Low Level Parallelization of Nonlinear Diffusion Filtering Algorithms for Cluster Computing Environments

David Slogsnat[1], Markus Fischer[1], Andrés Bruhn[2], Joachim Weickert[2], and Ulrich Brüning[1]

[1] Department of Computer Science and Engineering
University of Mannheim, 68131 Mannheim, Germany
slogsnat@uni-mannheim.de

[2] Faculty of Mathematics and Computer Science
Saarland University, Building 27.1, 66041 Saarbrücken, Germany

Abstract. This paper describes different low level parallelization strategies of a nonlinear diffusion filtering algorithm for digital image denoising. The nonlinear diffusion method uses a so-called additive operator splitting (AOS) scheme. This algorithm is very efficient, but requires frequent data exchanges. Our focus was to provide different data decomposition techniques which allow for achieving high efficiency for different hardware platforms. Depending on the available communication performance, our parallelization schemes allow for high scalability when using fast System Area Networks (SAN), but also provide significant performance enhancements on slower interconnects by optimizing data structures and communication patterns. Performance results are presented for a variety of commodity hardware platforms. Our most important result is a speedup factor of 210 using 256 processors of a high end cluster equipped with Myrinet.

1 Introduction

Over the recent years, distributed processing has been a powerful method to reduce the execution time of computationally intensive applications. Using components off the shelf (COTS) in combination with a fast network has become a viable approach to achieve high performance known from supercomputers, however at a fraction of the cost. The platform independent message passing interface MPI [7] allows applications to run on a variety of systems. This reduces the overhead of porting applications significantly.

Image processing is becoming a more and more important application area. Variational segmentation and nonlinear diffusion approaches have been very active research fields in the area of image processing and computer vision as well. Motivating factors for this research on PDE (partial differential equation) based models are for example the continuous simplification of images or shapes which help to understand what is depicted in the image. Image enhancement including denoising, edge enhancement, active contours and surfaces or closing interrupted lines are other points of interest. Within this context efficient numerical algorithms have been developed. They are the basis for this work, which focuses on reducing the execution time using distributed processing. AOS schemes for nonlinear diffusion filtering have been applied successfully on a parallel system with shared memory [2] and a moderate processor count. This motivated us to investigate their suitability on distributed memory systems with a large number of processors to enable close-to-real-time processing. One specific application area which we would like to address is the analysis of 2D and 3D medical images. For this scenario the approach of using clusters as computing resources gives a dynamic range from low- to high-end systems.

In the following chapter, we will describe the algorithm and how it was parallelized. Chapter 3 briefly describes the clusters that have been used as testbeds. In Chapter 4 we will present the performance results. Chapter 5 concludes our description.

2 Parallelization of the Algorithm

This section describes the algorithm and explains two possible solutions for data decomposition, a dimension segmentation and a mesh segmentation method. The algorithm is described for the three dimensional case, the two dimensional case is analogous. Figure 2 shows a filtered 2D image, figure 1 shows the original image.

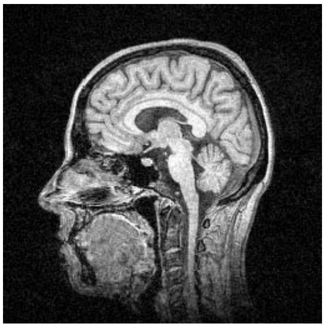

Fig. 1. Image with Noise

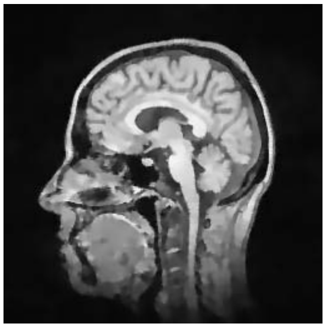

Fig. 2. Image after Filtering

2.1 Algorithm Components and Constraints

In the last decade, PDE based models have become very popular in the fields of image processing and computer vision. The nonlinear diffusion models as we know them today were first introduced by a work of Perona and Malik [3], who developed a model that allows denoising images while retaining and enhancing edges. This model has been improved by Catté et al. [11] from both a theoretical and practical viewpoint. Anisotropic extensions with a diffusion tensor are described in [5].

In practice, nonlinear diffusion filters require numerical approximations. In [1] a finite difference scheme based on an additive operator splitting (AOS) technique [4] is used for this purpose. This AOS technique is the basis for our parallelization efforts. Typical AOS schemes are one order of magnitude more efficient than simple diffusion algorithms.

The AOS scheme is applied in an iterative way until the desired level of denoising is reached. The AOS iteration loop starts with a Gaussian convolution, which needs to be processed for each dimension. Based on this smoothed image, derivatives are computed that are of need to determine the diffusivity values. Finally the diffusion processes for all dimensions are calculated and the results are averaged in a final recombination step.

Gaussian Convolution. The Gaussian convolution is processed seperately for each dimension d $\in \{1,2,3\}$. At iteration step n, the algorithm computes a matrix M^n using the matrix from the previous step $n-1$, thus $M^n_{gauss} = f_{gauss}(M^{n-1})$. The convolution takes places for one dimension after another. The computation itself is a stencil operation with a stencil size of $1 \times g$ for every dimension, with a typical value for $g = 3$. As a consequence, communication takes place before the computation for each dimension, but not during the computation for a single dimension.

Diffusivity. The diffusivity is calculated from the output matrix of the Gaussian convolution step: $M^n_{diff} = f_{diff}(M^n_{gauss})$. In contrast to the Gaussian convolution, the diffusivity is not calculated seperately for every dimension, but only once for all dimensions. The stencil has a fixed size of $3 \times 3 \times 3$ pixels. As a result, communication between nodes has to be performed only once before the diffusity is calculated, but not during the calculation process.

Diffusion. The diffusion is computed from two matrices, those are the matrix from the previous iteration M^{n-1} and the output matrix of the diffusivity M^n_{diff}. To compute the diffusion, d tridiagonal linear systems have to be solved by a fast Gaussian algorithm, where every system describes diffusion in one of the dimensions. These systems can be solved independently of each other, although they are solved one after another for practical reasons. Also, they can again be decomposed into many small independent equation systems, where each system corresponds to a single line in diffusion direction. These lines are solved by an algorithm using first forward and then backward propagation. This means that if a line crosses process boundaries, these processes have to communicate with each other during the computational phase. This is different from the Gaussian convolution and diffusivity steps, where no communication is required.

Finally, all three resulting matrices have to be merged into the final matrix M^n by taking the average.

2.2 Data Decomposition

In the following we present two data decomposition methods for 2D and 3D scenarios that have both been applied to the algorithm. Figure 3 depicts schemes for the decomposition methods.

The first decomposition method is a *multidimensional slice decomposition* [14]. Given a number of NP processes and a discrete 3D image $f[x_1, x_2, x_3]$, $x_1 \in \{0, .., X_1 - 1\}$, $x_2 \in \{0, .., X_2 - 1\}$ and $x_3 \in \{0, .., X_3 - 1\}$ then the partial image $f[x_1, x_{2np}, x_3]$ with $x_{2np} \in \{X_2/NP * np, .., X_2/NP * (np + 1) - 1\}$, will be processed by processor np, np $\in \{0, .., NP - 1\}$. In a similar manner, the image can also by sliced up in one of the other two dimensions. Since the diffusion algorithm performs filters in all three dimensions, this means that all three decomposition types are used to reduce communication between processes. When the program switches between filters of different dimensions, the direction of decomposition is changed accordingly. However, this implies that an all-to-all communication has to take place with a total traffic of $\frac{NP-1}{NP} * Imagesize$ pixels, since every processor has to aquire the actual values for all pixels in the new slice.

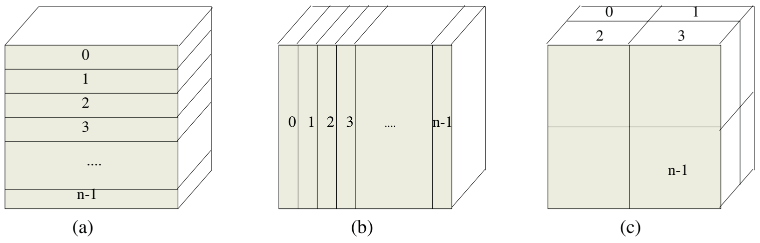

Fig. 3. Column (a) and Row- (b) versus Mesh Decomposition (c)

The second method is a *mesh partitioning* which divides the data as shown in the figure. Mesh partitioning has the advantage that it requires less interconnect performance than the slice decomposition method. In contrast to slice partitioning, the image does not have to be redistributed among the processors, but only the borders of the meshes have to be exchanged with the neighboring meshes. This leads to a lower amount of data that has to be communicated. Also, communication has only to be performed with a small number of neighbors in contrast to the all-to-all communication required by the slice decomposition method.

However, mesh partitioning has a downside too. Communication between processes takes places during computational steps, rather than in-between them. This increases the chance of processes waiting for results of neighboring processes. This problem can be reduced by means of pipeling. This has been implemented for the diffusion step of the algorithm: the processing of different lines is interleaved.

3 Available Cluster Environments

Ethernet type solutions for a network of workstations or a small cluster are currently still being considered as a solution for distributed computing. One of the reasons is that it is available as a built-in solution with no additional costs, and applications using sockets will work on any type of Ethernet ranging from Fast Ethernet to Gigabit Ethernet. In contrast to Ethernet network interfaces, System Area Networks (SANs) always provide direct user access. Also, SAN network adapters implement reliable transport services directly in hardware. They also deliver very high bandwidths (typically more than 1 GB/s) with very low latency. Currently, the most popular SAN is Myrinet from Myricom[8]. Other SAN implementations are the Scalable Coherent Interface (SCI) from Dolphin[9], Qsnet from Quadrics[10] and ATOLL from the University of Mannheim[13]. Our goal was to provide efficient implementations on clusters, using a variety of interconnection networks. Although SANs are clearly the choice of system for parallel processing, we analyzed our algorithm on Ethernet clusters too. The reason to do so is to provide an algorithm that performs best on any type of cluster, not only on dedicated high performance systems.

In the following, we will give a short overview of the platforms which were available for our performance evaluations.

Paderborn Center for Parallel Computing, Germany. The PC2 hosts a hpcLine system which includes 96 nodes, consisting of 2 PIII 850 Mhz CPU's each. A Linux 2.4 is running in SMP mode. Besides regular Ethernet, a faster interconnect is available with the SCI network plugged into a 32Bit/33Mhz PCI interface. The MPI library is a commercial MPI implementation from Scali, Norway, called Scampi.

Real World Computing Partnership, Japan. The ScoreIII cluster consists of 524 nodes, two PIII 933 Mhz processors each, running a modified Linux 2.4 SMP Kernel. The Cluster is fully interconnected to a CLOS network using a Myrinet2000 network interface. Although Myrinet offers a universal 64Bit/66Mhz interface, the motherboard only supports the 64Bit/33Mhz mode. The cluster makes use of the Score cluster system software[6], which provides fast access to both Myrinet and Ethernet devices. The cluster is ranked 90th in the November 2002 Top500 Supercomputer List.

FMI Passau, Germany. The FMI in Passau is also equipped with a hpcLine, offering latest SCI cards with 64Bit/66Mhz. The dual Intel PIII 1000 Mhz Processors are running a Linux 2.4 Kernel. Just as the PC2 cluster, the FMI uses the Scampi environment.

All clusters do also have Fast Ethernet network interfaces.

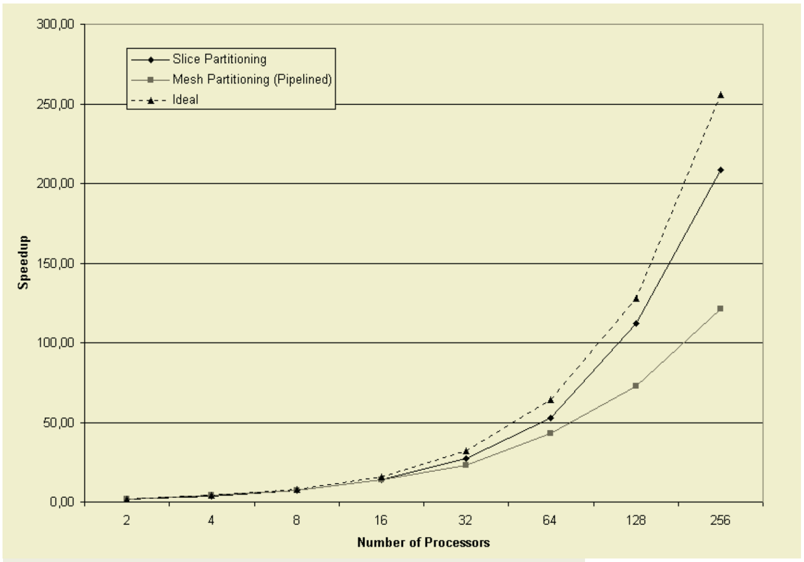

Fig. 4. Speedup on Myrinet

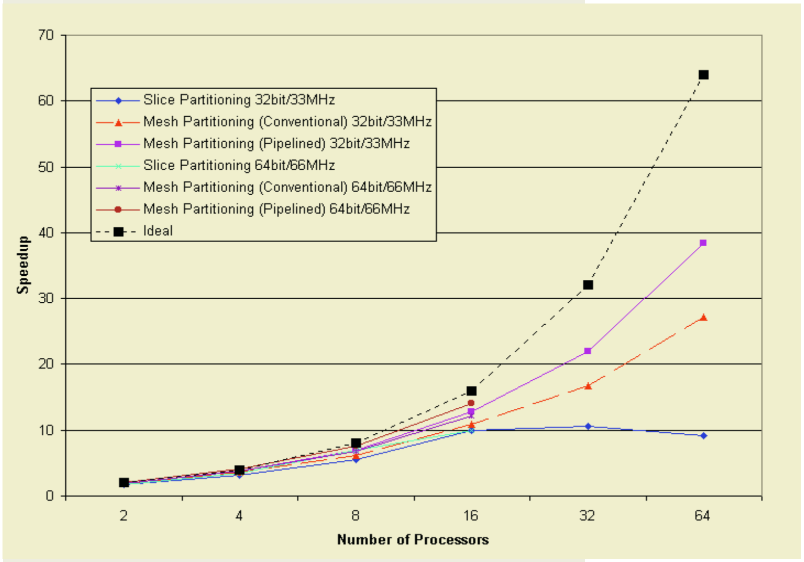

Fig. 5. Speedup on SCI with 32bit/33MHz PCI and 64bit/66MHz PCI

4 Performance

In the following, the results of a number of different versions of the algorithm will be presented. All are using non blocking communication calls. In contrast to collective scatter communication, non blocking communication allows for better scalability with the system size. We expected a performance difference when switching from a SAN to Fast Ethernet and we were interested in breaking down the ratio of communication and computation.

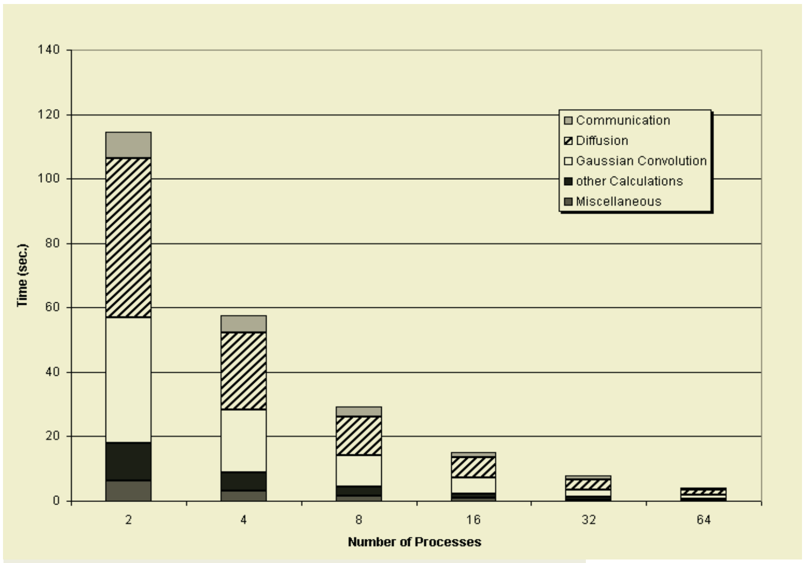

Fig. 6. Myrinet execution time breakdown using slice partitioning

As expected, the highest speedup can be achieved on Myrinet clusters. As shown in figure 4, a speedup factor of 210 on 256 processors using the slice decomposition method reveals an almost linear increase in performance. Mesh partitioning scales well too, but it is by far outperformed by the slice decomposition method, which is up to 50% faster than mesh partitioning. Much to our surprise, we observed a contrarian scaling behaviour on SCI, illustrated in figure 5. While mesh partitioning scales almost as good on SCI as on Myrinet, slice partitioning does not scale at all on SCI. The scaling behaviour using Ethernet is very similar to the behaviour using SCI. However, the total speedup achieved is noticeable smaller: the use of 64 processors leads to a speedup factor of 22.

A breakdown of the total execution time into computation and communication helps to reveal bottlenecks. For Myrinet, figure 6 shows that the time spent for communication is negligible. In contrast, the communication overhead for SCI increases significantly, as shown in figure 7. According to our results and as indicated by benchmarks like the Effective Bandwidth Benchmark [12], the bandwidth of the SCI interface itself is not the bottleneck. We measured a peak bandwidth of 85 MB/s and a one-way latency of 5.1μs for the slower 32bit/33MHz PCI cards, which is more than sufficient for our algorithm. It is more likely that the SCI ring structure can not deal very well with high data traffic. On the unidirectional SCI ring, applications may also be hampered by

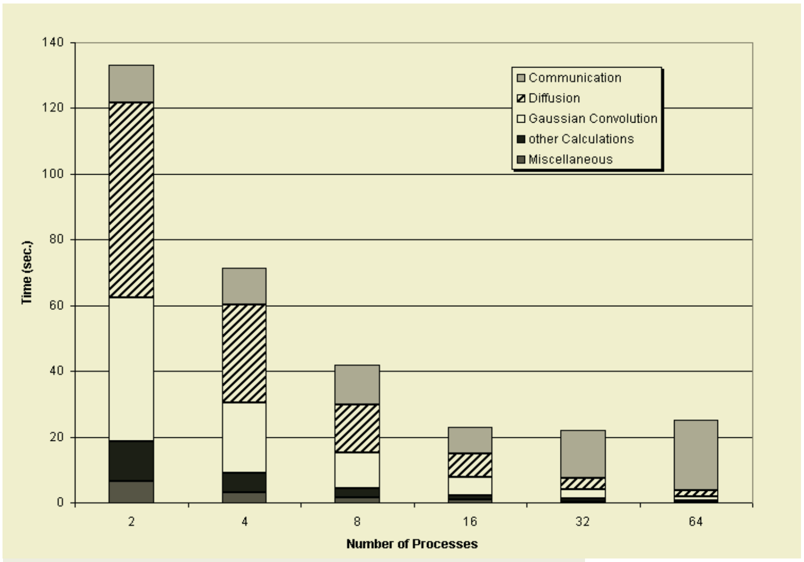

Fig. 7. SCI execution time breakdown using slice partitioning

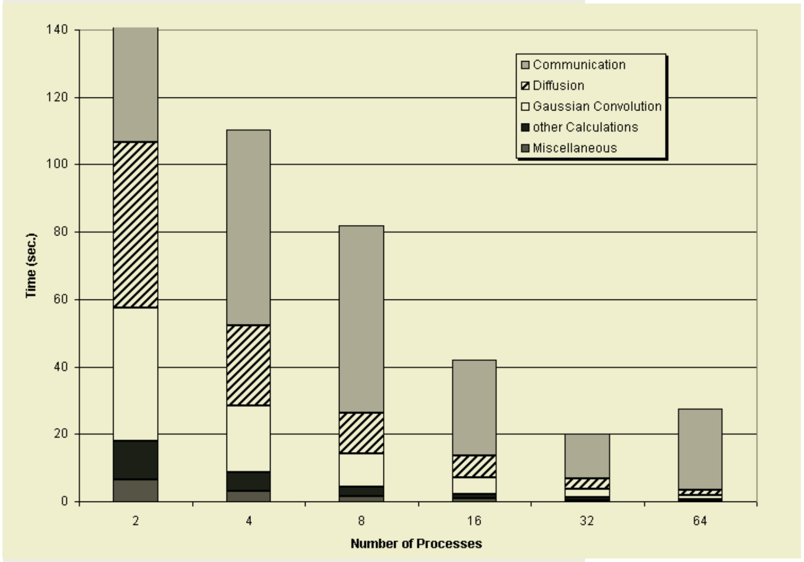

Fig. 8. Ethernet execution time breakdown using slice partitioning

other applications running on the cluster. Weak scalability is especially the case for low end networks such as Fast Ethernet. Figure 8 shows that the execution time of the algorithm is clearly dominated by the communication.

It can be concluded that using Myrinet, the communication overhead is negligible for our algorithms. On SCI and FastEthernet, communication is still the major bottleneck. Therefore mesh partitioning, solely optimized for a low interconnect usage, is the method of choice on the slower FastEthernet and SCI interconnects. Slice decomposition does not scale well on these interconnects.

5 Conclusion

We have shown the potential of cluster computing for parallelization of a nonlinear diffusion algorithm. Two major versions have been implemented and analyzed. With different variants, they offer solutions for both environments with highest network performance, as well as systems which are only loosely coupled. Our current focus is to generate an autonomous initial mapping based on cluster characteristics. This includes the processor performance as well as network features. Another investigation is to support heterogeneous computing resources in which processors obtain tasks based on their raw performance and current load. This will lead to an adaptive method which will enable the application to perform most efficient under several environments ranging from networks of workstations to tightly integrated systems.

Acknowledgements. We want to thank Andrés Bruhn and Timo Kohlberger for their valuable contributions, Joachim Weickert and Christoph Schnörr for the excellent cooperation in our joint project, and Tobias Jakob for his analysis of various data distribution schemes. Thanks go also to the operators of above clusters for granting us access to their systems. This work was sponsored by the DFG unter project number Schn 457/4-1.

References

[1] J. Weickert, B.M. ter Haar Romeny, M.A. Viergever. Efficient and reliable schemes for nonlinear diffusion filtering, *IEEE Transactions on Image Processing*, Vol. 7, 398–410, 1998.

[2] J. Weickert, J. Heers, C. Schnörr, K.J. Zuiderveld, O. Scherzer, H.S. Stiehl, Fast parallel algorithms for a broad class of nonlinear variational diffusion approaches, *Real-Time Imaging*, Vol. 7, 31–45, 2001.

[3] P. Perona and J. Malik. Scale space and edge detectioin using anisotropic diffusion *IEEE Trans. Pattern Anal. Mach. Intell. 12*, 629–639, 1990.

[4] T. Lu, P. Neittaanmaki and X.-C. Tai. A parallel splitting up method for partial differential equations and its application to Navier–Stokes equations, *RAIRO Mathematical Models and Numerical Analysis 26(6)*, 673–708, 1992.

[5] J. Weickert Anisotropic diffusion in image processing *Teubner*, 1998

[6] Y. Ishikawa, H. Tezuka, A. Hori, S. Sumimoto, T. Takahashi, F. O'Carroll, and H. Harada. RWC PC Cluster II and SCore Cluster System Software – High Performance Linux Cluster. *In Proceedings of the 5th Annual Linux Expo*, pages 55–62, 1999.

[7] Message Passing Interface *MIT Press*, 1994.
[8] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic and W. Su, Myrinet: A Gigabit-per-Second Local Area Network, *IEEE Micro*, Vol. 15, 29–36, 1995
[9] *IEEE Std for Scalable Coherent Interface (SCI).* Inst. of Electrical and Electronical Eng., Inc., New York, NY 10017, IEEE std 1596-1992, 1993.
[10] F. Petrini, A. Hoisie, W. Feng, and R. Graham. Performance Evaluation of the Quadrics Interconnection Network. *In Workshop on Communication Architecture for Clusters (CAC '01)*, San Francisco, CA, April 2001.
[11] F. Catté, P. L. Lions, J. M. Morel, and T. Coll. Image selective smoothing and edge detection by nonlinear diffusion. *SIAM J. Num. Anal.*, 29(1):182–193, 1992.
[12] K. Solchenbach. EMP: Benchmarking the Balance of Parallel Computers. *SPEC Workshop on Benchmarking Parallel and High-Performance Computing Systems*, Wuppertal, Germany, Sept. 13, 1999
[13] Ulrich Brüning, Holger Fröning, Patrick R. Schulz, Lars Rzymianowicz. ATOLL: Performance and Cost Optimization of a SAN Interconnect. *IASTED Parallel and Distributed Computing and Systems (PDCS)*, Nov. 4–6, 2002, Cambridge, USA
[14] A. Bruhn, T. Jakob, M. Fischer, T. Kohlberger, J. Weickert, U. Brüning, and C. Schnörr. Designing 3-d nonlinear diffusion filters for high performance cluster computing. *Pattern Recognition, Proc 24th DAGM Symposium*, volume 2449 of Lect. Not. Comp. Sci., pages 290–297, Zürich, Switzerland, 2002. Springer.