of the Fourteenth International Conference on Parallel Architecture and Compilation Techniques (PACT)*, Sep. 2005, pp. 185–196.

[14] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, "Brook for GPUs: Stream Computing on Graphics Hardware," in *Proceedings of ACM Siggraph*, August 2004.

[15] S. Ryoo, C. I. Rodrigues, S. S. Stone, S. S. Baghsorkhi, S.-Z. Ueng, J. A. Stratton, and W. mei W. Hwu, "Program optimization space pruning for a multithreaded GPU," in *CGO '08: Proceedings of the Sixth Annual IEEE/ACM International Symposium on Code Generation and Optimization*, 2008, pp. 195–204.

[16] T. Sumanaweera and D. Liu, "Medical Image Reconstruction with the FFT," in *GPU Gems 2*, M. Pharr, Ed. Addison Wesley, Mar. 2005, ch. 48, pp. 765–784.

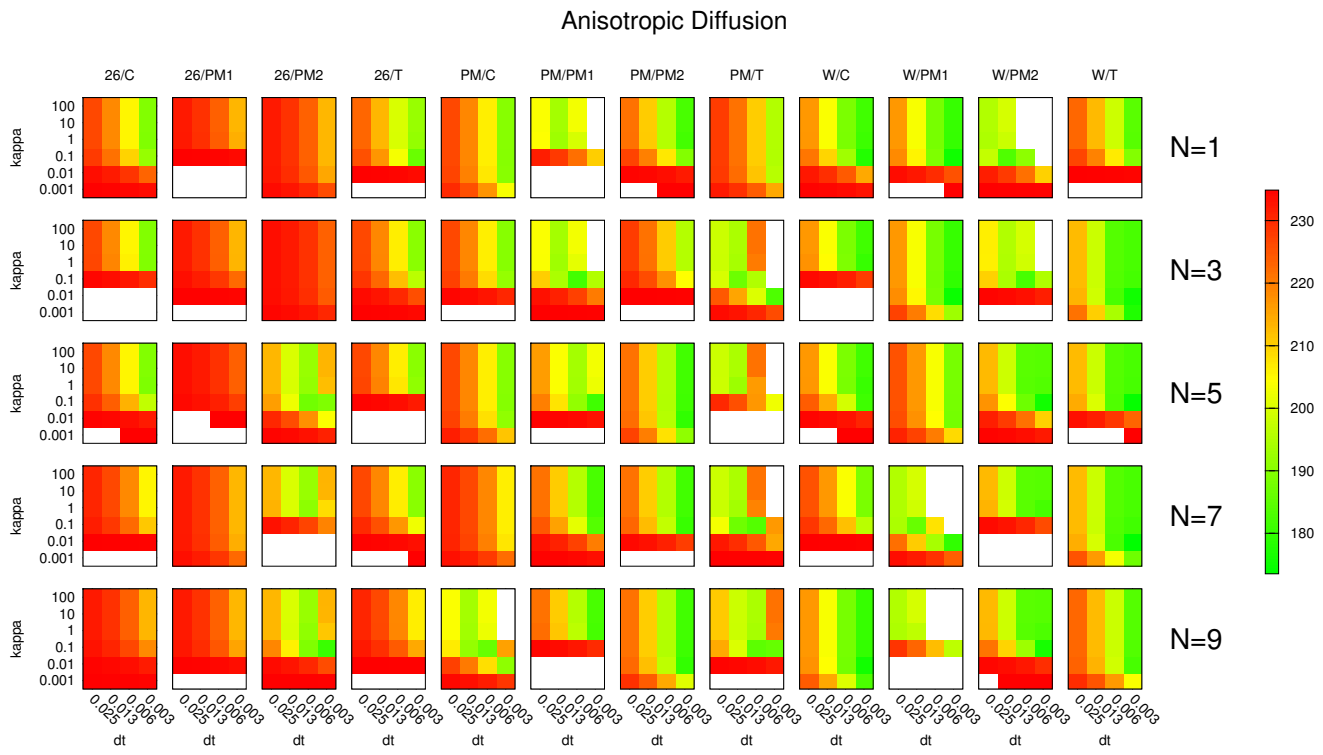

Fig. 4. MSE for the anisotropic diffusion filter for three discrete schemes: the 26-point stencil (26), Perona and Malik's (PM), and Weickert's (W); and four conductance functions: both of Perona and Malik's (PM1 and PM2), the Charbonnier (C), and the Tukey biweight norm (T). White areas indicate that the filter increased rather than reduced MSE.

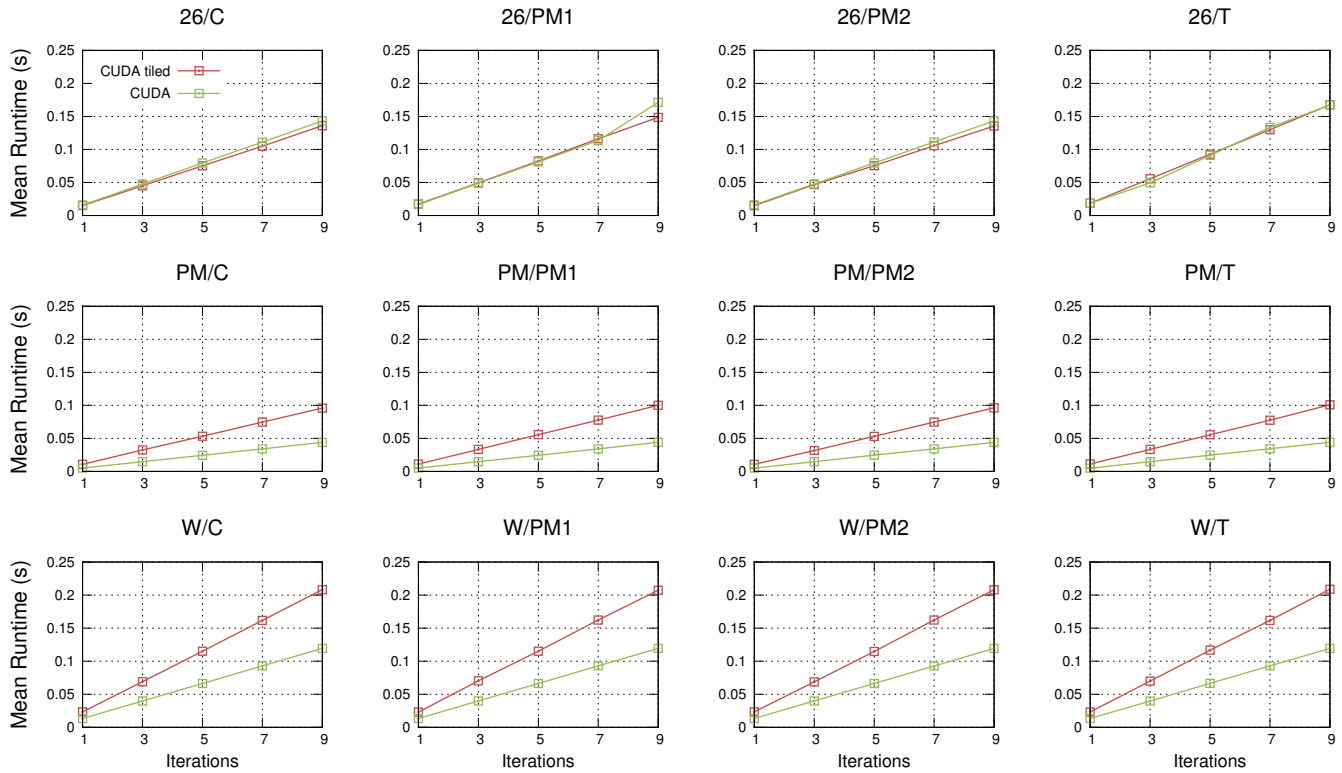

Fig. 5. Mean runtime for the anisotropic diffusion filter by number of iterations and the implementation and conductance function. There is only minor variation of runtime across the Δt and κ parameters.

[17] S. S. Stone, J. P. Haldar, S. C. Tsao, W. m. W. Hwu, B. P. Sutton, and Z. P. Liang, "Accelerating advanced mri reconstructions on gpus," *J. Parallel Distrib. Comput.*, vol. 68, no. 10, pp. 1307–1318, 2008.

[18] NVIDIA Corporation, *NVIDIA CUDA^{TM} Version 2.1 Programming Guide*, 2008.

[19] G. Rivera and C. Tseng, "Tiling optimizations for 3D scientific computations," in *SC '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing*, 2000.

[20] C. A. Cocosco, V. Kollokian, R. K.-S. Kwan, G. B. Pike, and A. C. Evans, "Brainweb: Online interface to a 3d mri simulated brain database," *NeuroImage*, vol. 5, p. 425, 1997.