# ON THE PERFORMANCE OF LINE INTEGRAL CONVOLUTION IN A DISTRIBUTED-MEMORY PARALLEL SETTING

by

GARRETT MORRISON

A THESIS

Presented to the Department of Computer and Information Science
and the Graduate School of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Master of Science

June 2018

THESIS APPROVAL PAGE

Student: Garrett Morrison

Title: On the Performance of Line Integral Convolution in a Distributed-Memory Parallel Setting

This thesis has been accepted and approved in partial fulfillment of the requirements for the Master of Science degree in the Department of Computer and Information Science by:

| | |
|---|---|
| Hank Childs | Chair |

and

| | |
|---|---|
| Sara D. Hodges | Interim Vice Provost and Dean of the Graduate School |

Original approval signatures are on file with the University of Oregon Graduate School.

Degree awarded June 2018

THESIS ABSTRACT

Garrett Morrison

Master of Science

Department of Computer and Information Science

June 2018

Title: On the Performance of Line Integral Convolution in a Distributed-Memory Parallel Setting

Line integral convolution (LIC) is a powerful tool for visualizing vector fields by combining particle advection with image convolution. Practical usage of LIC is limited by its computational expense, requiring many calculations for every cell in the mesh. Fortunately, computation of LIC can be accelerated through parallelization. In this thesis we evaluate whether LIC parallelizes better over distributed systems than comparable particle advection algorithms. We do this by harnessing the VisIt Parallel Integral Curve System for the generation of LIC convolution kernels. We also contribute an extension to LIC which reduces dependency on input data. We look at how the algorithm compares to other advection techniques with respect to performance and load balancing. We evaluate the performance of LIC with PICS across 36 different test configurations with three metrics. We find a 2x performance improvement and an up to 8x load balancing improvement for LIC over traditional parallel streamlines.

CURRICULUM VITAE

NAME OF AUTHOR: Garrett Morrison

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon, Eugene, OR, USA
Central Oregon Community College, Bend, OR, USA

DEGREES AWARDED:

Master of Science, 2018, University of Oregon
Bachelor of Science, 2016, University of Oregon
Associate of Applied Science, 2013, Central Oregon Community College

AREAS OF SPECIAL INTEREST:

Scientific Visualization
High-Performance Computing

TABLE OF CONTENTS

Chapter Page

I. INTRODUCTION . . . . . 1

II. RELATED WORK . . . . . 5

2.1 Line Integral Convolution . . . . . 5

2.2 Fast LIC . . . . . 5

2.3 Parallel Particle Advection . . . . . 5

2.4 Parallel LIC . . . . . 7

III. OVERVIEW . . . . . 8

3.1 PICS . . . . . 8

3.2 Algorithm Overview . . . . . 9

3.3 Procedural Noise Generation . . . . . 10

IV. EXPERIMENT OVERVIEW . . . . . 13

4.1 Data Sets and Resolutions . . . . . 13

4.2 Workloads . . . . . 14

4.2.1 LIC . . . . . 14

4.2.2 Streamlines . . . . . 15

4.3 Machines . . . . . 16

4.4 Timing Methodology . . . . . 17

4.5 Step Count . . . . . 17

V. RESULTS . . . . . 19

5.1 Maximum Node Time . . . . . 19

Chapter Page

5.2 Percent Imbalance . . . . . 21

5.3 Average Communication Load . . . . . 24

VI. CONCLUSION . . . . . 27

APPENDIX: TABLES . . . . . 28

REFERENCES CITED . . . . . 31

LIST OF FIGURES

Figure Page

1. Example of LIC applied to a magnetic field. . . . . 2

2. LIC output on complex field. . . . . 3

3. Example of LIC at three resolutions. . . . . 15

LIST OF TABLES

Table Page

A.1. Maximum node advection time on 100x100 field resolution. . . . . 28

A.2. Maximum node advection time on 500x500 field resolution. . . . . 28

A.3. Maximum node advection time on 1000x1000 field resolution. . . . . 28

A.4. Node percent imbalance measures on 100x100 field resolution. . . . . 29

A.5. Node percent imbalance measures on 500x500 field resolution. . . . . 29

A.6. Node percent imbalance measures for 1000x1000 field resolution. . . . 29

A.7. Mean node communication for 100x100 field resolution. . . . . 30

A.8. Mean node communication for 500x500 field resolution. . . . . 30

A.9. Mean node communication for 1000x1000 field resolution. . . . . 30

# CHAPTER I

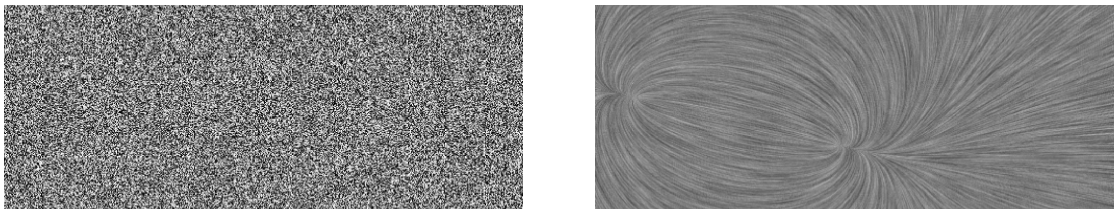
# INTRODUCTION

Vector field data is commonly found in scientific data sets, either from simulated or experimental sources. Examples where vector field data is important include fluid dynamics, astrophysics, nuclear reactors, climate modeling, and ocean modeling amongst others. This motivates the need for flow visualization techniques that enable exploration of these fields.

Particle advection refers to the process of seeding massless particles into a vector field and displacing them in discrete steps based on nearby vector values to build up integral curve approximations. This process is the foundation for many types of flow visualization.

Line integral convolution is a method for visualizing vector field data. It merges image convolution with particle advection to construct representations of field flow. There are two inputs to the algorithm: a vector field and an input image. LIC can be used in applications ranging from scientific visualization to image processing; for scientific visualization, the image is usually some variation of white noise. As originally described by Cabral and Leedom (1993), the process of line integral convolution can be split into two main phases. In the first phase, integral curve segments are generated via advection, with seed points placed at the center of each mesh cell. The second phase identifies cells through which advected seeds have passed and generates an output image. These cells serve as the elements of an image convolution kernel. A mapping is performed between cells and input image pixels. Each output pixel value is the average of pixel values for cells touched by its seed during advection. The final result is a blurring of the input image along

the field flow akin to dropping dye in moving water and observing its movement (see Figures 1 and 2).



(a) white noise input (b) convoluted output

*Figure 1.* Example of line integral convolution applied to a simulated magnetic field. The algorithm converts white noise into a representation of the field lines.

The process of LIC advection is computationally expensive. By default, the output pixels of LIC map one-to-one with the input mesh cells. Therefore, it is necessary to perform a convolution for every cell in the mesh. Each of these cell convolutions requires advecting a number of steps that, as noted by Cabral and Leedom (1993), is double the maximum advection distance. This doubling is due to symmetry requirements, which necessitate that we also advect particles backwards against the direction of the vector field. As a result, serial implementations are, not surprisingly, slow.

While serial LIC is slow, the algorithm exhibits opportunities for parallelization of both the advection and convolution phases. For this thesis we focus on the parallelization of integral curve generation. As previously mentioned, each output pixel depends on a forward and backwards advected integral curve for convolution. An advected particle depends solely on neighboring vectors and not on other particles. This particle independence allows for distributing advection workload across nodes without concern for synchronization.

Parallelization provides a method for scaling LIC to very large distributed systems. However, this technique is limited by the potential for load imbalance.

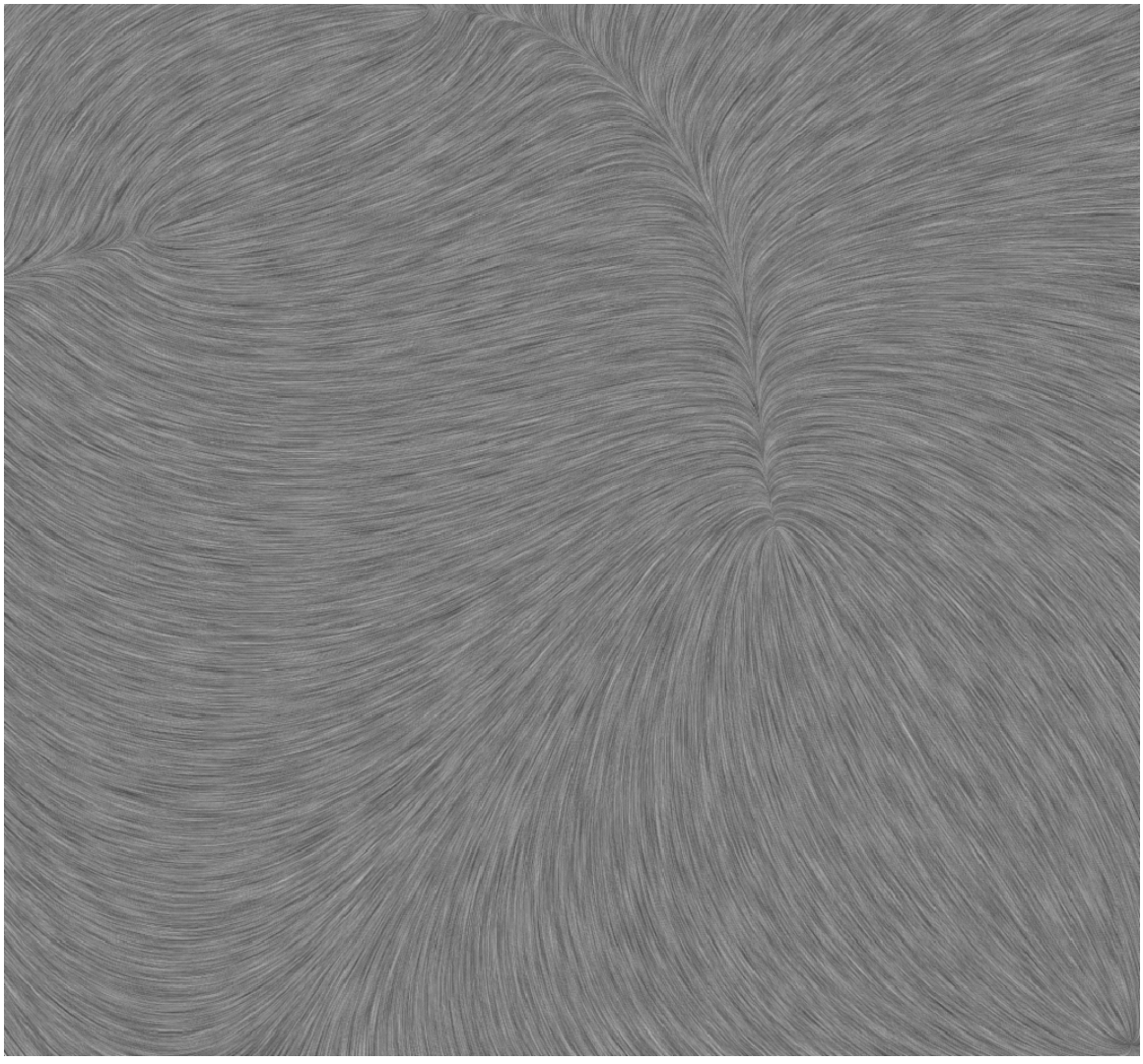


*Figure 2.* Example of LIC visualization for a complex field with multiple sources and sinks.

In particular, for an arbitrary vector field, there is no guarantee that evenly distributed particles will continue to be evenly distributed. It is more likely that seed particles will diverge or converge over time and particle movement is unlikely to be uniform across the field. For distributed-memory processing, this can result in uneven workload over nodes. Movement of a particle from one node to another also incurs network overhead. Both of these issues are the result of the iterative nature of particle advection. A single step is unlikely to cause a particle to transfer between nodes. It is the calculation of multiple advection steps that a particle likely to exit its current node. Therefore, the likelihood that our parallel advection becomes unbalanced is directly proportional to the average number of advection steps taken by the seed particles. This becomes important as the maximum step count of LIC is often much shorter than for other advection techniques. It is this property that serves to motivate our research.

With this thesis, we study the performance properties of distributed-memory parallel line integral convolution. We focus on the method involving work distribution over data and how various vector fields impact the performance of this parallel LIC implementation. We measure the ways in which line integral convolution influences load balancing and network overhead. We also introduce an algorithmic contribution for reducing the memory and communications overhead caused by the image-dependent nature of LIC. The primary research goal of this thesis is to understand parallel LIC performance. We do this by comparing a parallel version of LIC to a similar parallel integral curve method (streamlines), and exploring how the structure of LIC might allow for better scaling in a distributed environment.

# CHAPTER II

# RELATED WORK

## 2.1 Line Integral Convolution

The original line integral convolution algorithm was created by Cabral and Leedom (1993) as a method for visualizing vector fields. LIC merges image convolution with particle advection to provide a global view of field flow. A number of methods have been spawned from this, such as Oriented Line Integral Convolution (OLIC) conceived by Wegenkittl, Groller and Purgathofer (1997) which uses sparse noise as an input and an asymmetric convolution kernel to provide an improved view of the direction of field flow. Work has also been done by Okada and Lane (1998) to aid in flow feature detection and improve LIC image contrast.

## 2.2 Fast LIC

Several extensions have been proposed for decreasing the computational complexity of LIC. These include both optimizations to the convolution process as well as methods to reduce the required number of integral curves. Stalling and Hege (1995) proposed a modification to the advection stage referred to as Fast and Resolution Independent LIC (FastLIC) which used the similarity of integral curves for neighboring pixels to reduce the number of integral curves by two orders of magnitude. Meanwhile, Wegenkittl and Groller (1997) created an extension for OLIC referred to as Fast Rendering of OLIC (FROLIC) which uses a convolution approximation over disks as opposed to individual pixels.

## 2.3 Parallel Particle Advection

Bethel, Childs and Hansen (2013) survey parallel approaches for particle advection. Their treatment focuses on two main approaches for distribution

of advection tasks. These approaches are parallelization over particles and parallelization over data blocks. With parallelization over particles (POP), a set number of seed particles are assigned to each node. As advection is performed, POP loads sections of the input mesh (data blocks) into memory, based on the position of each particle. Parallelization over data (POD), by contrast, assigns blocks to each node and transfers particles between nodes as they pass between blocks. However, they note that both options tend to perform poorly due to workload imbalance. For POP, this bottleneck is a result of I/O overhead from requesting data blocks, while POD is affected by imbalances in particle assignment over nodes.

As an extension of POP and POD, Pugmire, Childs, Garth, Ahern and Weber (2009) offered a hybrid approach which dynamically assigns blocks and particles to nodes based on the current node and computation states. They found that this technique often works better than POD or POP, especially when the structure of the vector field is not yet well understood.

A number of studies have also been done on the parallelization of particle advection for specific visualization techniques. These include an examination of parallel particle tracing by Peterka et al. (2011) as well as one by Camp, Childs, Garth, Pugmire and Joy (2012) on parallel stream surface generation. In addition Müller, Camp, Hentschel and Garth (2013) designed a method to improve parallel advection load balancing via work-requesting-based dynamic scheduling.

Significant work has also gone into hybrid parallelism (i.e., incorporating both shared- and distributed-memory parallelism), including CPU architectures Camp, Garth, Childs, Pugmire and Joy (2011), GPU architectures Camp et al.

(2013), and workload analysis across many architectures (including multi-GPU architectures) Childs, Biersdorff, Poliakoff, Camp and Malony (2014).

### 2.4 Parallel LIC

Zöckler, Stalling and Hege (1997) performed prior work on parallel LIC which included a number of improvements to FastLIC that impact parallelization of animated LIC as well as a study of parallel performance over time and image space. They also examined how adjustments to convolution kernel buffer updates can reduce inter-node communication. Our work contrasts with theirs as it focuses on parallelization of LIC advection rather than convolution.

# CHAPTER III

# OVERVIEW

To investigate our research questione, we needed an efficient, parallel infrastructure. For this study, we chose to use the VisIt Parallel Integral Curve System (hereafter referred to as PICS). In this chapter, we describe PICS as well as the details of extending it to provide the correct functionality necessary for the convolution process. Finally, we offer an extension to LIC which uses procedural generation to reduce the I/O and cross-node communication overhead introduced by input pixel lookup during convolution.

## 3.1 PICS

Analysis of large amounts of data requires the ability to harness vast amounts of computational power. The VisIt visualization and analysis tool is a richly featured scientific application, built to visualize massive data sets Childs et al. (2005). For this reason, VisIt is designed to make use of the levels of distributed parallelism found on leading edge supercomputers. VisIt provides an extensive suite of tools for performing all manner of scientific visualization. Included among these is the Parallel Integral Curve System (PICS), meant for distributed computation of particle advection. PICS provides VisIt with a method for distributing the advection workload used by its integral curve techniques. While VisIt currently lacks an implementation of LIC, PICS provides the necessary foundation for implementing such an algorithm. The use of a mature parallel framework like PICS for this study serves to reduce implementation time. It additionally minimizes the need to manage all the subtleties of optimizing for supercomputing environments. For these reasons we have selected PICS as our framework in favor of constructing a parallel advection system ourselves. In this way we are better able to focus on

optimizing and testing LIC with parallelization instead of optimizing the parallel framework for the associated hardware.

### 3.2 Algorithm Overview

The PICS system splits integral curve algorithms into two primary parts. The first is that of an individual integral curve. The curve's construction is defined in terms of how it is built up by the advection process. This includes a way of checking if specific termination criteria have been met. For parallelization purposes, curves also require a serialization method that informs PICS of how to pass them between different nodes. The second part is an extensions of the base PICS filter. This defines both the placement of advection seeds, as well as instructions for processing the curves returned by PICS itself. It is this second part which handles advection initialization as well as the conversion of PICS output into a useful format. The actual generation of individual curves is handled by PICS itself, requiring only that the developer appropriately define their structure. PICS allows for the selection of a parallelization scheme; this includes the aforementioned POP and POD techniques, as well as the hybrid approach suggested by Pugmire et al. (2009). As the focus of our work is on how LIC's structure decreases cross-node communication, we have chosen to utilize the POD scheme.

LIC kernel construction mapped cleanly to the PICS integral curve structure. That said, LIC processing had one key difference from default PICS processing: LIC advection is from one cell boundary to the next, not in individual steps. Therefore, at each step of the advection, we check the current cell and compare it against the cell seen in the previous step. If these differ we have entered a new cell and can append it to our convolution kernel. Otherwise, we add nothing and move on to the next advection step. Termination constraints are two fold for

LIC. Obviously, we must compare the number of cells currently stored in our kernel against the maximum step size. However, there is an additional step count that must be tracked. Due to the possibility that a cell might contain a singularity, we also track the number of steps made within a given cell. If this number exceeds some threshold, we assume that we are circling a singularity and terminate the curve. Finding a good value for this cell step count could serve as useful future work, but for our purposes we settle on a cutoff of 30 intra-cell steps.

Finally, while it is possible to perform LIC for arbitrary seed placements, we elected to utilize the dense seeding method of standard LIC. For this seeding method, our LIC filter computes the centers of every cell defined within the vector field and places a seed particle at that location. These seeds are then passed to PICS for integral curve generation. Upon return from PICS execution, this filter performs the LIC convolution phase. Each returned curve includes a kernel array with discrete cell locations recorded during advection. By passing these indices to a procedural noise generator, we can map kernel cells to input pixel values and perform an averaging of cell values for each kernel. In this way, the final pixel values of each cell in the field can be computed. For pseudocode of this parallel LIC process, refer to Algorithm 1.

### 3.3 Procedural Noise Generation

The standard PICS process uses a white noise image as the convolution input, as it helps ensure reasonable feature contrast for LIC output. Though the convolution stage requires consistent mappings between kernel elements and pixel values, the exact values are unimportant provided distribution is reasonably random. Using this property, we extend the image input step of LIC for use in a distributed parallel environment. This is done via the use of a counter-based

**Algorithm 1** Pseudocode for PICS line integral convolution

```
for cell ∈ block do
  block.seedPoints.Append(GetCenterCoords(cell))
end for
for all seed ∈ block do
  curve = new LICCurve
  while not seed.Terminated and seed.InBounds(block) do
    seed.Advect()
    if not seed.InBounds(block) then
      transferSeed
    else if seed.GetCurrentCell() not seed.LastCell then
      curve.SaveStep(seed.GetCurrentCell())
      seed.LastCell = seed.GetCurrentCell()
    end if
  end while
  seed.Reset()
  while not seed.Terminated and seed.InBounds(block) do
    seed.ReverseAdvect()
    if not seed.InBounds(block) then
      transferSeed
    else if seed.GetCurrentCell() not seed.LastCell then
      curve.SaveStep(seed.GetCurrentCell())
      seed.LastCell = seed.GetCurrentCell()
    end if
  end while
  block.curves[GetCellIndex(seed)] = curve
end for
for i = 0 to numCells do
  outPixel[i] = GetAverage(block.curves[i].GetKernel())
end for
```

random number generator (CBRNG), namely the technique proposed by Salmon, Moraes, Dror and Shaw (2011).

We note that typical pseudorandom number generators such as the C rand() function generate their results by successive application of an algorithm to a seed value. Each call of the PRNG modifies the seed and returns this new seed as its next result. This means that getting the $n^{th}$ random number for a sequence requires n iterations of the generator. CBRNGs instead allow us to index into the generator, directly returning the $n^{th}$ random number in a single step. By utilizing the cell index as our CBRNG input we can consistently map every cell of a vector field to a randomly generated, yet consistent, value. In order to vary this image for successive runs of LIC, we simply provide the image generator an offset which is added to all input indices.

For parallel purposes, this seed value is generated using rand() by the MPI master process at runtime and broadcast to all slave processes, with each one maintaining a local instance of the image generator. Given the same seed and same generator, two different nodes are then able to agree on the value of a particular cell. This use of a seeded CBRNG removes the need to store the input in memory and requires only a single communication across all nodes, instead of one for every out-of-block pixel lookup. The result is that LIC convolution can be performed in a distributed parallel fashion without introducing inter-node or disk-based dependencies.

# CHAPTER IV

# EXPERIMENT OVERVIEW

## 4.1 Data Sets and Resolutions

For our experiments we selected several data sets to test against. Each data set represents a common scenario that LIC might encounter, each affecting cross-node communications in different ways. For every data set we examined several different resolutions, as the number of mesh cells has a strong influence on the chance that an integral curve will cross a block boundary, as well as determining how many integral curves are created. The resolutions selected were 100x100 (10,000 cells), 500x500 (250,000 cells), and 1000x1000 (1,000,000 cells). The description of each data set is as follows:

- **Flat Field**: A constant field in which all vectors have the same magnitude and direction. This tested performance under a situation with a high likliehood of good load balancing.

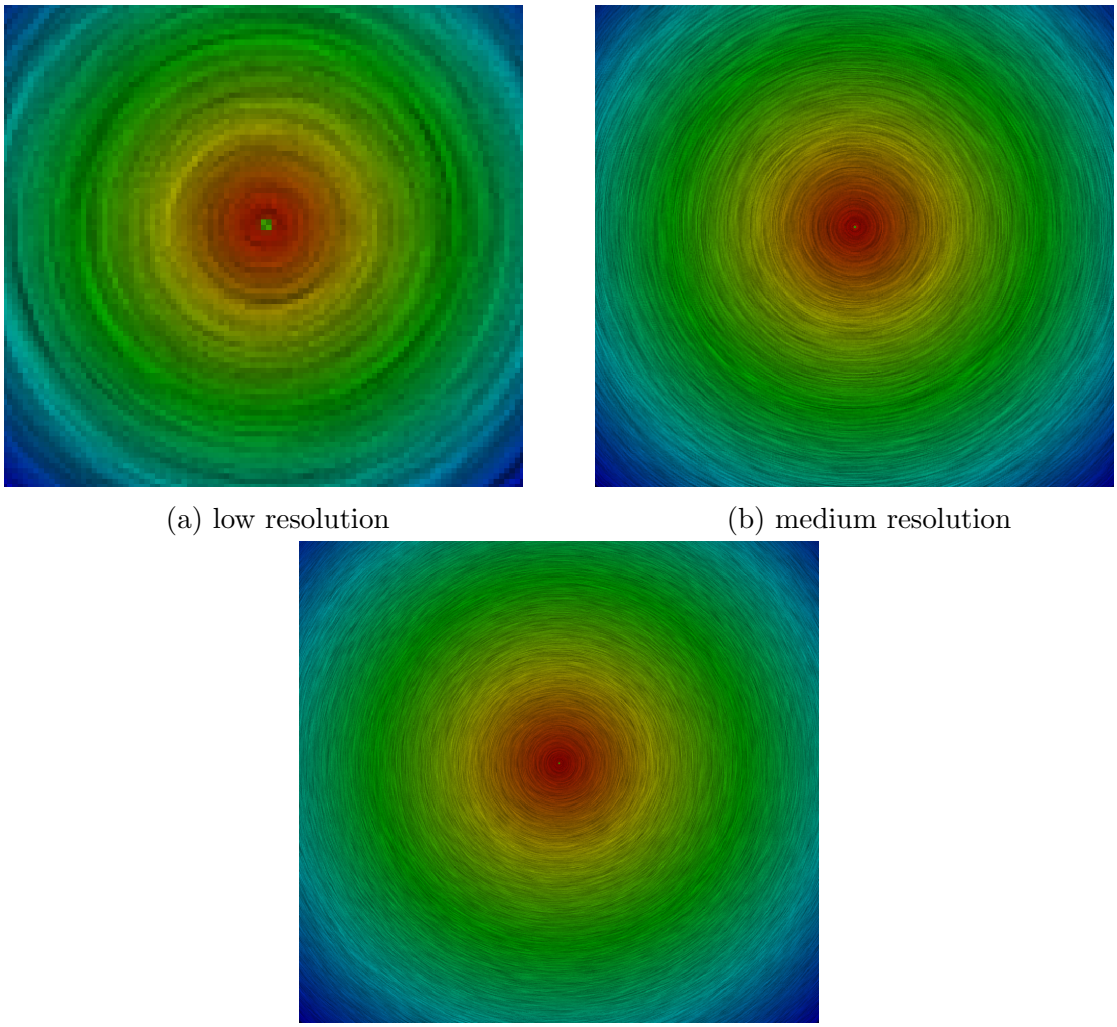
- **Whirlpool**: A circular field where vectors follow a spiral around a central singularity. This has potential for many cross-node communications when the singularity is near a block boundary.

- **Single-Source**: A field with a single source location from which all vectors flow outwards. This tends to push particles from one block to all other blocks (high load imbalance).

- **Single-Sink**: A field with a single sink location from which all vectors flow inwards. This tends to pull many particles from all other blocks to a single block (high load imbalance).

### 4.2 Workloads

**4.2.1 LIC.** We varied our tests across the 4 data sets and 3 resolutions discussed in Section 4.1. Since the number of steps can impact parallel advection performance, we chose to take measures for several LIC step counts on each possible combination of data set and resolution. The selected maximum step counts of 5, 10, and 20 represent likely options for real world runs of line integral convolution. Overall, we ran 36 different tests for parallel LIC advection. Examples of our output for each resolution can be found under Figure 3.



(a) low resolution (b) medium resolution

(c) high resolution

*Figure 3.* Example of line integral convolution (LIC) applied to the spiral test at the three different resolutions. Outputs are colorized to represent the field magnitude at each pixel.

**4.2.2 Streamlines.** To serve as a baseline, streamlines were also generated for the same data sets and resolutions as LIC. Since LIC utilizes similar advection methods, streamlines functioned as a good point of comparison for the LIC tests. In practice, streamlines tend to only be terminated upon exiting the

volume or after a much larger number of steps than LIC (generally in the range of over a thousand advection iterations). Because of this property, we chose to use a cutoff of approximately 1000 steps for all streamline tests. As streamlines are usually forward advected (as opposed to the forward and backwards advection of LIC), we opted to perform only forward advections for our baseline tests. Additionally, in order to isolate our tests to a workload balance comparison, we set the streamline parameters for number of particles and total advection steps to match a number of total advections comparable to its associated LIC test. For LIC, this was computed as:

$$numSL * numSLSteps = 2 * numCells * numLICSteps$$

For example, a run of LIC on a 100x100 mesh (99*99 cells) for 10 steps would perform $2 * (99 * 99) * 10 \approx 196020$ steps. We can then generate $11 * 18 = 198$ streamlines with a step count of 990 for a total of $(11 * 18) * 990 \approx 196020$ advection steps. While early termination can result in a difference between LIC and streamlines, a maximum step size of $\frac{fieldWidth}{numSLSteps}$ was chosen to reduce the likelihood of a disproportionate number of early streamline terminations while still allowing streamlines to cross block boundaries. This ensures that both our LIC tests and our streamline tests perform approximately the same number of advections within an acceptable margin of error.

### 4.3 Machines

All tests were performed on Alaska, the CDUX research group's compute cluster. Alaska includes a head node and 4 compute nodes. Each of the compute nodes contains a single Intel Xeon E5-1650v3 processor with a 3.5GHz clock speed

as well as 32GB of local DDR4 RAM and a 256GB SSD. As our focus was on distributed-memory parallel as opposed to shared-memory, we elected to distribute work across all four of these nodes, using a single processor core on each one.

### 4.4 Timing Methodology

For all experiments, timings were collected with VisIt's existing timing infrastructure. In each test we obtained the maximum time taken across all nodes as well as the average number of bytes communicated between nodes. To assist in understanding the load imbalance we use the percent imbalance metric described by Pearce, Gamblin, de Supinski, Schulz and Amato (2012). This metric uses the maximum and average time to measure how much performance is wasted due to workload imbalance, with the optimal value being 0%.

### 4.5 Step Count

With particle advection algorithms, construction of an integral curve continues until a termination condition is met. In most cases this is because the particle has left the edge of the volume, has encountered a singularity and cannot move further, or some advection constraint has been met (such as maximum distance or time). In the first two cases, no useful information can be gained by continuing to advect and we can terminate the integral curve at this point. For the last condition, advection could continue but has been stopped anyways, possibly due to limited computational resources or because it is believed that continuing would not increase understanding of the simulation. LIC does share in these termination constraints but the usefulness of taking additional steps drops off much quicker than other methods. Additionally, as noted by Cabral and Leedom (1993), high step counts can negatively impact LIC output. As such the generation of line integrals not only allows for, but expects a much smaller maximum step count than

alternative advection algorithms. For example, Cabral and Leedom used a LIC step count of ten to generate most of their results. It should be noted that, for PICS, this LIC step count is different than the advection step count referenced for streamlines (which is closer to the intra-cell step count referenced in Section 3.2). However, as LIC does minimal computation (and no cross-node communication) for advections that don't cross cell boundaries, we chose to compare streamline step counts to LIC cell step counts.

# CHAPTER V

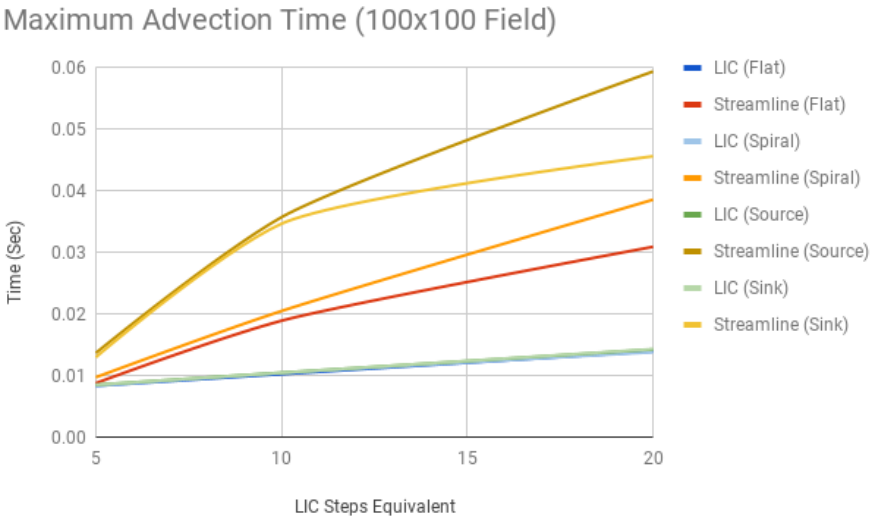
# RESULTS

Within this section we examine the results of our test runs divided across the metrics of maximum runtime, percent imbalance (as described in Section 4.4), and average communication load. Specific results for all tests can be found under Appendix A.

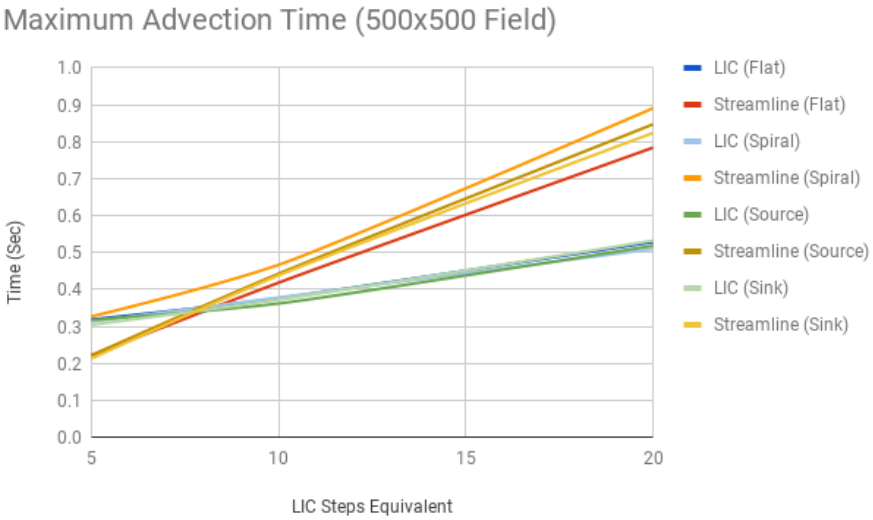
## 5.1 Maximum Node Time

This metric is a measure of the maximum time spent advecting particles across all four nodes. We found that, for small step counts, the maximum LIC advection time is similar to that of streamlines, taking slightly longer for larger resolutions. However, as the step size was increased, we noticed that the maximum time grew much faster for streamlines than LIC; we can see this displayed in Figure 4c. We also found that LIC times were more consistant across different types of vector fields. This is especially noticeable for small resolution tests (see Figure 4a). This consistancy makes intuitive sense when considering our assumption that short LIC curves have fewer chances to cross block boundaries compared to the longer streamlines. Therefore, the shape of the vector field has less chance to influence LIC. Across all resolutions and step sizes we noticed that the maximum times were fastest for the flat dataset, being slowest for the sink and source data sets for the small and large resolutions. As seen in Figure 4b, the 500x500 resolution shows a smaller deviation for streamline times than for other resolutions. It is possible that this resolution was small enough to limit the number of streamline particles that could cross block boundaries, while being large enough to allow these particles to advect further before running into these boundaries. In general we observed a 1.5x to 3x speedup for LIC over streamlines at 20 steps across all resolutions and data

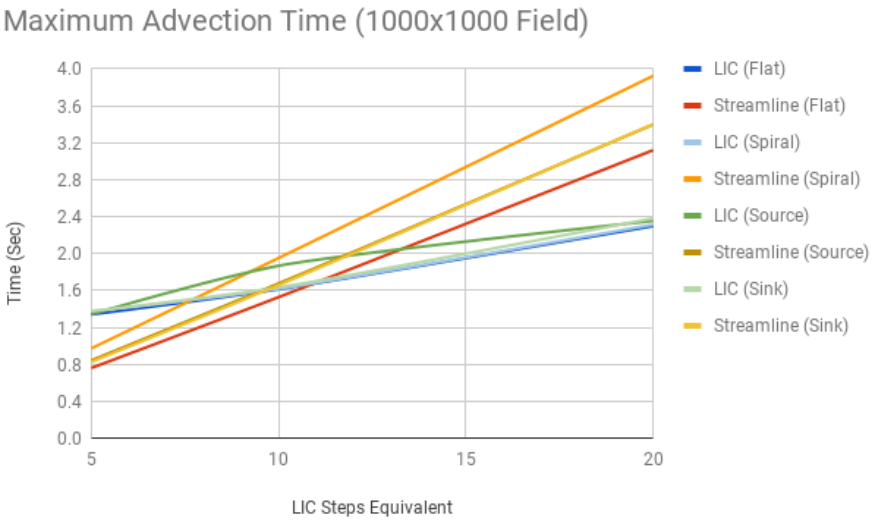
sets, as well as noticing better runtime consistency for LIC across different data sets.




(a) Maximum advection time taken across all four nodes at a 100x100 field resolution.




(b) Maximum advection time taken across all four nodes at a 500x500 field resolution.

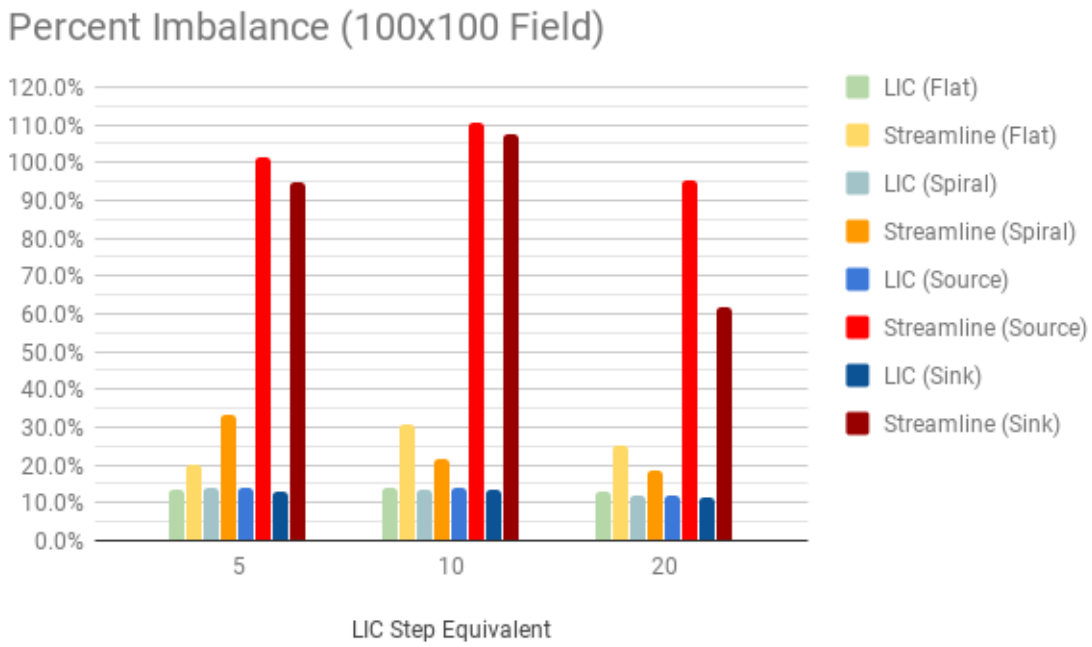



(c) Maximum advection time taken across all four nodes for 1000x1000 field resolution.

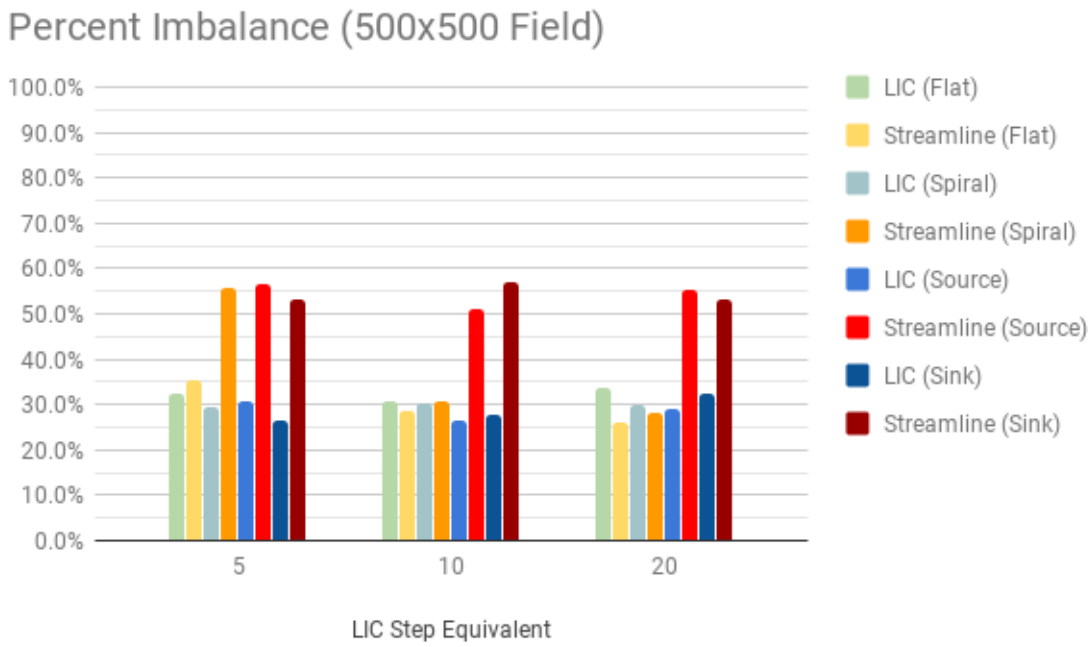
### 5.2 Percent Imbalance

Here we compared the mean completion time against the maximum completion time to get an idea of how different workloads could become imbalanced between the two algorithms. We found a major load imbalance for streamlines under small LIC step counts for the unbalanced source and sink data sets (see Figure 5a). This makes sense when we consider that these data sets will tend to bias particles towards or away from specific blocks. In the case of sink this results in the overloading of a specific node while source can cause node starvation. For both cases we are left with a large difference between minimum and maximum execution time. As seen in Figures 5b and 5c, larger step sizes mitigate this issue. However, LIC was still left performing more consistently across nodes for unbalanced vector fields. As seen in our results, there is a rough parity between LIC and streamlines for the more balanced spiral and flat data sets, with both being roughly 30% off of optimal balance on larger data sets. There is one

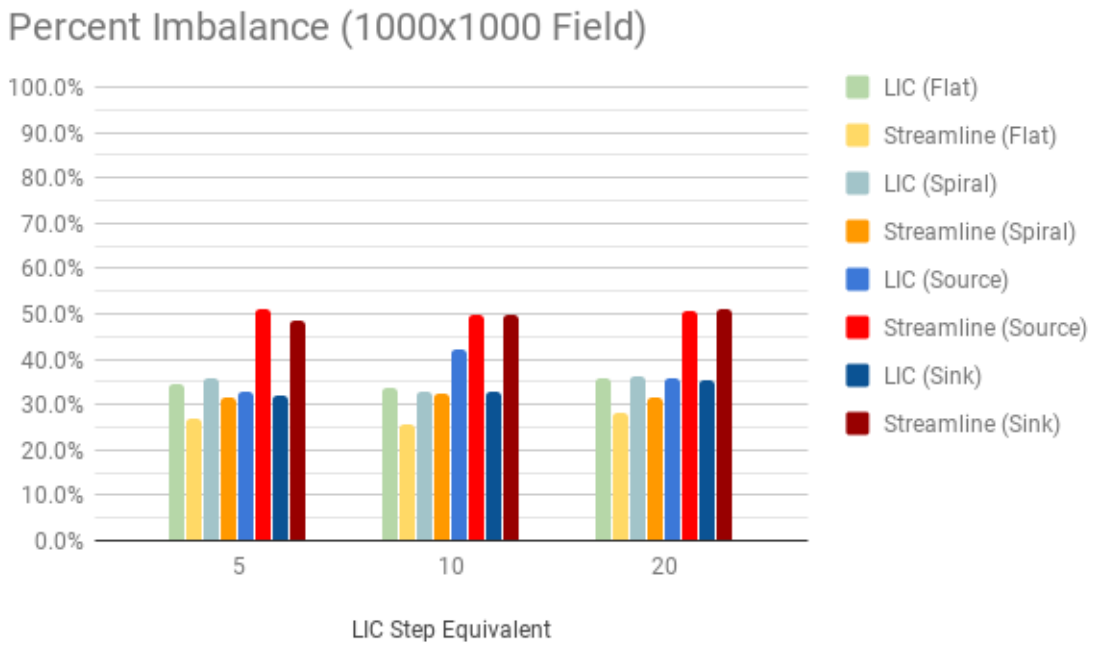
exception with the 500x500 spiral test in that streamlines for small step counts have a comparable imbalance to the source and sink data sets. It is less clear why this occurs, though it is possibly a fluke of the specific combination of data set and resolution (similar to the cause of the reduced variance observed in 4b). Overall, we note that the source and sink tests suffer from imbalance factors for streamlines at least 15-20% higher than LIC. In the case of small resolutions this difference becomes even more noticeable for imbalanced data sets. This may be due in part to the low number of streamline particles serving to decrease the number of unbalanced communications necessary to affect the ratio of curves between any two nodes.




(a) Percent imbalance measure for 100x100 field resolution.




(b) Percent imbalance measure for 500x500 field resolution.

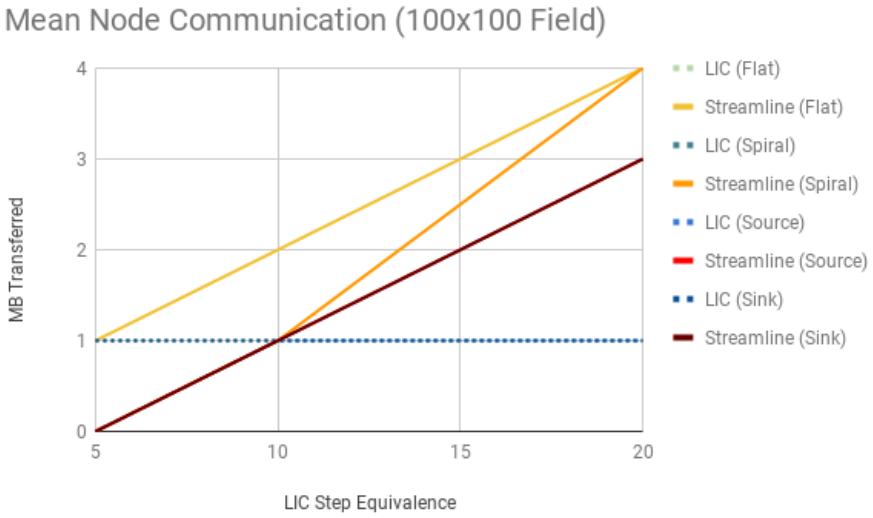



(c) Percent imbalance measure for 1000x1000 field resolution.

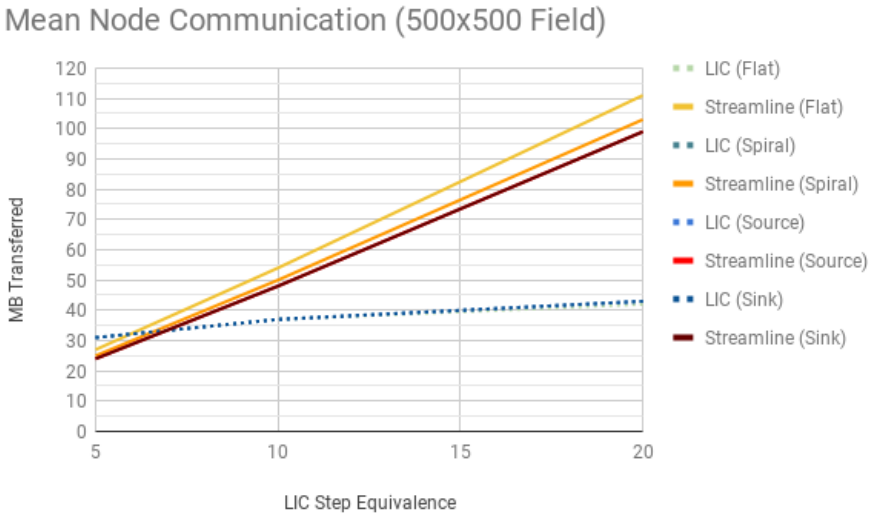
### 5.3 Average Communication Load

The final measure taken was the mean communication load across all nodes. This represents the total amount of data (in MB) passed between nodes during the advection process. These communications primarily happen when an advected particle leaves the block of the field that its associated node is working on, and must be passed to a neighboring node. For all tests, LIC showed a strong reduction for communication load when compared to streamlines. Smaller step sizes showed rough parity between streamlines and LIC. For twenty LIC steps however, streamlines required communicating an average of 2.5x as much data between nodes. In the worst case (shown in Figure 6c), this difference amounted to passing approximately 175MB of data for LIC while streamlines passed nearly half a gigabyte of integral curve information. Interestingly, this worst case is not on the source or sink data sets, but on flat. This may be caused by the constant field

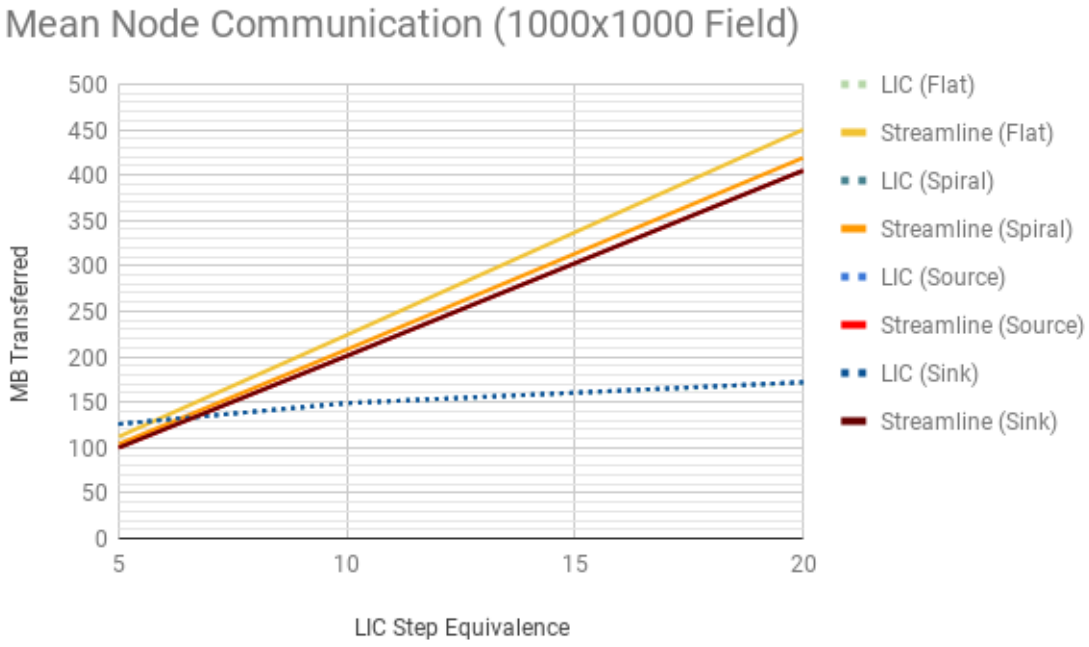
direction meaning that the streamlines on one half of the field are all being pushed into the other half of the field. This is a case that wouldn't cause load balance due to early termination of streamlines in the other half but could still cause high communications overhead.




(a) Average node communication load for 100x100 field resolution.




(b) Maximum node communications load for 500x500 field resolution.




(c) Maximum node communications load for 1000x1000 field resolution.

# CHAPTER VI

# CONCLUSION

We have compared the line integral convolution and streamlines advection algorithms in a distributed-memory parallel environment. The shorter integral curves inherent to LIC showed a positive influence on load balancing compared to equivalently sized streamline calculations. Overall this work finds a notable improvement for LIC over streamlines in message overhead as well as improved workload balancing and consistency. This serves to motivate the usage of parallel LIC for vector field analysis where scaling over a distributed system becomes necessary. In such cases the lower message overhead of LIC is a desired quality that reduces the impact of network related performance bottlenecks.

In terms of future work, we note that our tests were limited by available hardware. While the system used was sufficient to get a feel for the differences between LIC and streamlines, it would be ideal to analyze the difference between the two algorithms on a much larger number of nodes. Furthermore, it would be valuable to look at distributed-memory parallel LIC with shared-memory parallel operations on each compute node. We would also like to examine how the parallel over particles and hybrid approaches referenced by Camp et al. (2012) and Pugmire et al. (2009) influence LIC load balancing. Finally, there are a number of further extensions to LIC (see Sections 2.1 and 2.2) which would be useful to test in a parallel context.

# APPENDIX

## TABLES

| Maximum: 100x100 Field Resolution | | | |
|---|---|---|---|
| **Test** | **5 Steps** | **10 Steps** | **20 Steps** |
| LIC (Flat) | 0.008 | 0.010 | 0.014 |
| Streamline (Flat) | 0.009 | 0.019 | 0.031 |
| LIC (Spiral) | 0.008 | 0.010 | 0.014 |
| Streamline (Spiral) | 0.010 | 0.021 | 0.039 |
| LIC (Source) | 0.009 | 0.010 | 0.014 |
| Streamline (Source) | 0.014 | 0.036 | 0.059 |
| LIC (Sink) | 0.009 | 0.010 | 0.014 |
| Streamline (Sink) | 0.013 | 0.035 | 0.046 |

Table A.1. Maximum node advection time (in seconds) for various LIC equivalent step counts on 100x100 field resolution.

| Maximum: 500x500 Field Resolution | | | |
|---|---|---|---|
| **Test** | **5 Steps** | **10 Steps** | **20 Steps** |
| LIC (Flat) | 0.318809 | 0.376098 | 0.52861 |
| Streamline (Flat) | 0.221561 | 0.418506 | 0.784233 |
| LIC (Spiral) | 0.312902 | 0.378434 | 0.509695 |
| Streamline (Spiral) | 0.32704 | 0.466844 | 0.890946 |
| LIC (Source) | 0.317805 | 0.362125 | 0.518558 |
| Streamline (Source) | 0.221908 | 0.442928 | 0.847323 |
| LIC (Sink) | 0.304354 | 0.372294 | 0.532403 |
| Streamline (Sink) | 0.213011 | 0.438187 | 0.824061 |

Table A.2. Maximum node advection time (in seconds) for various LIC equivalent step counts on 500x500 field resolution.

| Maximum: 1000x1000 Field Resolution | | | |
|---|---|---|---|
| **Test** | **5 Steps** | **10 Steps** | **20 Steps** |
| LIC (Flat) | 1.34112 | 1.61386 | 2.29797 |
| Streamline (Flat) | 0.760751 | 1.52802 | 3.11975 |
| LIC (Spiral) | 1.37416 | 1.61913 | 2.31357 |
| Streamline (Spiral) | 0.973355 | 1.95205 | 3.92439 |
| LIC (Source) | 1.34172 | 1.86823 | 2.35398 |
| Streamline (Source) | 0.843754 | 1.67645 | 3.39724 |
| LIC (Sink) | 1.37642 | 1.63351 | 2.38128 |
| Streamline (Sink) | 0.826903 | 1.65865 | 3.40103 |

Table A.3. Maximum node advection time (in seconds) for various LIC equivalent step counts on 1000x1000 field resolution.

| Percent Imbalance: 100x100 Field Resolution | | | |
|---|---|---|---|
| **Test** | **5 Steps** | **10 Steps** | **20 Steps** |
| LIC (Flat) | 13.5% | 14.1% | 12.9% |
| Streamline (Flat) | 19.9% | 30.8% | 24.9% |
| LIC (Spiral) | 13.9% | 13.4% | 11.7% |
| Streamline (Spiral) | 33.2% | 21.4% | 18.6% |
| LIC (Source) | 13.9% | 14.0% | 11.7% |
| Streamline (Source) | 101.5% | 110.5% | 95.5% |
| LIC (Sink) | 13.2% | 13.6% | 11.2% |
| Streamline (Sink) | 94.8% | 107.3% | 62.0% |

Table A.4. Node percent imbalance measures for various LIC equivalent step counts on 100x100 field resolution.

| Percent Imbalance: 500x500 Field Resolution | | | |
|---|---|---|---|
| **Test** | **5 Steps** | **10 Steps** | **20 Steps** |
| LIC (Flat) | 32.6% | 30.6% | 33.8% |
| Streamline (Flat) | 35.4% | 28.7% | 25.9% |
| LIC (Spiral) | 29.5% | 30.2% | 29.9% |
| Streamline (Spiral) | 55.8% | 30.5% | 28.3% |
| LIC (Source) | 30.9% | 26.3% | 29.0% |
| Streamline (Source) | 56.5% | 51.0% | 55.2% |
| LIC (Sink) | 26.7% | 27.7% | 32.5% |
| Streamline (Sink) | 53.2% | 57.0% | 53.3% |

Table A.5. Node percent imbalance measures for various LIC equivalent step counts on 500x500 field resolution.

| Percent Imbalance: 1000x1000 Field Resolution | | | |
|---|---|---|---|
| **Test** | **5 Steps** | **10 Steps** | **20 Steps** |
| LIC (Flat) | 34.4% | 33.5% | 35.6% |
| Streamline (Flat) | 26.8% | 25.8% | 28.2% |
| LIC (Spiral) | 35.9% | 32.9% | 36.3% |
| Streamline (Spiral) | 31.6% | 32.3% | 31.3% |
| LIC (Source) | 33.0% | 42.3% | 35.9% |
| Streamline (Source) | 50.8% | 49.7% | 50.8% |
| LIC (Sink) | 31.8% | 32.7% | 35.3% |
| Streamline (Sink) | 48.3% | 49.9% | 50.9% |

Table A.6. Node percent imbalance measures for various LIC equivalent step counts on 1000x1000 field resolution.

| Mean Communication: 100x100 Field Resolution | | | |
|---|---|---|---|
| **Test** | **5 Steps** | **10 Steps** | **20 Steps** |
| LIC (Flat) | 1 | 1 | 1 |
| Streamline (Flat) | 1 | 2 | 4 |
| LIC (Spiral) | 1 | 1 | 1 |
| Streamline (Spiral) | 0 | 1 | 4 |
| LIC (Source) | 0 | 1 | 1 |
| Streamline (Source) | 0 | 1 | 3 |
| LIC (Sink) | 1 | 1 | 1 |
| Streamline (Sink) | 0 | 1 | 3 |

Table A.7. Mean node communication (in MB) for various LIC equivalent step counts on 100x100 field resolution.

| Mean Communication: 500x500 Field Resolution | | | |
|---|---|---|---|
| **Test** | **5 Steps** | **10 Steps** | **20 Steps** |
| LIC (Flat) | 31 | 37 | 42 |
| Streamline (Flat) | 27 | 54 | 111 |
| LIC (Spiral) | 31 | 37 | 43 |
| Streamline (Spiral) | 25 | 50 | 103 |
| LIC (Source) | 31 | 37 | 43 |
| Streamline (Source) | 24 | 48 | 99 |
| LIC (Sink) | 31 | 37 | 43 |
| Streamline (Sink) | 24 | 48 | 99 |

Table A.8. Mean node communication (in MB) for various LIC equivalent step counts on 500x500 field resolution.

| Mean Communication: 1000x1000 Field Resolution | | | |
|---|---|---|---|
| **Test** | **5 Steps** | **10 Steps** | **20 Steps** |
| LIC (Flat) | 126 | 149 | 171 |
| Streamline (Flat) | 112 | 224 | 450 |
| LIC (Spiral) | 126 | 149 | 172 |
| Streamline (Spiral) | 104 | 208 | 419 |
| LIC (Source) | 126 | 149 | 172 |
| Streamline (Source) | 100 | 201 | 405 |
| LIC (Sink) | 126 | 149 | 172 |
| Streamline (Sink) | 100 | 201 | 405 |

Table A.9. Mean node communication (in MB) for various LIC equivalent step counts on 1000x1000 field resolution.

REFERENCES CITED


Bethel, E. W., Childs, H. & Hansen, C. (2013). *High performance visualization: Enabling extreme-scale scientific insight*. Chapman & Hall/CRC.

Cabral, B. & Leedom, L. C. (1993). Imaging vector fields using line integral convolution. In *Proceedings of the 20th annual conference on computer graphics and interactive techniques* (pp. 263–270). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/166117.166151` doi: 10.1145/166117.166151

Camp, D., Childs, H., Garth, C., Pugmire, D. & Joy, K. I. (2012, Oct). Parallel stream surface computation for large data sets. In *Ieee symposium on large data analysis and visualization (ldav)* (p. 39-47). doi: 10.1109/LDAV.2012.6378974

Camp, D., Garth, C., Childs, H., Pugmire, D. & Joy, K. I. (2011, November). Streamline Integration Using MPI-Hybrid Parallelism on a Large Multicore Architecture. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, *17*, 1702-1713. doi: http://doi.ieeecomputersociety.org/10.1109/TVCG.2010.259

Camp, D., Krishnan, H., Pugmire, D., Garth, C., Johnson, I., Bethel, E. W., ... Childs, H. (2013, May). GPU Acceleration of Particle Advection Workloads in a Parallel, Distributed Memory Setting. In *Proceedings of eurographics symposium on parallel graphics and visualization (egpgv)* (pp. 1–8). Girona, Spain.

Childs, H., Biersdorff, S., Poliakoff, D., Camp, D. & Malony, A. D. (2014, December). Particle Advection Performance Over Varied Architectures and Workloads. In *Ieee international conference on high performance computing (hipc)* (pp. 1–10). Goa, India.

Childs, H., Brugger, E. S., Bonnell, K. S., Meredith, J. S., Miller, M., Whitlock, B. J. & Max, N. (2005). A contract-based system for large data visualization. In *Proceedings of ieee visualization 2005* (pp. 190–198).

Müller, C., Camp, D., Hentschel, B. & Garth, C. (2013, Oct). Distributed parallel particle advection using work requesting. In *2013 ieee symposium on large-scale data analysis and visualization (ldav)* (p. 1-6). doi: 10.1109/LDAV.2013.6675152

Okada, A. & Lane, D. (1998, 01). Enhanced line integral convolution with flow feature detection. In *Proceedings of spie - the international society for optical engineering* (Vol. 3017).

Pearce, O., Gamblin, T., de Supinski, B. R., Schulz, M. & Amato, N. M. (2012). Quantifying the effectiveness of load balance algorithms. In *Proceedings of the 26th acm international conference on supercomputing* (pp. 185–194). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/2304576.2304601` doi: 10.1145/2304576.2304601

Peterka, T., Ross, R., Nouanesengsy, B., Lee, T. Y., Shen, H. W., Kendall, W. & Huang, J. (2011, May). A study of parallel particle tracing for steady-state and time-varying flow fields. In *2011 ieee international parallel distributed processing symposium* (p. 580-591). doi: 10.1109/IPDPS.2011.62

Pugmire, D., Childs, H., Garth, C., Ahern, S. & Weber, G. H. (2009, Nov). Scalable computation of streamlines on very large datasets. In *Proceedings of the conference on high performance computing networking, storage and analysis* (p. 1-12). doi: 10.1145/1654059.1654076

Salmon, J. K., Moraes, M. A., Dror, R. O. & Shaw, D. E. (2011, Nov). Parallel random numbers: As easy as 1, 2, 3. In *2011 international conference for high performance computing, networking, storage and analysis (sc)* (p. 1-12). doi: 10.1145/2063384.2063405

Stalling, D. & Hege, H.-C. (1995). Fast and resolution independent line integral convolution. In *Proceedings of the 22nd annual conference on computer graphics and interactive techniques* (pp. 249–256). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/218380.218448` doi: 10.1145/218380.218448

Wegenkittl, R. & Groller, E. (1997, Oct). Fast oriented line integral convolution for vector field visualization via the internet. In *Visualization '97., proceedings* (p. 309-316). doi: 10.1109/VISUAL.1997.663897

Wegenkittl, R., Groller, E. & Purgathofer, W. (1997, Jun). Animating flow fields: rendering of oriented line integral convolution. In *Computer animation '97* (p. 15-21). doi: 10.1109/CA.1997.601035

Zöckler, M., Stalling, D. & Hege, H.-C. (1997, July). Parallel line integral convolution. *Parallel Comput.*, *23*(7), 975–989. Retrieved from http://dx.doi.org/10.1016/S0167-8191(97)00039-2 doi: 10.1016/S0167-8191(97)00039-2