4-1-2003

Parallel Computation of the Topological Skeleton of Vector Fields

Thomas Wischgoll
Wright State University - Main Campus, thomas.wischgoll@wright.edu

Gerik Scheuermann

Parallel Computation of the Topological Skeleton of Vector Fields

Thomas Wischgoll Gerik Scheuermann
Department of Computer Science
University of Kaiserslautern
P.O. Box 3049, D-67653 Kaiserslautern, Germany
email: [wischgol|scheuer]@informatik.uni-kl.de

Keywords
vector field, 2D flow, parallel, streamline computation, Linux cluster, streamlines

Abstract

Vector fields occur in many of the problems in science and engineering. In combustion processes, for instance, vector fields describe the flow of the gas. This process can be enhanced using vector field visualization techniques. Also, wind tunnel experiments can be analyzed. An example is the design of an air wing. The wing can be optimized to create a smoother flow around it.

To analyze such kind of datasets topological methods that clearly show the whole structure of the vector field in one picture are a very good tool. During the last years, many extensions were proposed for this method. In addition to standard topological methods we also detect closed streamlines since they are a topological feature that completes the topological analysis. To accelerate the computation of such a topological analysis we developed a parallel method to reduce computational time. Therefore, we spread the computation of the separatrices of the topological skeleton to clients of a computer cluster.

To test our implementation we use a numerical simulation of a swirling jet with an inflow into a steady medium. We built two different Linux clusters as parallel test systems where we check the performance increase when adding more processors to the cluster. We show that we have a very low parallel overhead due to the neglectable communication expense of our implementation.

1. INTRODUCTION

One visualization method to analyze vector fields is to compute the topological skeleton. This topological skeleton consists mainly of the singularities, i.e. the critical points or zeros of the vector field, and the separatrices that connect these singularities. The separatrices are computed by starting a streamline at the saddle singularities displaced in both positive and negative eigendirections of the interpolating matrix of the vector field.

Since the number of separatrices may be large depending on the given vector field, the computation of the topological skeleton may take several minutes or even hours. Therefore we propose a parallel version of this algorithm to decrease computational time by distributing the computation to several clients.

There are mainly two tasks that have to be accomplished: first, the critical points have to be determined in order to identify the saddle singularities. Second, the separatrices need to be computed. Both tasks can be calculated individually so that these tasks can be spread to different clients. Then the clients can compute the whole streamline beginning at the given starting point and return the graphical result to the server. The server collects all the different separatrices to create the whole topological skeleton. To get the whole topological skeleton we use a streamline computation method that is able to detect closed streamlines. Closed streamlines are – in the same way as singularities – a topological feature of vector fields. Also, they can be a link between to separate parts of the topological skeleton. Consequently, the topological skeleton may be incomplete if we ignore this feature.

In both tasks the number of tasks is very high and the time that is necessary to fulfill the task is relatively low compared to the whole computational effort. Consequently, spreading these kinds of tasks to the clients results in a very good load balancing.

As a parallel machine we use Linux clusters because of the low price of standard PC components. The advantage is that the processors are faster then the ones of for instance an SGI/Cray T3E with the disadvantage of a slower communication between server and client. But altogether, a Linux cluster is the best way to get a great performance at a low price.

To test our implementation we use a numerical simulation of a swirling jet with an inflow into a steady medium. We built different Linux clusters as parallel test systems where we check the performance increase when adding more processors to the cluster. We show that we have a very low parallel overhead due to the neglectable communication expense of our implementation. Consequently, this implementation fully gains from the speed of every added processor which will be proven in several examples.

In the next section we summarize previous work, while section 3 gives some background about parallel machines. In section 4 we explain the parallel version of the algorithm. The results including performance tests are explained in section 5. Finally, we conclude in section 6 and give some ideas for improvements of our method.

2. RELATED WORK

Topological methods clearly depict the structure of the flow by connecting sources, sinks, and saddle singularities with separatrices. Critical points were first investigated by Perry [12][10][11], Dallmann [2], Chong [1] and others. The method itself was first introduced in visualization for two dimensional flows by Helman and Hesselink [6][5][7][8][4]. Several extensions to this method exist. Scheuermann et al. [14] extended the method to work on a bounded region. To get the whole topological skeleton of the vector field, points on the boundary have to be taken into account, also. These points are called boundary saddles. These topological methods have been widely applied in the last decade. We use a different streamline computation method [16] which is based on a Runge-Kutta ODE solver but detects closed streamlines while integrating to complete the topological analysis.

Sujudi et al. [15] present a method for computing streamlines in a parallel environment by splitting the dataset into several sub-domains. If the streamline leaves a sub-domain another process responsible for the actual domain has to continue the computation. Reinhard et al. [13] present a parallel rendering method that distributes tasks for each ray which has to be computed to the different processors of the parallel machine. A parallelization of line integral convolution is presented by Zöckler et al. [17] where the vector field is divided into several subdomains depending on the number of processors used.

3. PARALLEL MACHINES

In this section we briefly describe some parallel machines, the Cray/SGI T3E, the IBM RS/6000 SP, and Linux clusters. The first one uses a distributed shared memory concept while the other two ones do not share their memory at all.

The Cray/SGI T3E is available since 1996. It uses a virtual address space to access the memory that is spread among all nodes. The nodes use a DEC Alpha processor 21164. This processor consists of two integer and two floating point units with IEEE 64 bit arithmetic. It has an eight KB first level and 96 KB second level cache directly on the chip. The *GigaRing* technology based on the IEEE SCI standard is used to connect the nodes. Every node is bi-directional connected to its neighbor in a three dimensional network topology.

The IBM RS/6000 SP uses POWER4 microprocessors. This type of processor has an *SMP-on-a-chip* design. It consists of two 1.3 GHz processors including second level cache directly on one chip. Every node has its own memory. Up to sixteen nodes are grouped together in a network configuration using *high performance switches* (HPS).

With Linux clusters there are no restrictions concerning network topology, memory, or CPU speed. Almost every standard PC component can be used in a Linux cluster. Even several desktop Linux computers that are connected through an Ethernet can be called a Linux cluster. But usually the network is the bottleneck in such a system. Therefore faster network devices like for instance a GigaBit network device or Myrinet host interface is used. A flat, tree shaped network topology is possible for a Linux cluster. But especially with a greater number of nodes a network with routes from any to any other node is desirable to avoid collisions and facilitate faster transfers.

Because of their low prices and the great scalability Linux clusters become more and more popular. They also appear nowadays in the list of the top 500 Supercomputer Sites. The main advantage of Linux clusters is the low price of standard PC components. It is very extendable because there is no limit in the number of nodes used in the cluster. In principle, you only have to add a new computer to the network to increase computational power. The processors are faster than the ones used for the other two parallel machines. The advantage of both, the Cray/SGI T3E and the IBM RS/6000 SP, is the faster network. Both commercial systems are limited with respect to extendability.

4. PARALLEL ALGORITHM

To compute the topological skeleton [8] of a vector field we have to compute all the critical points that are present in the vector field. Since we only need the data of the cells, i.e. the position of the vertices and the vectors at these vertices, to determine if there exists a critical point inside the cell and where it is located, we can transfer these tasks to the various clients of the cluster. When a client receives the index of a cell it computes the critical point and returns the position and its type, if it has found one, to the server. All tasks are controlled by a scheduler which is a part of the server.

The scheduling of the tasks works as follows: the server creates one task for each cell containing the index of this cell and queues it in the scheduler. The scheduler itself checks if there are any tasks left and if there is any client that has finished its task yet. If there is more than one client without an active job, the fastest is chosen. Then the next task is sent to this client. The client receives this task, computes the critical point and sends it, if it has found one, back to the server and tells the scheduler that it has finished its job. Since the amount of data to control the clients and transfer the critical points back to the server is very low, we can fully benefit from the performance of each client.

After we computed all critical points we start streamlines at each saddle point in positive and negative eigendirection with respect to the matrix of the linear interpolant. In this paper we present an extended version of the topological skeleton which consists not only of the singularities and separatrices. Since closed streamlines usually act in the same way as singularities – they can either attract or repel the flow – they additionally have to be taken into account when drawing the topological skeleton. Otherwise there may be different parts of the topological skeleton that are not connected to each other.

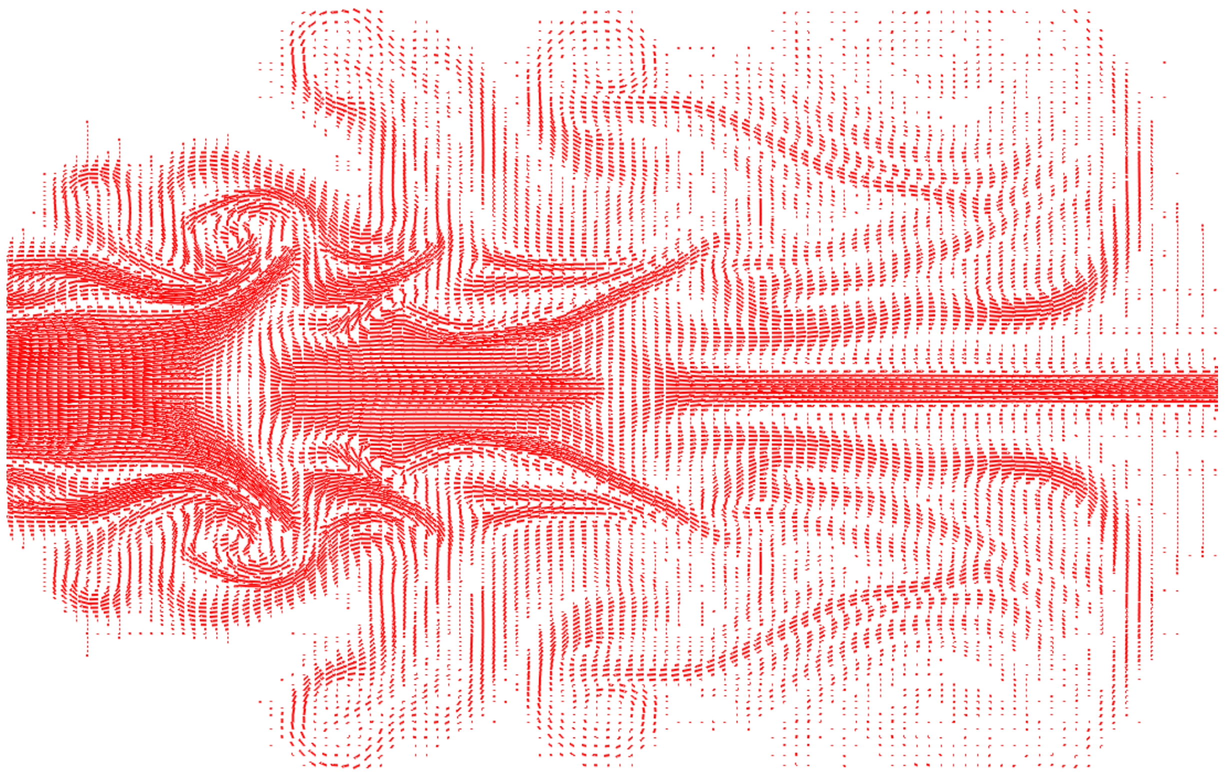

Figure 1: Vector field on a cutting plane of a swirling jet simulation visualized by a hedgehog

Therefore we use a streamline integrator that automatically detects closed streamlines[16]. When we encounter a closed streamline we can continue the topological analysis inside the closed streamline to get the whole topological skeleton. Consequently, we get a complete topological analysis of the vector field.

Computing streamlines is not a local task since the streamlines may cross any region of the flow. Therefore, we do not subdivide the data into several blocks like in some rendering tasks [9] or other flow visualizations [17]. In this case, such an approach would lead into an unbalanced parallel algorithm or the tasks need to switch their assigned part of the dataset quite often. This would cause much network traffic which usually slows down the algorithm. Our implementation uses a functional approach instead of dividing the dataset. We create several tasks each of them representing the whole computation of one streamline starting at a given position. Then we use the scheduler to distribute the tasks to the various clients of our cluster. The individual tasks are relatively small with respect to computational time compared to the computation of the whole topological skeleton. The scheduler provides every client that finished a job directly with the next one. Consequently, we achieve a very good load balancing because there is no client waiting during the whole computation.

Since the data of the vector field including octree and the program fit into 128 MB of RAM we decided to use a configuration where every client loads the whole dataset into its own memory. This facilitates the fastest possible access to the data. Since the server and every client loads the data at the same time there is no time lost because otherwise the clients would simply wait for the server until it has loaded the dataset. When dealing with larger datasets we have to use an out of core method which will be done in the future. But since memory is very cheap nowadays many cluster computers are equipped with up to 1 GB of RAM which allows to deal with big datasets.

Since we want to spread tasks that represent the whole computation of one streamline, each task contains two items: a point where the streamline has to start and the integration direction. The other data that is needed for the computation is already present at each client because the client has loaded the whole dataset yet. Due to the minimal amount of data of each task the communication cost which is produced by migrating tasks is very low.

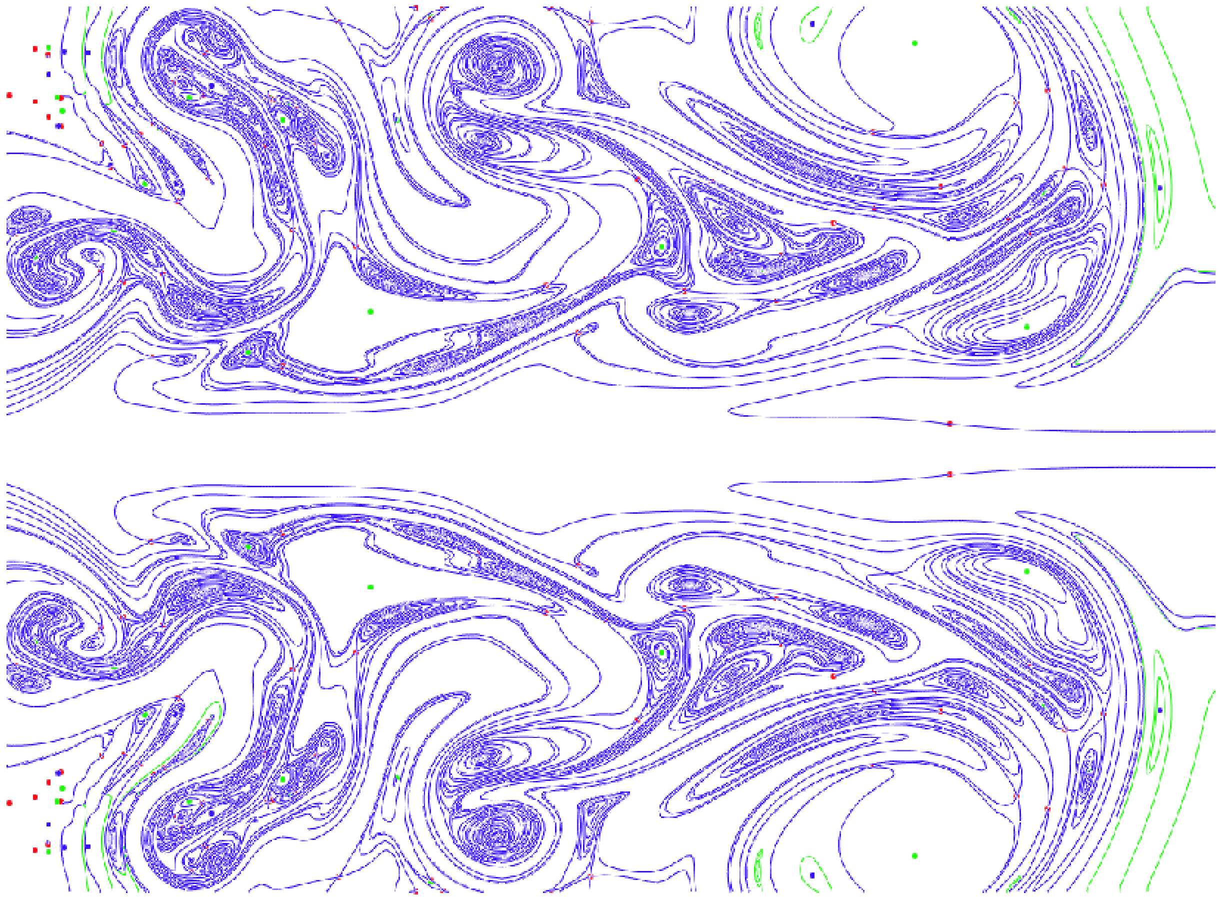

Figure 2: Topological skeleton of the swirling jet simulation

To distribute the tasks to the various clients we use the previously described scheduler: the server determines the start positions of the streamline using each saddle point found in the vector field. Then a task containing this start position and the integration direction is created and spooled into the queue of the scheduler, while the scheduler sends the next job to the fastest client that has no active job. The client receives this task, computes the separatrices and sends the result back to the server. Again, the amount of data to control the clients and transfer the separatrices back to the server is very low, so that we can fully benefit from the performance of each client.

5. RESULTS

Our algorithm is implemented in C++, while the server communicates with the clients using PVM[3]. The different tasks are encapsulated in C++-classes. This facilitates that the tasks can transfer itself to the client on demand and the clients only need to call a method to execute the received task.

To test the performance of our implementation we use mainly two different systems. One is a Linux cluster consisting of seven clients. Each node is equipped with an AMD Duron 600 or AMD Duron 700 processor and 128 MB of RAM. The server is a multiprocessor computer with two Pentium III 500 processors. The second system is based on some of our desktop computers with a Pentium II 350. We use Linux and normal PC components since this is a cheap way to get a great performance compared to other parallel computers. In order to get a more heterogeneous configuration we mix both systems by using all Linux computers available in our group for a last performance test.

The test dataset is a simulation of a swirling jet with an inflow into a steady medium. The simulation uses a cylindrical domain and assumes rotational symmetry, so that we are left with a two dimensional vector field on a plane through the center axis of the cylinder. This results in a vector field with high turbulence so that a lot of streamlines have to be computed for the topological skeleton. In figure 1 a hedgehog consisting of the vectors displayed as arrows is included. The vector field has 362 critical points and for the topology about six hundred streamlines have to be computed. Figure 2 shows the resulting topological skeleton of the vector field.

To determine the optimal timing of our algorithm we used the benchmark utility *nbench*[1] in order to get a suitable ratio between the speeds of the processors. *Nbench* is a port to Linux/Unix of release 2 of BYTE Magazine's BYTEmark benchmark program[2]. We computed the *floating-point index* of each processor which gives the relative speed of the

[1]http://www.tux.org/~mayer/linux/bmark.html
[2]http://www.byte.com/bmark/bmark.htm

floating-point unit compared to an AMD K6-233 processor. The results can be found in table 1. Using these values we computed the floating-point index of the whole parallel machine by summing up the indices corresponding to the involved processors. Knowing the speed of the whole cluster we can calculate the optimal runtime. If we neglect the communication between server and clients we only need to scale the runtime by multiplying the runtime on a single machine divided by the speed gain of the cluster compared to that single machine.

Table 1: Floating-point indices of the different processors

Processor	Floating-point index
Pentium II 350	2.404
Pentium III 500	3.561
AMD Athlon 650	5.163
AMD Duron 600	4.768
AMD Duron 700	5.547

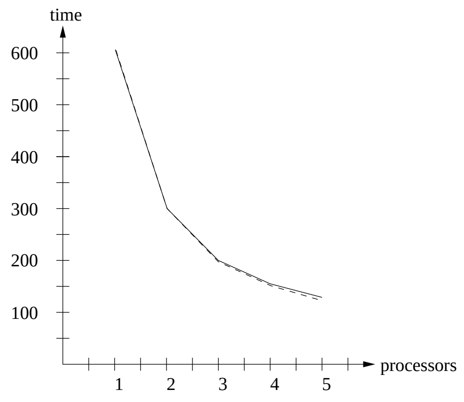

Figure 3: Time needed to compute topological skeleton using Pentium PII-350 processors displayed as graph

Table 2: Time needed to compute topological skeleton using Pentium PII-350 processors shown in a table

# CPUs	Time	Optimum
1	612s	—
2	306s	306s
3	205s	204s
4	158s	153s
5	134s	122s

Figure 3 and table 2 show the timings on the desktop computers. Up to five machines were used. The optimal timings are displayed using a dashed line while the real timings are shown by a solid line. This configuration is very suitable for testing the scalability of our implementation because every computer has identical performance. Obviously, the computation time is halved if the number of processors is doubled which indicates a good scalability of our implementation since they only differ slightly from the optimal ones.

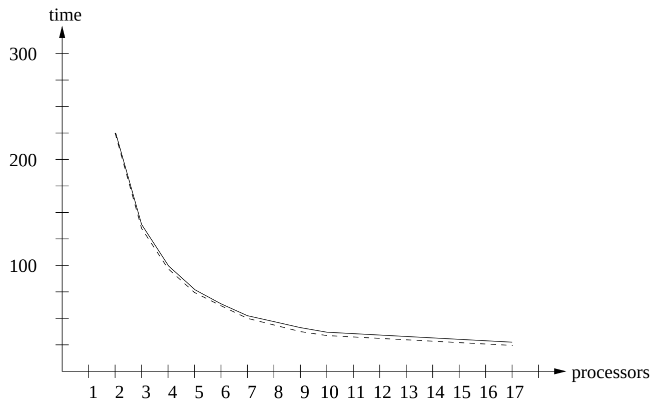

Figure 4: Time needed to compute topological skeleton using a Linux cluster with AMD Duron 600 and AMD Duron 700 processors displayed as graph

Table 3: Time needed to compute topological skeleton using a Linux cluster with AMD Duron 600 and AMD Duron 700 processors shown in a table

# CPUs	Time	Optimum
2	224s	—
3	138s	134s
4	99s	96s
5	77s	74s
6	63s	61s
7	53s	50s
8	46s	43s
9	39s	37s
17	28s	24s

The timings of the algorithm running on our Linux cluster with up to seven clients is displayed in figure 4 and table 3. Again, the optimal timings are displayed using a dashed line while the real timings are shown by a solid line. Since the server has two processors there are always running at least two tasks at the same time on this machine. Adding more clients to the Linux cluster the time needed for the algorithm is reduced correspondingly to the speed of its processor. Again, we can see that we nearly benefit from the full performance of each client due to the minimal communication between server and client as can be seen from the difference between the optimal and the real timings.

Our last test used all Linux machines of our visualization group. This resulted in a parallel machine consisting of six Pentium II-350, two AMD Athlon 650, one dual processor machine with two Pentium III-500, four AMD Duron 600, and three AMD Duron 700. Altogether, the algorithm used seventeen processors and it took 28 seconds to compute the whole topological skeleton of our test dataset. As expected, this is faster than using the cluster alone corresponding to the speed of the processors and slightly slower than the optimal runtime of 24 seconds. This also tests our implementation in a more heterogeneous parallel machine due to the different speeds of the processors. It shows that we can decrease the time needed for the computation by adding more processors no matter what sort of machine it is.

6. CONCLUSIONS AND FUTURE WORK

We have presented a parallelization of an algorithm that computes the topological skeleton of a vector field. The time needed for the computation is reciprocally proportional to the number of CPUs used in the cluster which gives a great performance enhancement when increasing the number of clients. Until the number of clients is lower than the number of streamlines that have to be computed, the overall performance of the cluster increases. Altogether, our implementation uses the full performance of the parallel machine.

There are only very few parallel systems available that are capable of computing streamlines in parallel. Since they use a different parallelization scheme – the dataset is usually divided – these systems tend to be more unbalanced when it comes to spreading the tasks compared to our implementation. Consequently, it takes more time to compute the topological skeleton with these systems. When dealing with vector fields with high turbulence closed streamlines are likely to occur. In these cases detecting this feature also speeds things up because standard integration methods usually use stopping criteria like for instance a maximal length of the streamline. As a consequence time is wasted by circling around the closed streamlines which is not the case for our implementation.

Since the clients in our cluster only have 128 MB of RAM we are currently working on an out of core method to cope with larger datasets compared to the one we used in this paper. When dealing with larger vector fields we can fully benefit from the performance increase of our method.

7. ACKNOWLEDGMENT

This research was supported by the DFG project "Visualisierung nicht-linearer Vektorfeldtopologie". Further, we like to thank Tom Bobach, Holger Burbach, Stefan Clauss, Jan Frey, Christoph Garth, Aragorn Rockstroh, René Schätzl and Xavier Tricoche for their programming efforts. The continuous support of all members of the computer graphics and visualization team in Kaiserslautern gives us a nice working environment. Wolfgang Kollmann, MAE Department of the University of California at Davis, provided us with the dataset.

REFERENCES

[1] M. S. Chong, A. E. Perry, and B. J. Cantwell. A General Classification of Three-Dimensional Flow Fields. *Physics of Fluids*, A2(5), pp. 765–777, 1990.

[2] U. Dallmann. Topological Structures of Three-Dimensional Flow Separations. Technical Report DFVLR-AVA Bericht Nr. 221-82 A 07, Deutsche Forschungs- und Versuchsanstalt fr Luft- und Raumfahrt e.V., April 1983.

[3] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine, A Users' Guide and Tutorial for Networked Parallel Computing*. The MIT Press, Cambridge, 1994.

[4] J. L. Helman. *Representation and Visualization of Vector Field Topology*. PhD thesis, Stanford University, 1997.

[5] J. L. Helman and L. Hesselink. Automated analysis of fluid flow topology. In *Three-Dimensional Visualization and Display Techniques, SPIE Proceedings Vol. 1083*, pp. 144–152, 1989.

[6] J. L. Helman and L. Hesselink. Representation and Display of Vector Field Topology in Fluid Flow Data Sets. *Computer*, 22(8), pp. 27–36, 1989.

[7] J. L. Helman and L. Hesselink. Surface Representations of Two- and Three-Dimensional Fluid Flow Topology. In G. M. Nielson and B. Shriver, editors, *Visualization in scientific computing*, pp. 6–13, Los Alamitos, CA, 1990.

[8] J. L. Helman and L. Hesselink. Visualizing Vector Field Topology in Fluid Flows. *IEEE Computer Graphics and Applications*, 11(3), pp. 36–46, May 1991.

[9] V. Isler, C. Aykanat, and B. Ozguc. Subdivision of 3D space based on the graph partitioning for parallel ray tracing. In P. Brunet and F. Jansen, editors, *Photorealistic Rendering in Computer Graphics. Proceedings of the Second Eurographics Workshop on Rendering*, pp. 182–90. Springer-Verlag, 1994.

[10] A. E. Perry. A Study of Non-degenerate Critical Points in Three-Dimensional Flow Fields. Technical Report DFVLR-FB 84-36, Deutsche Forschungs- und Versuchsanstalt fr Luft- und Raumfahrt e.V., 1984.

[11] A. E. Perry and M. S. Chong. A description of eddying motions and flow patterns using critical point concepts. *Ann. Rev. Fluid Mech.*, pp. 127–155, 1987.

[12] A. E. Perry and B. D. Fairly. Critical Points in Flow Patterns. *Advances in Geophysics*, 18B, pp. 299–315, 1974.

[13] E. Reinhard, A. Chalmers, and F. W. Jansen. Hybrid Scheduling for Parallel Rendering using Coherent Ray Tasks. In *Proceedings of IEEE Parallel Visualization and Graphics Symposium*. ACM SIGGRAPH, New York, 1999.

[14] G. Scheuermann, B. Hamann, K. I. Joy, and W. Kollmann. Visualizing local Vetor Field Topology. *Journal of Electronic Imaging*, 9(4), 2000.

[15] D. Sujudi and R. Haimes. Integration of Particle and Streamlines in a spatially-decomposed Computation. In *Proceedings of the Parallel Computational Fluid Dynamics*. IEEE Computer Society Press, Los Alamitos CA, 1996.

[16] T. Wischgoll and G. Scheuermann. Detection and Visualization of Closed Streamlines in Planar Flows. *IEEE Transactions on Visualization and Computer Graphics*, 7(2), 2001.

[17] M. Zöckler, D. Stalling, and H.-C. Hege. Parallel line integral convolution. In *Parallel Computing*, volume 23, 1996.