Eurographics Symposium on Parallel Graphics and Visualization (2016)
W. Bethel, E. Gobbetti (Editors)

A Scalable Streamline Generation Algorithm Via Flux-Based Isocontour Extraction

Ayan Biswas[†1], Richard Strelitz[‡2], Jonathan Woodring[§2], Chun-Ming Chen[¶1], and Han-Wei Shen[‖1]

[1]The Ohio State University, USA
[2]Los Alamos National Laboratories, USA

Abstract

Streamlines are commonly used for visualizing flow fields, but particle-tracing based streamline computation usually does not scale well as the data size and complexity increase. Large flow simulations like global ocean or climate models can obtain near perfect load balancing and the resulting data sets are generally analyzed in two dimensional slices. To match the computational properties of these simulations, we propose the use of flux-based stream functions for generating streamlines in parallel. In our method, local stream functions are efficiently generated per block based on flux conservation property, followed by low-cost communication of flux offsets among neighboring blocks. A scalar field is thus generated where streamlines can be extracted through parallel isocontouring. Experimental results show that our system offers higher streamline computation performance with higher scalability than traditional particle-tracing based method.

1. Introduction

The need to analyze large flow data sets has driven the researchers to strive for strategies that can generate streamlines in parallel. Traditionally, parallel streamlines are generated by parallelizing over the vector field domain and/or over the seeds, where the path of a streamline is computed serially by one processing element (PE) at a time (which may be passed from PE to PE). Generating a single streamline in parallel by multiple processing elements, has largely remained unsolved due to the dependency among the consecutive streamline segments. A parallel algorithm that can generate individual streamline segments simultaneously, will reduce the process of particle advection to only local computations and make it highly parallel. In this work, we present such a new parallel method to construct a streamline in parallel segments using a flux-based technique for large two dimensional flows. Analysis of large two dimensional flow fields is of prime importance and often the most preferred means in many research areas and applications, including ocean and climate modeling that explore large three dimensional data sets slice-by-slice. Our proposed algorithm addresses the needs of these scientific domains by exhibiting much improved execution time, scalability and load balancing compared to the traditional parallel advection methods.

Our reason for developing a new parallel streamline computation method is for scalability and to match the computational properties of the source simulations. Flow simulation models such as parallel ocean models are frequently computed using a finite-element (FE) or finite-volume (FV) method that scales well in parallel, by dividing the domain among the processing elements and they achieve nearly-perfect load balancing. With a FE/FV method, all processing elements have an equal amount of work through independent computation on several domains and bulk-synchronize through "ghost cell" communication.

The various known parallel streamline methods do not follow this computational and communication pattern, as they use a mixture of strategies to achieve run-time load balancing. Frequently, they shuffle the domain or seeds among processors. In contrast, our new parallel streamline algorithm matches the computational, communication, and load-balancing properties of the large flow simulation models

† email: biswas.36@osu.edu
‡ email: strelitz@lanl.gov
§ email: woodring@lanl.gov
¶ email: chen.1701@osu.edu
‖ email: shen.94@osu.edu

(e.g., Parallel Ocean Program from Los Alamos National Laboratory), by parallelizing over the domain, and it is able to utilize and run in the same parallel framework as the source simulation. We achieve this through a parallel flux computation and isosurfacing strategy.

Instead of computing streamlines as a series of integrations, an alternative solution is to calculate implicit streamlines by deriving a flux-based scalar field from a given vector field. Then the isocontours of this field represent the streamlines. Computation of flux-based isocontours to represent streamlines can be achieved efficiently in two dimensions and for incompressible flows. Since generation of isocontours in parallel is well optimized, in this work we take up this approach to generate streamlines for large scale two-dimensional flows. Apart from being load balanced, this method provides faster data exploration through the use of large number of streamlines. Also, space usage is reduced since the original vector field can be discarded benefiting the storage and processing of large data sets.

In our work, we adopt a novel divide and conquer strategy to analyze large two-dimensional data sets. We formulate a parallel streamline generation method that uses the load balanced parallel data computation model of the existing flow simulations. Given a data set, initially it is divided into multiple data blocks and flux-based scalar fields are computed in these blocks in parallel. The isocontours of these local flux-fields represent the streamlines in those local blocks. The local data is then globally synchronized by propagating the flux offset values that allows for stitching of the shorter streamline segments. Finally, the complete streamlines are generated in parallel by calculating isocontours of the data blocks in parallel. Although this method is currently designed to work on multi-processor environment, any GPU based architecture can also benefit from this algorithm.

Our contributions in this work are multi-fold:

1. For analyzing large scale two-dimensional flow simulation data, we use a flux-based approach to compute the local streamlines of a region and assign flux ids to the streamlines to store them in the form of a scalar field.
2. We propose a novel algorithm to stitch the different segments of a streamline seamlessly across local regions to produce a longer streamline.
3. Our streamline generation algorithm provides an efficient multi-processor work-flow to achieve speed up, scalability and load balancing.

2. Related Works

Integral curves are very popular for analyzing vector data and it has been widely used in the past. A comprehensive discussion on the basics of streamlines and related issues can be found in the survey by McLoughlin *et al.* [MLP*10]. Due to the complexity of particle tracing and its data dependency, how to compute streamlines in parallel has been an important research topic. To compute streamlines on very large datasets in a distributed setting, Camp *et al.* [CGC*11] study the benefits of parallelize-over-seeds and parallelize-over-blocks, and propose a hybrid approach. Peterka *et al.* [PR11] study data partitioning and job assignment for the parallelize-over-blocks scenario and find that the Round-Robin partitioning generally achieves better load balance. Nouanesengsy *et al.* [NLS11] use a preprocessed graph representation of the flow field to optimize block assignment to the processes. Muller *et al.* [MCHG13] take the dynamic job assignment approach to balance the workloads by a work requesting algorithm for the parallelize-over-seeds scheme. Kendall *et al.* [KWA*11] propose a MapReduce-like framework called DStep to achieve high scalability. Recently, Agranovsky *et al.* [ACG*14] provided an interpolation-based integral curve generation method for performance and accuracy improvement.

Although the use of numerical integration is popular for generating integral curves, it can be problematic depending on multiple factors [KM92]: method of integration, selected step size, interpolation scheme etc. Implicit integral curves and surfaces are an alternative to numerical integration based methods which mostly stem from the idea of stream functions that are popular for two-dimensional incompressible flows as well as three dimensional flows [Gie51,Yih57,Gre93,Kel96,ER92,Mat93,SF90,SH88] since a long time. Some notable works in this topic were provided by Kenwright and Mallinson [KM92, KM96], Beale [Bea97], Van Wijk [vW93]. A closely related concept is the analysis of vector field using the popular Helmotz-Hodge decomposition [PP03, TLHD03]. Although widely popular, implicit streamline generation using these methods was mostly limited to serial sequential algorithms. In this work, we take up the idea of implicit streamlines and extend it to work efficiently in parallel environment.

Use of short streamlines are useful for flow visualization and one of the most popular methods that uses short streamlines was presented by Cabral and Leedom [CL93] in their Line Integral Convolution (LIC) work. Recently these streamlets [MLP*10] were used to visualize vortices formed near insects at the time of flight. Hlawatsch et al. [HSW11] address the issue of redundant calculation during dense streamline or pathline integrations such as LIC and FTLE computation, and propose to concatenate short traces pre-generated for each spatial or temporal regions of a vector field. Since the pre-computed traces cannot cover all possible seed positions, interpolation of the stored traces is used, which unavoidably induces errors. To reduce the errors while keeping storage small, particle traces are computed in hierarchical lengths. Bhatia *et al.* [BJB*11] presented a new data structure called Edge-maps to move away from the issues incurred in numerical integration based streamlines. Recently, another interpolation based streamline construction method was provided by Agranovsky *et al.* [AOGJ15] where the authors generated stremlines at various level-of-

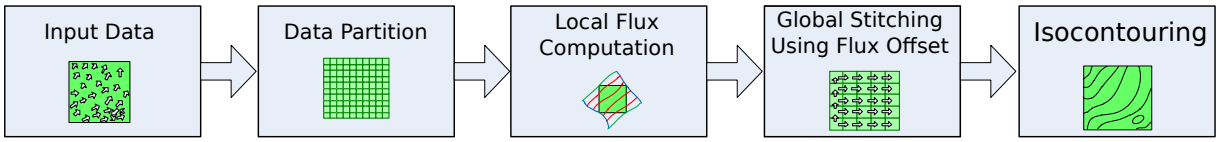


Figure 1: A schematic representation of our system pipeline.

detail with varying degrees of accuracy. Compared to these techniques, our method uses flux values to identify streamlines and based on this idea, we provide a complete parallel framework that facilitates load-balanced and scalable streamline extraction.

3. System Overview

In this section, we briefly discuss the different stages of our pipeline. The goal of this work is to generate streamlines in a distributed manner. We want to generate the different segments of a given streamline independently and stitch them seamlessly over the different blocks. To achieve this, we leverage the flux invariance property of incompressible flows. Given the data set, first the data set is subdivided into different blocks. After this data partition stage, a local computation is applied on each individual block. In this local computation stage, the vector data is converted to a scalar field such that the isocontours of the resulting scalar field represent the streamlines. To achieve this, the flux values of the local block are computed to implicitly store the local streamlines. After the vector field of local blocks are converted to scalar flux field, the flux values are synchronized across the blocks. To achieve this, we exploit the additive property of the flux values and each block sends out its flux offset value to its neighbor. By propagating the flux values across the neighbors, the global flux field is synchronized. To generate the final streamlines, each block can extract the isocontours independently which are stitched across the blocks seamlessly. These isocontours represent streamlines for the input vector data set. A schematic view of our work flow is presented in Figure 1.

4. Method

4.1. Flux-Based Ordering of Streamlines

Given a large two-dimensional vector field, $\vec{V} =< u, v > \in R^2$, initially the data set is decomposed into several blocks to initiate the parallel local streamline generation process. In the local computation stage, the goal is to assign unique local identifiers to the streamlines of each data block without explicitly computing them. Since streamlines do not intersect each other except at the critical points, each point in the spatial domain can be assigned a scalar as the streamline identifier, where points with the same scalar are part of the same streamline. This transformation from the input vector field to a scalar field can effectively be used to implicitly compute the streamlines. Although there are many possible ways of finding unique streamline identifiers, assigning the identifiers based on flux values is more advantageous because, a) no explicit streamline computation is necessary, and b) these local streamlines can be stitched across the blocks efficiently in the later stages of our pipeline according to the flux conservation property of the flow. Compared to this flux-based method of streamline ordering, other methods such as employed by Van Wijk [vW93], do not allow for efficient and smooth streamline stitching as they are not based on the physical property of the fluid.

In the field of transport phenomena, flux refers to flow per unit area. For a given quantity, this is measured as the amount of transport normal to the infinitesimal surface patch in three-dimensions. In two-dimensional scenario, flux Φ across a two-dimensional curve is defined by the amount of flow that is normal to that curve:

$$\Phi(\vec{C}) = \int_{\vec{C}} \vec{V} \cdot d\vec{c} = \int_{\vec{C}} \vec{V} \cdot (\vec{N}_c * dc) = \int_{\vec{C}} (\vec{V} \cdot \vec{N}_c) dc \quad (1)$$

where the integral is performed along the curve $\vec{C}$ and $\vec{N}_c$ is the unit normal to the segment $d\vec{c}$. A schematic example is shown in Figure 2a where curve C_1C_2 with normal N_C is placed in the vector field V. In this vector field, if C_3C_4 is a streamline then the flux across this curve is 0. The key property we leverage here is that flux is conserved within a control area for steady state incompressible fluid flows with zero divergence, i.e., divergence $div(\vec{V})$ at every point follows this:

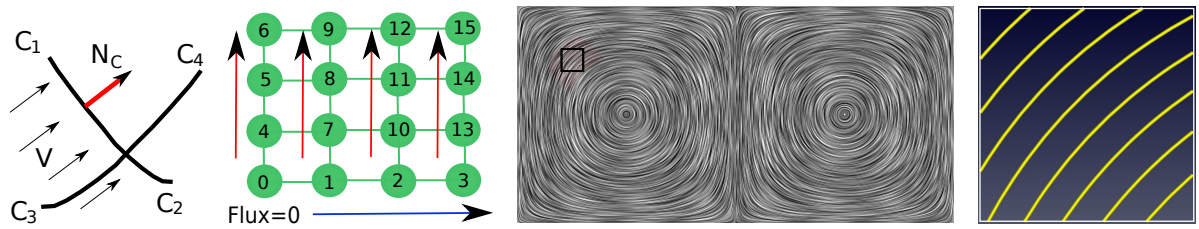
$$div(\vec{V}) \equiv \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0. \quad (2)$$

So, the total flow going into a finite area will be equal to the total flow going out of that area. As a discrete approximation of Equation 1, total flux $Flux_{A_1A_n}$ across a piecewise linear curve A_1A_n where n points on the curve $A_1A_n = \{A_i\}, i = 1...n$, can be computed as :

$$Flux_{A_1A_n} = \sum_{i=1}^{n-1} (((V(A_i) + V(A_{i+1}))/2) \cdot N(A_iA_{i+1})) * d(A_iA_{i+1}) \quad (3)$$

Here, $V(A_i)$ returns the vector of location A_i, $N(A_iA_j)$ denotes the unit normal to the segment A_iA_j whose length is given by $d(A_iA_j)$ and $Flux_{A_iA_j}$ denotes the flux of segment A_iA_j with A_i as the flux origin point. Here we note that, $Flux_{A_iA_j} = 0$ if $i = j$. Instead of the trapezoidal rule presented in Equation 3, any other method (e.g. Simpson's rule) can also be applied to evaluate the integral of Equation 1.

To generate a field of streamline identifiers, we compute



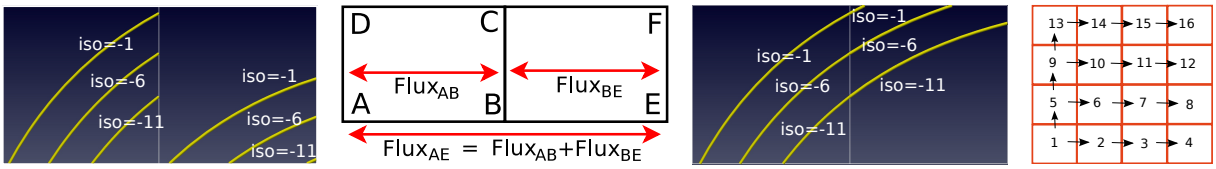
(a) Two-dimensional flux across planar curves.

(b) Local flux computation process.

(c) LIC visualization of the Double Gyre data set.

(d) Isocontours from the flux field.

Figure 2: An illustrative example of flux-based streamline computation.



(a) Isocontours generated in parallel without stitching.

(b) A schematic example of the relationship of the flux values for the neighboring data blocks.

(c) Isocontours generated in parallel after stitching across the blocks.

(d) A simple flux propagation scheme among the data blocks.

Figure 3: Flux based stitching and parallel propagation.

the flux values from a flux origin for every grid point. Assuming an arbitrary flux origin point A_1 within a data block, flux computation can be performed for a grid point A_n by applying Equation 3 along a path A_1A_n that connects the grid point and the local flux origin. Extending this to all the grid points of the block, the local vector field can be converted to the intenteded flux field. In our case, as shown in Figure 2b, we take the left-bottom most point of the rectangular block as the local flux origin and propagate the flux values to the neighboring grid points until all the grid points of the block are assigned flux values given the local flux origin. First the flux propagation is performed along the lower edge (along the blue arrow) and then the flux values are propagated column-wise (along the red arrows). The time stamps shown in Figure 2b represent the order in which the grid points receive the propagated flux values. The isocontours of thus produced flux field yields the streamlines. In Figure 2c, the Double Gyre data set is visualized using the popular Line Integral Convolution (LIC) technique and a 10×10 block is extracted as shown by the black box. The yellow curves of the Figure 2d are the isocontours of the flux field that show a good conformance with the numerical integration based streamlines from the vector field. Next we discuss how this flux-based ordering facilitates effective streamline stitching across blocks.

4.2. Flux Based Stitching of Local Blocks

Using the grid based flux computation approach, it is possible to generate local isocontours as local streamline segments in parallel that are coherent within individual data blocks. But when two neighboring blocks generate two segments of one streamline independently by generating isocontours of a given flux iso-value, the two isocontours (i.e., the two streamline segments) across blocks may not match at the block boundary as shown in Figure 3a. This is because the flux values at each block were computed assuming local flux origins. Figure 3a shows three isocontours from two neighboring blocks where they show a mismatch at the block boundary due to local computation. To achieve a proper stitching of the streamline segments across blocks, communication among the blocks is needed to synchronize all the blocks to a common flux origin.

When an incompressible fluid flow is decomposed into blocks, two neighboring blocks produce flux values assuming local flux origins. Since flux is conserved across the blocks and is independent of the path, for two neighboring blocks to have consistent flux values, their flux origins need to be aligned. This synchronization of flux values can be achieved using the boundary points shared by the neighboring blocks. Following Figure 3b, two neighboring blocks ABCD and BEFC initially compute local flux values with respect to flux origin points A and B respectively. Due to this mismatch of flux origins in the two blocks, given an iso-value, the local isocontour segments of the individual blocks are not globally coherent. Evidently, in general scenario, $Flux_{AB} \neq Flux_{BB}$ which causes the mismatch and the flux values of the block BEFC require a change of origin to be aligned with global flux origin A instead of local origin B. The key property we leverage here is that when flux is conserved in the domain, the flux difference between two points B and C is independent of flux origin. In other words, when

flux difference between B and C is computed assuming flux origin A within block ABCD as $\Delta F_1 = Flux_{AC} - Flux_{AB}$ and the same flux difference is computed assuming flux origin B within block BEFC as $\Delta F_2 = Flux_{BC} - Flux_{BB}$, then :

$$\Delta F_1 = \Delta F_2 \Rightarrow Flux_{AC} - Flux_{AB} = Flux_{BC} - Flux_{BB}, \quad (4)$$

Following this, synchronization of flux values across the blocks is achieved by observing the flux difference between the local and global origins, $Flux_{AB} - Flux_{BB}$ and adding this flux offset to all the flux values of block BEFC. Thus, flux values can be directly communicated whenever two blocks share a common grid point or a common edge. In general, if two blocks $B1$ and $B2$ share a common point i with local flux origins p_1 and p_2 respectively, then change of flux origin from p_2 to p_1 for $B2$ is achieved as:

$$Flux_{p_1 j} = Flux_{p_2 j} + (Flux_{p_1 i} - Flux_{p_2 i}), \forall j \in B2. \quad (5)$$

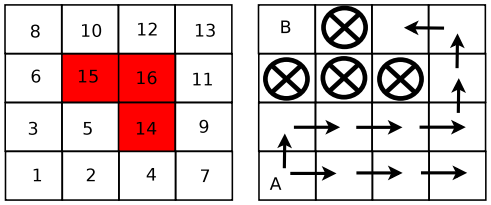
If $B1$ and $B2$ share a common edge, then any point i from that shared edge can be chosen for this change of flux origin following the flux conservation theory. To ensure that the neighboring blocks have a shared edge or a point, we use one layer of ghost cells in the data decomposition phase.

Given multiple data blocks and locally computed flux values, a parallel communication algorithm can be devised to facilitate the flux offset propagation as shown in Figure 3d. We assume that the global origin is contained in the left-bottom most block. Thus, in Figure 3d, block 1 contains the global origin and it sends the required flux offset value to block 2 and block 5 in the first phase of communication. Next, after receiving the offsets, the two blocks adjust their local flux values to align with block 1 and then further send out the flux offsets to their neighbors, block 3, block 6, and block 9. The overall communication takes $O(N^{\frac{1}{2}})$ stages to complete where N is total number of data blocks. Since only a single flux-offset value is to be sent among neighboring blocks, the communication cost is much less compared to seed or data exchange in traditional parallelize-over-block or parallelize-over-seed techniques. After this communication stage, the resulting stitched flux field can be used to generate isocontours that represent streamlines. Since isocontour extraction routines are highly parallel and optimized for performance, the final streamline generation stage is also very efficient. After this stitching, as shown in Figure 3c compared to 3a, now isocontours of the same iso-value can be generated independently in two neighboring blocks that match seamlessly the block boundary.

4.3. Error Handling

4.3.1. Handling Non-zero Divergence Regions

The flux-based parallel streamline generation and stitching algorithm provided above works very well for two-dimensional data sets that have zero divergence. Although incompressibility is a popular assumption in flow simulations, the real data sets can consist of regions that contain non-zero divergence values and the local flux values at these regions contain uncertainty. Since flux computation is additive, this uncertainty can spread from one region to another if a simple flux computation algorithm is employed, e.g., the method shown in Figure 2b. To restrict this uncertainty from spreading throughout the domain, we devise an error-aware flux computation in the local blocks. Here, we use divergence of the field as the error indicator as described by the Equation 2 and we assume an additive error model since flux is also additive. Given a data block, a conceptual graph is created where each node represents a grid point and absolute values of divergence errors are assigned to the nodes as their weights. Undirected edges are present if two nodes are neighbors. Now, a possible flux computation path is found by finding a path starting from the start node that visits all the nodes where the node weights are propagated at each hop. Our goal is to find a path that minimizes the sum of flux errors of all the nodes of this graph after the flux values are computed for all nodes. Instead of finding all possible spanning trees of the graph, we apply a greedy flood fill like algorithm similar to Dijkstra's algorithm for the single-source-shortest-paths problem. The source node is taken as the node with minimum divergence error and we maintain a sorted list that determines the next node to be visited. The neighbors of the current explored node are put into this list sorted according to their divergence error values. This greedy scheme generates good quality results with a small performance overhead.



(a) A schematic example of handling regions with non-zero divergence values. Red regions indicate larger divergence regions.

(b) A schematic example of handling regions with discontinuities. The marked-circular glyphs indicate cluster of discontinuities.

Figure 4: Handling of erroneous regions while propagating flux.

A schematic example of this is provided in Figure 4a where a 4×4 block is shown and red marked regions are non-zero divergence regions. Instead of applying a naive algorithm, our error-aware algorithm defers the exploration of the erroneous regions which in turn restricts the error propagation. The numbers show the timestamps of regions as they were explored. Global origin point is now selected to be the grid point within the left-bottom most block (block 1) that has the least error (or the divergence at this point is closest to zero).

4.3.2. Handling Discontinuities and Critical Points

Critical points are important features of the flow data sets and their existence helps define the topology of the data. Invariably, most of the interesting flow data sets generally contain critical points. Although less prevalent, discontinuities can also be observed in the data sets. Examples of this can be an object inside the flow (flow around a cylinder data sets) or presence of land in the ocean (Parallel Ocean Program data sets). Using our method, isolated critical points are automatically handled as will be shown in the Results section. For clusters of discontinuities, the parallel algorithm needs to be aware of the fact that flux computation can not propagate through those regions, instead it needs to go around the discontinuities as shown in Figure 4b. Conceptually, this is similar to a flood fill algorithm starting from region A. Compared to the previously mentioned non-divergence regions, in this scenario, we only need to ignore the processing of the discontinuous regions. If there is a region that is surrounded by discontinuities, for example region B in Figure 4b, then this region is not synchronized with other regions but processed independently since no streamlines can leave this block.

4.4. Quality Analysis and Quantification

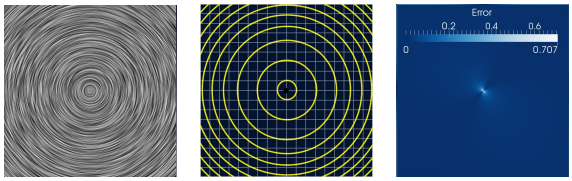
The method described in the previous sections is calculating a global field based on the local properties of the data set. Since the final outputs from the proposed method are streamlines which are essentially isocontours of the stitched flux field, it is essential to provide the uncertainty factors in the analysis. The first important source of uncertainty for our case is the local numerical integration for computing the flux values for the blocks. Secondly, the other source of uncertainty is the interpolation scheme for generating isocontours to show the streamlines. Higher order integration and interpolation techniques can be applied to reduce these two types of uncertainty without introducing much performance overhead. Since the particle-tracing based streamlines are also erroneous due to multiple factors [KM92], while checking the quality of our method, we do not directly compare with the streamlines; instead here we show point-wise error of the generated scalar field and evaluate the quality. To achieve this, we construct a vector field from the scalar flux field F by taking the local derivative at every grid point and check if it is everywhere normal to the origin vector field $\vec{V}$. Formally, our error measure is given as:

$$Err(F,\vec{V}) = (\nabla F)\cdot\vec{V} \tag{6}$$

After generating this error field, we can visualize it to understand the error in our method. For a perfect reconstruction, the error should be 0 and the values closer to 1 will report higher amount of error.

5. Results

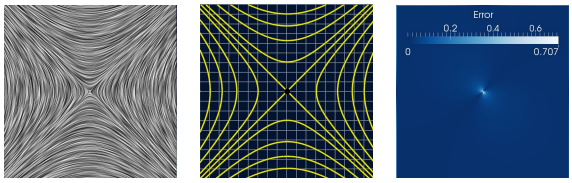
Our experiments and scaling study were conducted on the Blue Gene/Q Vesta system at the Argonne Leadership Computing Facility. Vesta consists of 2,048 nodes where each node is equipped with 16 PowerPc A2 1600MHz cores and 16 GB of RAM. The total memory of Vesta is 32TB and it has a General Parallel File System. Next we present the visualization results and the correctness of the generated streamlines to show the efficacy of our proposed method at varying block sizes.



(a) LIC visualization of Circle data set.

(b) Stitched streamlines from the Circle data set.

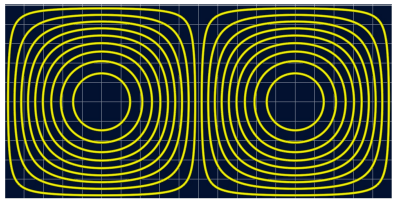
(c) Error visualization of the streamlines from Circle data set.



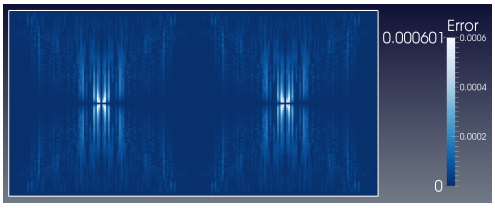
(d) LIC visualization of Saddle data set.

(e) Stitched streamlines from the Saddle data set.

(f) Error visualization of the streamlines from Saddle data set.



(g) Stitched streamlines from the Double gyre data set.

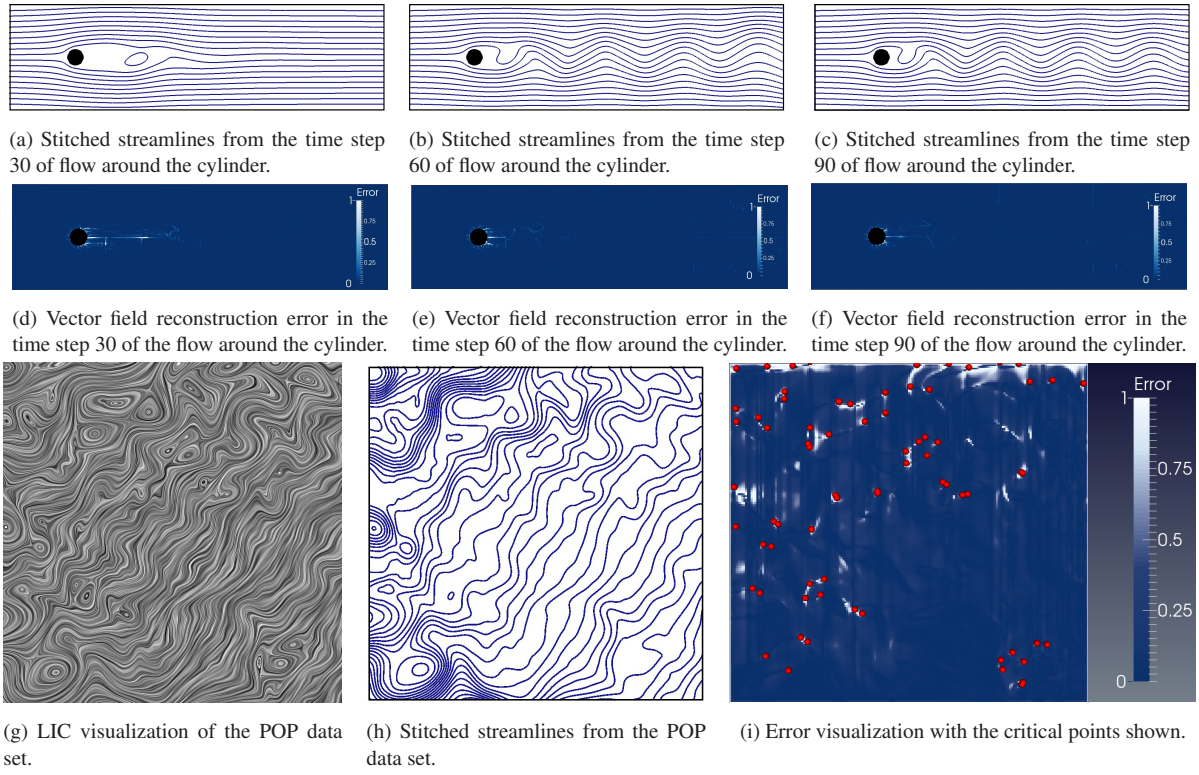


(h) Error of Double gyre data set.

Figure 5: Stitched Streamlines from different data sets and their respective error visualizations.

5.1. Circle and Saddle Data Sets

First we apply our algorithm on two simple example data sets to demonstrate its effectiveness. These two data sets are generated using the code distributed by Turk and Banks [TB96]. Both the data sets were set to a resolution of 6400×6400. The first data set Circle describes a circular



(a) Stitched streamlines from the time step 30 of flow around the cylinder.

(b) Stitched streamlines from the time step 60 of flow around the cylinder.

(c) Stitched streamlines from the time step 90 of flow around the cylinder.

(d) Vector field reconstruction error in the time step 30 of the flow around the cylinder.

(e) Vector field reconstruction error in the time step 60 of the flow around the cylinder.

(f) Vector field reconstruction error in the time step 90 of the flow around the cylinder.

(g) LIC visualization of the POP data set.

(h) Stitched streamlines from the POP data set.

(i) Error visualization with the critical points shown.

Figure 6: Results for different time steps for flow around a cylinder data set and the POP data set.

flow. LIC visualization of the Circle data set is presented in Figure 5a. The other data set, Saddle, describes a two dimensional saddle structure and it is presented in Figure 5d. Both the data sets contain critical points at the center. Initially, the two data sets are divided into multiple small data blocks and a flux field is generated using local flux based computation. A very similar result can be obtained even if the block size is varied. The local data blocks are then stitched across the blocks through the propagation of flux values and the results are presented in Figure 5. Figure 5b represents the stitching results from the Circle data set where yellow curves are the isocontours in the stitched field that correspond to the streamlines of the field. The data subdivision is shown as white grids in the background of the figures. The quality of the proposed method on this data set is shown in Figure 5c where the error is quantified according to the measure described in Section 4.4. Similarly, for the Saddle data set, the stitched streamlines are depicted in Figure 5e.The corresponding error visualization is shown in Figure 5f. In both the error visualizations, it is seen that most of the data set has very low error and the highest error occurs near the center critical point of the data sets. From these results, it can be concluded that our proposed methods work quite well for these data sets.

5.2. Double Gyre

Next we show our result on the analytic data set Double Gyre which contains two symmetric vortices. This data set was generated using the following equations:

$$\begin{aligned} u(x,y) &= -\pi \sin(\pi x)\cos(\pi y) \\ v(x,y) &= \pi \cos(\pi x)\sin(\pi y) \end{aligned} \tag{7}$$

over the region $[0,2] \times [0,1]$. We test on the data set that has a resolution of 2001×1001. To generate the results, we first divide the whole data set into contiguous 101×101 blocks and compute the local flux fields for each block independently. Then starting from the left-bottom corner-most block, the flux values are propagated similar to what is shown in Figure 3d. After the stitching process, the isocontours are computed. In Figure 5g, we show the isocontours computed from the stitched flux field of this data set and the data partitioning is shown in the background of the figure. From this figure, it can be observed that the maximum error is very small which reflects a good reconstruction of the vector field.

5.3. Flow Around a Cylinder

Next we show the results on the data set that describes a two-dimensional flow around a cylinder for multiple time steps. This data set represents the karman vortex streets in the fluid flow. From this data set, we have chosen three time-

steps at equal intervals: time step 30, 60 and 90 and applied our method. Each time step has a resolution of 768×231. For computation of the local streamlines within the blocks, the block size was chosen to be 11×11. Computation of divergence field on this data set yields very small values that suggests that this data set can be used as incompressible flow without much error. This data set contains discontinuities where the cylinder is placed inside the flow as shown by the black color in Figure 6. Figure 6a represents the reconstructed isocontours after the stitching across the blocks. The effectiveness of the reconstruction is shown in Figure 6d. As can be seen in this figure, the flux field is in close agreement with the original vector field. The higher error regions are near critical points and regions of discontinuity. Further, Figure 6b and Figure 6c represent the isocontours from the time steps 60 and 90 with corresponding error visualizations provided in Figure 6e and Figure 6f. In these figures, we again see that our local computation and stitching process was able to generate streamlines with very low amount of error.

5.4. Parallel Ocean Program

Here we present the output of Los Alamos National Laboratory's Parallel Ocean Program (POP) model as our next data set and show the results. The full data set is three dimensional with a resolution of $3600 \times 2400 \times 42$ but scientists generally use this data set slice by slice to explore the characteristics of the different regions of sea. The velocity in the z direction is often ignored for general analysis and we have followed the same approach to produce our results. From the data set, we have extracted a region near the land of the South American continent with a resolution of 200×200 and applied our method on it. A LIC visualization of this region is provided in Figure 6g. From this figure we can see that the flow in the selected region is quite complex and contains many critical points with high curvature regions. Now we divide the data into 11×11 chunks and apply our algorithm to generate the flux field with stitching. The isocontours of this stitched flux field is presented in Figure 6h. By visual analysis, we can readily observe that the streamlines conform with the flow directions shown in the LIC image. To estimate the quality, we compute the gradient of the flux field and find the dot product with the original vectors and present in Figure 6i. Also, the critical points are highlighted as the red points in this image to show the correlation of error and critical points of the field. As we can observe, although the quality overall is quite good, near the critical points and high curvature regions, we get higher error. We note that due to the non-negligible divergence values in some regions of this data set, the flux differences across the blocks had some mismatch which was compensated by adding an extra flow amount in the block. Despite this flux mismatch, as can be seen from the resulting figures, the reconstruction has high accuracy in other regions and generated streamlines are smooth and represent the flow features very well.

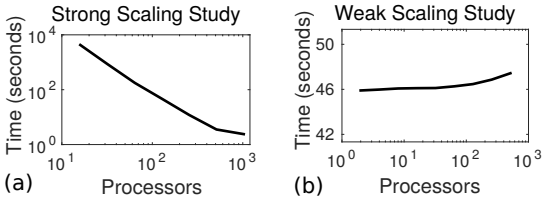


Figure 7: Scaling study. (a) Strong scaling study performed on upsampled circle data set. (b) Weak scaling study performed on various upsampled circle data sets.

6. Discussions

6.1. Scaling and Performance Study

To understand the scalability of our proposed system, experiments were conducted on multi-core machines and these results are presented in Figure 7. For the strong scaling experiment, an upsampled version of the Circle data set was used whose dimensions were 6400×6400 for the vector field. Number of processors were increased upto 1024 and total running time (total running time = local flux computation + communication + isocontour extraction) was recorded as shown in 7a. This log-log scale plot shows that our method scales well with the increase in number of processors. Figure 7b represents weak scaling study where again the Circle data set was used. For generating the plots, data resolution and processor number were doubled starting from data size 400×400 with 2 processors upto data size 6400×6400 for 512 processors. In this weak scaling study, again a good scaling can be seen.

Next we present the comparison of our flux based streamline generation method with a numerical integration based approach. For this comparison, we have used the popular OSUFlow library that implements Peterka et al.'s proposed method [PR11] of parallelizing streamline computation by distributing the data blocks across processors. This method is chosen for comparison instead of parallel-over-seeds since the execution model of large ocean simulations is also parallel-over-blocks. For integration based streamline generation, a fixed step size of one voxel length was used along with RK4 integration method. In our proposed method, the VTK [SAH00] library was used to generate the final isocontours. We have used 128 processors and varied the generated streamlines from 10^3 to 10^5 in an upsampled version of flow around a cylinder data set (with effective data size 4800×2000) and recorded the total running times in Figure 8. As can be seen from this figure, our method takes much less time than OSUFlow when more streamlines are traced. Since the execution speed of our flux-based method only depends on the data size rather than the complexity of vector data, we expect to see similar speed-up trends and scalability reports for other data sets of similar sizes. As can be seen from the scalability and performance results, our method is well suited for the use cases where a large number of streamlines are to be generated.

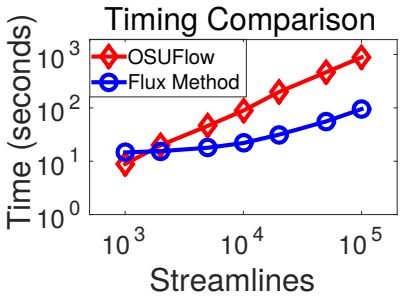


Figure 8: A comparison of performance between our flux based method and OSUFlow generated streamlines.

6.2. Consistency

While generating long streamlines, generally numerical integration based approaches become less reliable due to the accumulation of local integration errors at each step. Reliability of numerical integrations vary depending on the integration method, step size and interpolation method used as discussed by several researchers [BJB*11, Bun88, KM92, MP88, YP88]. Stream function based approaches generally provide a more reliable solution compared to the numerical integration based techniques. Apart from having a very low error in the reconstructed vector fields, one specific example can be shown to depict the consistency of our proposed method. As described by Perry *et al.* [PCL82], in two dimensional incompressible flow, the critical points can only be of two types: center and saddle points. Now, if we start to trace a particle using Runge-Kutta 4th order (RK4) method in the flow around a cylinder data near a critical point (as shown in Figure 9a), the ideal behavior should be to get a close loop streamline. Instead, as depicted in Figure 9b, we see that the streamline is spiraling and tending outwards from the critical point. This behavior should be observed near a spiraling source but as we mentioned before, this data set does not contain a source/sink. It is observed that, compared to Figure 9b, our method can produce more consistent results as depicted in Figure 9c.

6.3. Limitations and Future Work

As shown in the Results section, the proposed approach performs well for two-dimensional data sets that can be approximated as incompressible flows. In this section, we discuss about the possible extensions of our work for three-dimensional fluids and compressible fluid flows. As mentioned before, the incompressibility assumption is quite popular in the CFD community and computation of divergence is an indicator to the incompressibility of the generated data set. When the divergence is relatively low or close to zero for a data set, our proposed method performs very well. If divergence is higher everywhere in the data set, e.g. for a compressible fluid flow simulation, then the errors can become non-negligible. To formally propose a solution for compressible flows, the density can be considered in the mass flow

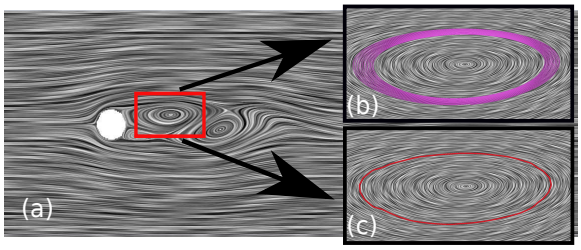


Figure 9: A comparison of RK4 method and our flux based method. (a) LIC visualization of time step 30 of flow around a cylinder data where a region near a critical point is selected. (b) Streamline generated using the traditional RK4 integration incorrectly suggests the existence of a spiraling source. (c) Our flux based approach correctly highlights the existence of a center with a closed loop streamline.

computation. Following this approach, the data sets will require the density field along with the velocity fields for this method to work and this is one of our future extensions to this current algorithm. We note that, if the flow does not have a z-component or if z-component can be ignored like the POP data set, then even three-dimensional data sets can also be handled by applying our algorithm on each slice. For extending to general three-dimensional data sets, our algorithm needs to be modified such that it generates a three-dimensional flux field. The motivations for this can be taken from three-dimensional stream function computations which are computationally involved [Gre93, HD86, Mat93] but are possible to compute with some constraints and assumptions. Apart from extending our work to three-dimensional compressible flows, we also plan to extend our work for uncertain data sets as part of our future work.

7. Conclusions

In this work, we proposed a novel technique for generating streamlines in parallel with high scalability for large two-dimensional flow fields. Instead of applying the traditional numerical integration schemes on the vector fields, we propose to compute a flux-based scalar field for implicit streamlines in parallel. Initially, data is subdivided into smaller blocks and local flux field is generated whose isocontours represent local streamlines. Since each block computes flux values according to a local flux origin, alignment of flux origins is needed to generate coherent streamlines across blocks. To achieve this, we use the additive property of flux and each block sends its local offset to its neighbors. Since this communication stage involves sending one float value across the neighbors of each block, it is efficiently performed and a stitched flux field is generated. Finally, isocontours of this stitched field is extracted in parallel to produce the final streamlines. The results obtained from a variety of data sets, both analytical and simulated, show the effectiveness of the stitching method. Our experiments also reveal good strong

and weak scaling properties as well as much improved execution time.

8. Acknowledgments

This work was supported in part by NSF grants IIS-1250752, program manager Almadena Chtchelkanova, IIS-1065025, and US Department of Energy grants DE-SC0007444, DE-DC0012495, program manager Lucy Nowell.

References

[ACG*14] AGRANOVSKY A., CAMP D., GARTH C., BETHEL E., JOY K., CHILDS H.: Improved post hoc flow analysis via lagrangian representations. In *Large Data Analysis and Visualization* (Nov 2014), pp. 67–75. 2

[AOGJ15] AGRANOVSKY A., OBERMAIER H., GARTH C., JOY K. I.: A multi-resolution interpolation scheme for pathline based lagrangian flow representations. *Proc. SPIE 9397* (2015). 2

[Bea97] BEALE S. B.: Visualisation of three-dimensional flow fields using two stream functions. *10th International Symposium on Transport Phenomena* (1997). 2

[BJB*11] BHATIA H., JADHAV S., BREMER P.-T., CHEN G., LEVINE J., NONATO L., PASCUCCI V.: Edge maps: Representing flow with bounded error. In *Pacific Visualization Symposium, IEEE* (March 2011), pp. 75–82. 2, 9

[Bun88] BUNING P.: Sources of error in the graphical analysis of cfd results. *J. Sci. Comp 3* (1988). 9

[CGC*11] CAMP D., GARTH C., CHILDS H., PUGMIRE D., JOY K. I.: Streamline integration using mpi-hybrid parallelism on a large multicore architecture. *IEEE Trans. Vis. Comput. Graph. 17*, 11 (2011), 1702–1713. 2

[CL93] CABRAL B., LEEDOM L. C.: Imaging vector fields using line integral convolution. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (1993), ACM, pp. 263–270. 2

[ER92] EVANS D. G., RAFFENSPERGER J. P.: On the stream function for variable-density groundwater flow. *Water Resources Research 28*, 8 (1992), 2141–2145. 2

[Gie51] GIESE J.: Stream functions for three-dimensional flows. *J. Math. Phys. Vol: 30, No. 1* (Apr 1951). 2

[Gre93] GREYWALL M. S.: Streamwise computation of three-dimensional flows using two stream functions. *Journal of Fluids Engineering 115* (1993), 233–238. 2, 9

[HD86] HUANG C. Y., DULIKRAVICH G. S.: Stream function and stream-function-coordinate (sfc) formulation for inviscid flow field calculations. *Comput. Methods Appl. Mech. Eng. 59*, 2 (nov 1986), 155–177. 9

[HSW11] HLAWATSCH M., SADLO F., WEISKOPF D.: Hierarchical line integration. *IEEE transactions on visualization and computer graphics 17*, 8 (Aug. 2011), 1148–63. 2

[Kel96] KELLER J.: A pair of stream functions for three-dimensional vortex flows. *Zeitschrift fur angewandte Mathematik und Physik ZAMP 47*, 6 (1996), 821–836. 2

[KM92] KENWRIGHT D., MALLINSON G.: A 3-d streamline tracking algorithm using dual stream functions. In *Visualization, 1992. Visualization '92, Proceedings., IEEE Conference on* (Oct 1992), pp. 62–68. 2, 6, 9

[KM96] KNIGHT D., MALLINSON G.: Visualizing unstructured flow data using dual stream functions. *Visualization and Comp. Graphics, IEEE Transactions on 2*, 4 (Dec 1996), 355–363. 2

[KWA*11] KENDALL W., WANG J., ALLEN M., PETERKA T., HUANG J., ERICKSON D.: Simplified parallel domain traversal. *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '11* (2011), 1. 2

[Mat93] MATANGA G. B.: Stream functions in three-dimensional groundwater flow. *Water Resources Research 29*, 9 (1993), 3125–3133. 2, 9

[MCHG13] MULLER C., CAMP D., HENTSCHEL B., GARTH C.: Distributed parallel particle advection using work requesting. In *IEEE Symposium on Large Data Analysis and Visualization 2013, LDAV 2013 - Proceedings* (2013), pp. 1–6. 2

[MLP*10] MCLOUGHLIN T., LARAMEE R. S., PEIKERT R., POST F. H., CHEN M.: Over two decades of integration-based, geometric flow visualization. *Computer Graphics Forum 29*, 6 (2010), 1807–1829. 2

[MP88] MURMAN E., POWELL K.: Trajectory integration in vortical flows. *AIAA Journa 27* (1988), 982–984. 9

[NLS11] NOUANESENGSY B., LEE T.-Y., SHEN H.-W.: Load-balanced parallel streamline generation on large scale vector fields. *IEEE transactions on visualization and computer graphics 17*, 12 (Dec. 2011), 1785–94. 2

[PCL82] PERRY A. E., CHONG M. S., LIM T. T.: The vortex-shedding process behind two-dimensional bluff bodies. *Journal of Fluid Mechanics 116* (3 1982), 77–90. 9

[PP03] POLTHIER K., PREUSS E.: *Visualization and Mathematics III*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, ch. Identifying Vector Field Singularities Using a Discrete Hodge Decomposition, pp. 113–134. 2

[PR11] PETERKA T., ROSS R.: A study of parallel particle tracing for steady-state and time-varying flow fields. *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International* (2011), 580–591. 2, 8

[SAH00] SCHROEDER W. J., AVILA L. S., HOFFMAN W.: Visualizing with vtk: A tutorial. *IEEE Comput. Graph. Appl. 20*, 5 (Sept. 2000), 20–27. 8

[SF90] SENGER R. K., FOGG G. E.: Stream functions and equivalent freshwater heads for modeling regional flow of variable-density groundwater: Application and implications for modeling strategy. *Water Resources Research 26*, 9 (1990), 2097–2106. 2

[SH88] SHERIF A., HAFEZ M.: Computation of three-dimensional transonic flows using two stream functions. *International Journal for Numerical Methods in Fluids 8*, 1 (1988), 17–29. 2

[TB96] TURK G., BANKS D.: Image-guided streamline placement, 1996. http://www.cc.gatech.edu/∼turk/streamlines/streamlines.html. 6

[TLHD03] TONG Y., LOMBEYDA S., HIRANI A. N., DESBRUN M.: Discrete multiscale vector field decomposition. In *ACM SIGGRAPH 2003 Papers* (New York, NY, USA, 2003), SIGGRAPH '03, ACM, pp. 445–452. 2

[vW93] VAN WIJK J. J.: Implicit stream surfaces. In *Proceedings of the 4th Conference on Visualization '93* (Washington, DC, USA, 1993), VIS '93, IEEE Computer Society, pp. 245–252. 2, 3

[Yih57] YIH C. S.: Stream functions in three-dimensional flows. *La Houille Blanche*, 3 (1957), 445–450. 2

[YP88] YEUNG P. K., POPE S.: An algorithm for tracking fluid particles in numerical simulations of homogeneous turbulence. *J. Comp. Physics. 79* (1988), 373. 9