Distributed Computation of Planar Closed Streamlines

Thomas Wischgoll, Gerik Scheuermann, and Hans Hagen

University of Kaiserslautern, P.O. Box 3049, 67653 Kaiserslautern, Germany

ABSTRACT

Closed streamlines are an integral part of vector field topology, since they behave like sources respectively sinks but are often neither considered nor detected. If a streamline computation makes too many steps or takes too long, the computation is usually terminated without any answer on the final behavior of the streamline. We developed an algorithm that detects closed streamlines during the integration process. Since the detection of all closed streamlines in a vector field requires the computation of many streamlines we extend this algorithm to a parallel version to enhance computational speed.

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.

Keywords: vector field, 2D flow, parallel, streamline computation, Linux cluster, closed streamline, limit cycle

1. INTRODUCTION

An intuitive and often used method for vector field visualization is the calculation of streamlines. If one uses this technique in turbulent fields, one encounters often the problem of closed streamlines.

The difficulty with standard integration methods is that streamlines, approaching a closed curve, cycle around that curve without ever approaching a critical point or the boundary. Usually, one uses a stopping criteria like elapsed time or number of integration steps to prevent infinite loops. Instead, we present here a parallel version of an algorithm that detects this behavior and that can be used to visualize closed streamlines since these topological features are an essential topological property of the field. The algorithm uses the underlying grid to check if the same cell is crossed while integrating the streamline: this results in a cycle of cells. In that case, the algorithm determines if the streamline can leave this cell cycle or not. If it does not leave it is proven that there exists a closed streamline inside the cell cycle on condition that there is no critical point inside the involved cells.

To determine the closed streamlines of a vector field one has to compute many streamlines. In fact, we compute the topological skeleton. This is a graph which connects the critical points, where the vector field is zero, with streamlines called separatrices. This graph leads us to the closed streamlines. Since the number of streamlines may be large depending on the given vector field, this may take several minutes or even hours. Therefore we propose a parallel version of this algorithm to decrease computational time by distributing the streamline computation to several clients.

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.

In the next section we summarize previous work, while section 3 gives some theoretical background. 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.

Further author information: (Send correspondence to T.W.)
T.W.: E-mail: wischgol@informatik.uni-kl.de, Telephone: +49 631 205 3800
G.S.: E-mail: scheuer@informatik.uni-kl.de, Telephone: +49 631 205 3899
H.H.: E-mail: hagen @informatik.uni-kl.de, Telephone: +49 631 205 4072

2. RELATED WORK

Previously, two of the authors[1] published an algorithm that computes streamlines while detecting if it runs into a limit cycle in two dimensional flows. Haimes[2] discusses a similar problem where residence time is used to find recirculation regions. When reaching a closed streamline the residence time is infinite. The problem with closed streamlines is also related to the study of dynamical systems,[3 4] which have also been an application area for visualization. In the numerical literature, we can find several algorithms for the calculation of closed curves in dynamical systems,[5 6] but these algorithms are tailored to deal with smooth dynamical systems where a closed form solution is given. In contrast, visualization faces far more often piecewise linear or bilinear vector fields. Here, the knowledge of the grid and the linear structure of the field in the cells allow a direct approach for the search of closed streamlines.

Sujudi et al.[7] 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.[8] 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.[9] where the vector field is divided into several subdomains depending on the number of processors used.

3. THEORY

The topological analysis of vector fields considers the asymptotic behavior of streamlines. The origin set or α-limit set of a streamline c is defined by

$$\{p \in \mathbb{R}^2 | \exists (t_n)_{n=0}^{\infty} \subset \mathbb{R}, t_n \to -\infty, \lim_{n\to\infty} c(t_n) \to p\}\,.$$

The end set or ω-limit set of a streamline α is defined by

$$\{p \in \mathbb{R}^2 | \exists (t_n)_{n=0}^{\infty} \subset \mathbb{R}, t_n \to \infty, \lim_{n\to\infty} c(t_n) \to p\}\,.$$

If the α- or ω-limit set of a streamline consists of only one point, this point is a critical point or a point at the boundary ∂D of our domain D. (It is assumed that the streamline stays at the boundary point forever in this notation.) The critical points can be clearly identified because they are simply the zeros of the vector field.

The most common case of an α- or ω-limit set in a planar vector field containing more than one inner point of the domain is a closed streamline.[4] This is a streamline c_a, so that there is a $t_0 \in \mathbb{R}$ with

$$c_a(t + nt_0) = c_a(t) \quad \forall\, n \in \mathbb{N}\,.$$

4. PARALLEL ALGORITHM

In principle, the algorithm computes the topological skeleton[10] of the vector field which automatically leads to the closed streamlines. While integrating the streamlines we check if we run into a closed streamline. The basic idea is to determine a region of the vector field that is never left by a streamline. In case of a continuous vector field the Poincaré-Bendixson-Theorem ensures that this streamline approaches a closed streamline if no critical point exists in that region. We assume that the data of the vector field is given on a grid consisting of triangles and/or quadrilaterals. The vectors inside a cell are interpolated linearly resp. bilinearly so that we get a continuous vector field as needed for the theorem.

A streamline approaching a closed streamline has to reenter the same cell again. In this case we check if the cells were crossed by the streamline in the same order for the last two turns. This results in a *cell cycle* which identifies the above mentioned region. To examine if this cell cycle is left by the streamline we detect possible changes by checking the edges of the cells of the cell cycle. Therefore we identify points on each edge which we call *potential exits* where an outflow out of the cell cycle may occur in the vicinity. These points are identical with the vertices of the edge and points where the vector field is tangential to the edge.

Then we have to figure out if the actually investigated streamline will leave the cell cycle near such an exit. Therefore we integrate a streamline backwards from the potential exit to see if it leaves the cell cycle. If it does not leave after it crossed every cell of the cell cycle it converges to our streamline. We call this potential exit a *real exit* because the streamline will leave the cell cycle after a finite number of turns near that exit. Figure 1 displays an example for that case.

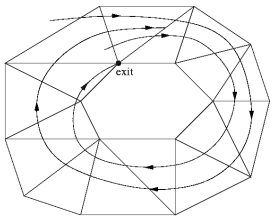

Figure 1: If a real exit can be reached, the streamline will leave the cell cycle.

If the backward integrated streamline leaves the cell cycle, there will also be an entry point as shown in figure 2. A streamline starting at that point cannot be crossed by our actually investigated streamline. Consequently we cannot leave the cell cycle at this exit.

If there is no real exit for the streamline, we have proven that the streamline will never leave the cell cycle. If there is no critical point inside the cell cycle the Poincaré-Bendixson-Theorem ensures that there exists a closed streamline in our cell cycle and the integral curve tends toward it.

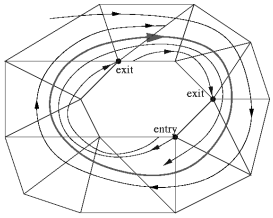

Figure 2: If no real exit can be reached, the streamline will approach a closed streamline.

To parallize this algorithm we have to compute all the critical points that are present in the vector field, first. 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 the clients receive the index of a cell they compute the critical point and return the position and its type, if they have 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 still 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 have 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 and check for closed streamlines while computing the streamlines as previously described.[1] 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.[11] Our implementation uses a functional approach where 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.

Since the data of the vector field including octree and the program fit into 64 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.

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.

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, searches for closed streamlines and sends it, if it has found one, back to the server. Again, the amount of data to control the clients and transfer the closed streamlines 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.[12] 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.

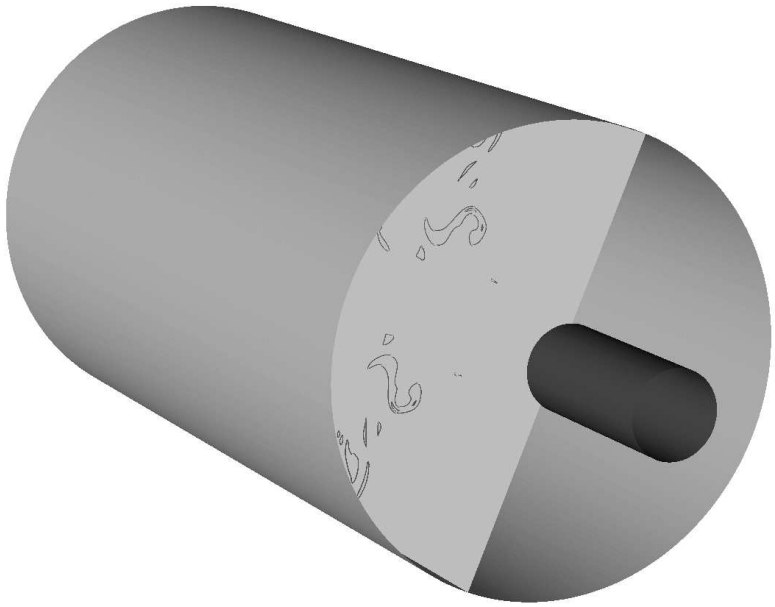

Figure 3: Configuration of the swirling jet simulation

To test the performance of our implementation we mainly use 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 64 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. In this application one is interested in investigating the turbulence of the vector field and in recirculation zones where the fluid stays very long. Swirling jets play a significant role in many combustion processes. It is important to find such recirculation regions indicated by closed instantaneous streamlines. This permits the conclusion that even in the three dimensional flow the fluid will stay there for a longer period of time. Figure 3 shows the configuration of this simulation. The jet is located in the front in the center of the cylindrical domain indicated by a small cylinder. The domain is displayed in light gray. The closed streamlines of that vector field found by our algorithm are shown on the cutting plane that divides the turbine diagonal into two halves. In figure 4 a hedgehog consisting of the vectors displayed

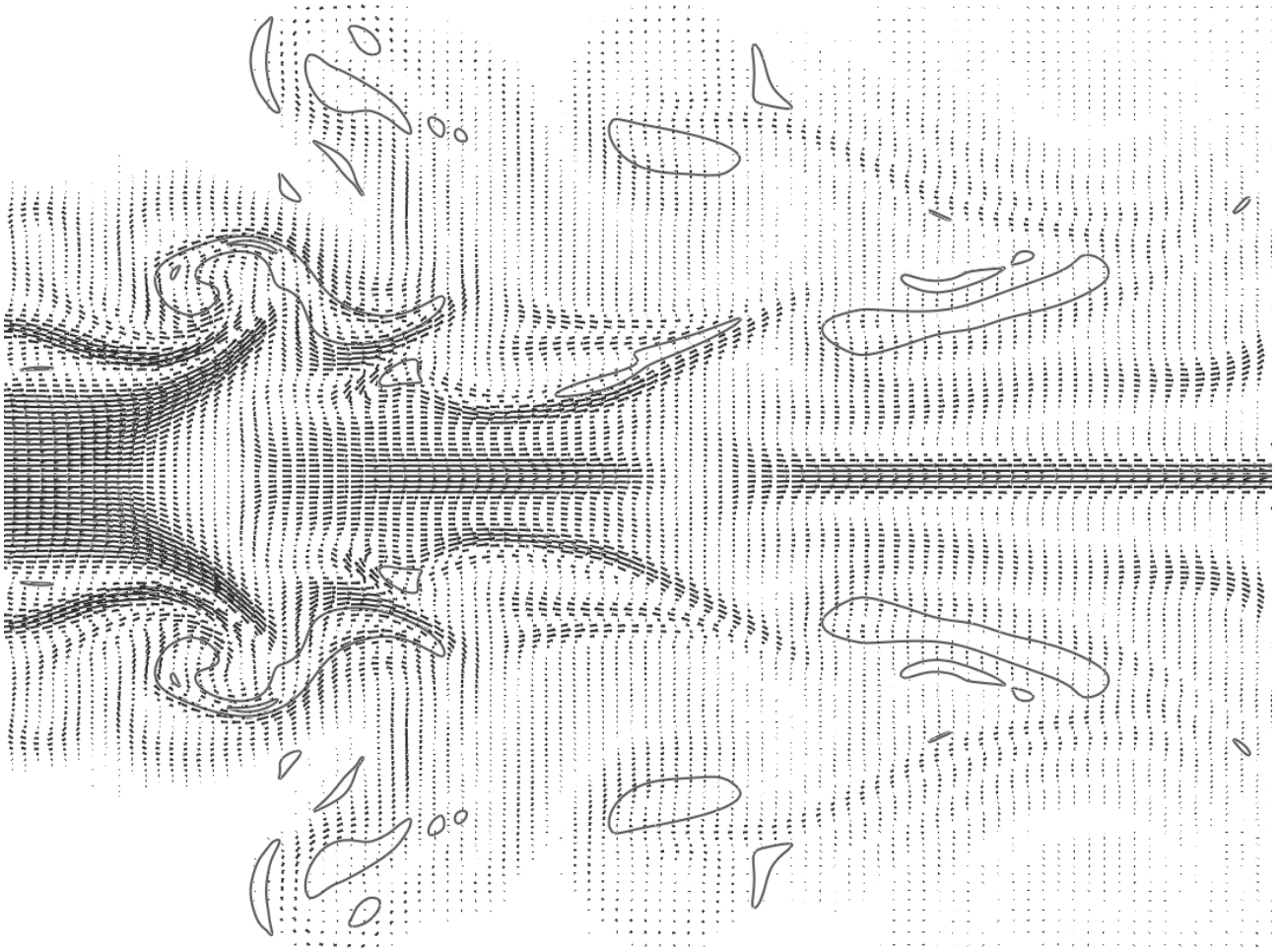

Figure 4: Closed streamlines including hedgehog on a cutting plane of a swirling jet simulation

as arrows is included. The vector field has 362 critical points and for the topology including closed streamlines about six hundred streamlines have to be computed.

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
Intel Celeron 800	6.125
AMD Thunderbird 1400	11.227

Figure 5: Floating-point indices of the different processors

To determine the optimal timing of our algorithm we used the benchmark utility *nbench*[*] 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

[*] http://www.tux.org/~mayer/linux/bmark.html

benchmark program[†]. We computed the *floating-point index* of each processor which gives the relative speed of the floating-point unit compared to an AMD K6-233 processor. The results can be found in figure 5. Using these values we computed the floating-point index of the whole parallel machine by summing up the indices corresponding to the involved processors and calculated the optimal runtime by neglecting the communication between server and clients.

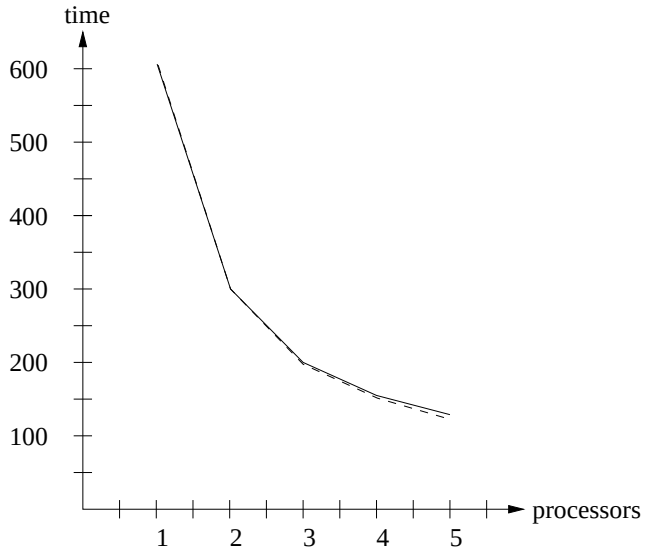

Figure 6: Time needed to compute closed streamlines using Pentium PII-350 processors displayed as graph

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

Figure 7: Time needed to compute closed streamlines using Pentium PII-350 processors shown in a table

Figures 6 and 7 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.

The timings of the algorithm running on our Linux cluster with up to seven clients is displayed in figures 8 and 9. 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.

In our next test we also used the Linux desktop machines in all the offices 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 all closed streamlines that are present in our test dataset. As expected, this is faster than

[†]http://www.byte.com/bmark/bmark.htm

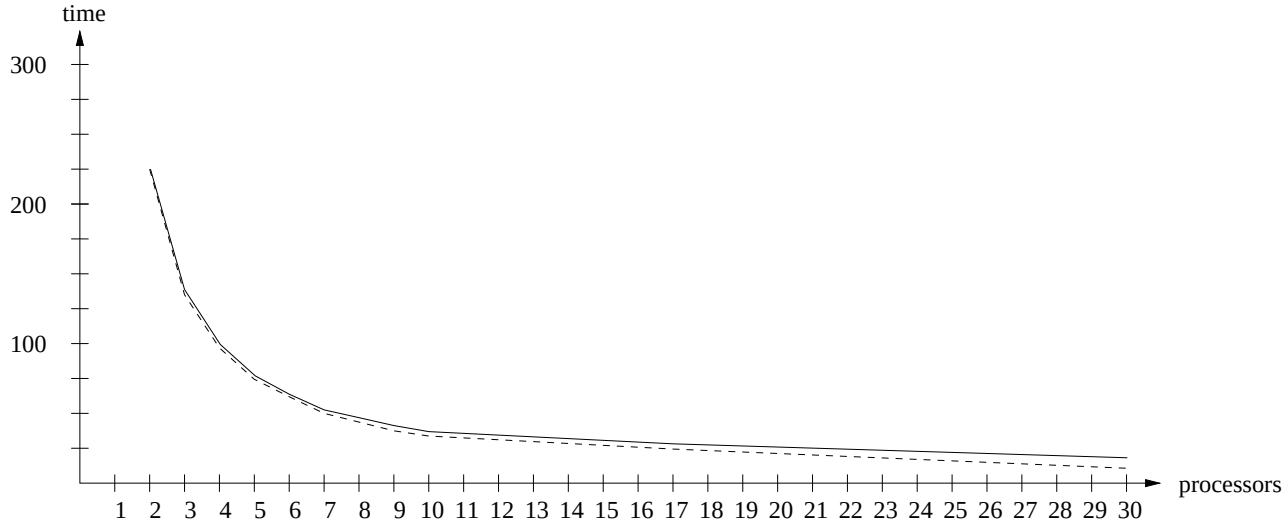

Figure 8. Time needed to compute closed streamlines using a Linux cluster with AMD Duron 600 and AMD Duron 700 processors displayed as graph

# 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
30	17s	9s

Figure 9. Time needed to compute closed streamlines using a Linux cluster with AMD Duron 600 and AMD Duron 700 processors shown in a table

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.

Then we also added the Linux machines in our student rooms for a last test. These are five machines equipped with an Intel Celeron 800, two machines with a Pentium III-500, and six with an AMD Thunderbird 1400 processor. So we end up with 30 processors. Our algorithm needed 17 seconds. Compared to the optimal timing of 9 seconds this is a little bit too slow. This is due to the slow network connection. Because all computers reside in different areas of our working group and several other processes such as network file system also use this network we do not have the full bandwidth available. Consequently, the communication costs are not neglectable anymore so that the real and the optimal timings differ.

6. CONCLUSIONS AND FUTURE WORK

We have presented a parallelization of our algorithm that detects closed streamlines. 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.

Since the clients in our cluster only have 64 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.

ACKNOWLEDGMENTS

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. T. Wischgoll and G. Scheuermann, “Detection and Visualization of Closed Streamlines in Planar Flows,” *IEEE Transactions on Visualization and Computer Graphics* **7**(2), 2001.
2. R. Haimes, “Using Residence Time for the Extraction of Recirculation Regions,” *AIAA Paper 99-3291* , 1999.
3. J. Guckenheimer and P. Holmes, *Dynamical Systems and Bifurcation of Vector Fields*, Springer, New York, 1983.
4. M. W. Hirsch and S. Smale, *Differential Equations, Dynamical Systems and Linear Algebra*, Academic Press, New York, 1974.
5. M. Jean, “Sur la méthode des sections pour la recherche de certaines solutions presque périodiques de syst‘emes forces periodiquement,” *International Journal on Non-Linear Mechanics* **15**, pp. 367 – 376, 1980.
6. M. van Veldhuizen, “A New Algorithm for the Numerical Approximation of an Invariant Curve,” *SIAM Journal on Scientific and Statistical Computing* **8**(6), pp. 951 – 962, 1987.
7. 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.
8. 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.
9. M. Zöckler, D. Stalling, and H.-C. Hege, “Parallel line integral convolution,” in *Parallel Computing*, **23**, 1996.
10. J. L. Helman and L. Hesselink, “Visualizing Vector Field Topology in Fluid Flows,” *IEEE Computer Graphics and Applications* **11**, pp. 36–46, May 1991.
11. V. Isler, C. Aykanat, and B. Ozguc, “Subdivision of 3d space based on the graph partitioning for parallel ray tracing,” in *Photorealistic Rendering in Computer Graphics. Proceedings of the Second Eurographics Workshop on Rendering*, P. Brunet and F. Jansen, eds., pp. 182–90, Springer-Verlag, 1994.
12. 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.