Numerical Modeling of Space Plasma Flows: ASTRONUM-2014
ASP Conference Series, Vol. 498
N. V. Pogorelov, E. Audit, and G. P. Zank, eds.

A Screen Space GPGPU Surface LIC Algorithm for Distributed Memory Data Parallel Sort Last Rendering Infrastructures

B. Loring,[1] H. Karimabadi,[2] and V. Rortershteyn[2]

[1]*Lawrence Berkeley Laboratory, Berkeley, California, USA;* bloring@lbl.gov

[2]*Department of Electrical and Computer Engineering, University of California, San Diego, USA;* homakar@gmail.com

Abstract. The surface line integral convolution(LIC) visualization technique produces dense visualization of vector fields on arbitrary surfaces. We present a screen space surface LIC algorithm for use in distributed memory data parallel sort last rendering infrastructures. The motivations for our work are to support analysis of datasets that are too large to fit in the main memory of a single computer and compatibility with prevalent parallel scientific visualization tools such as ParaView and VisIt. By working in screen space using OpenGL we can leverage the computational power of GPUs when they are available and run without them when they are not. We address efficiency and performance issues that arise from the transformation of data from physical to screen space by selecting an alternate screen space domain decomposition. We analyze the algorithm's scaling behavior with and without GPUs on two high performance computing systems using data from turbulent plasma simulations.

1. Introduction

The line integral convolution is a technique for producing dense visualizations of vector fields. The technique works by convolving a noise texture along vector field streamlines to produce streaks of varying intensity. The original LIC algorithm (Cabral & Leedom 1993) was designed for use on images. A two pass method was introduced to address the algorithm's inherent dynamic range truncation and resulting low contrast streaks (Okada & Lane 1997). The image LIC algorithm was later extended to arbitrary geometric surfaces (Stalling 1997) followed by the development of screen space algorithm (Laramee et al. 2003) that allowed the computational power of GPUs to be utilized. The screen space algorithm has been parallelized for use on clusters of GPUs using a hybrid sort-first sort-last algorithm (Bachthaler et al. 2006). However, the necessity of duplicating vector and geometric data on all nodes in the hybrid sort-first sort-last approach limits its applicability to relatively small datasets and is incompatible with popular scientific data visualization frameworks which are based on a distributed memory data parallel sort-last rendering(DMDPSLR) approach. VisIt (Childs et al. 2005) and ParaView (Ayachit 2015) are two examples of popular DMDPSLR based frameworks. In both tools the parallel sort-last rendering infrastructure is provided by IceT (Moreland et al. 2011), a highly optimized distributed memory data parallel image compositing library.

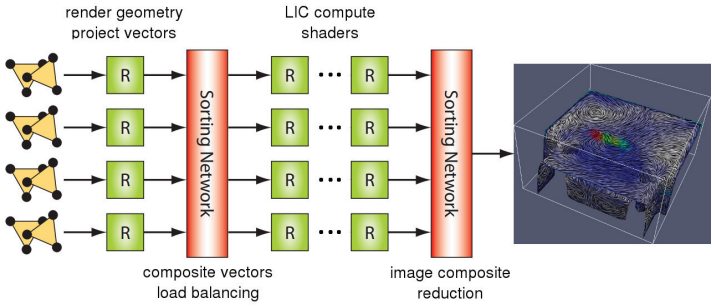

Figure 1. Screen space surface LIC in a DMDPSLR setting is comprised of a sequence of local rendering(blocks) and global sorting operations(vertical bars). During parallel execution a sorting stage is needed during screen space transformation of vector data. See section 2.1.

Here we introduce a parallel screen space GPGPU surface LIC algorithm targeted for use in DMDPSLR based frameworks on arbitrarily large scientific datasets. The motivations for our work are to support the processing of very large scientific datasets, specifically those which are too large to fit in the memory of a single computer, and compatibility with existing sort-last rendering frameworks. The screen space algorithm is implemented as part of the rendering pipeline using OpenGL and therefore can take advantage of the computational power of GPUs where they are available. The very same code can be run without GPUs by using a software based OpenGL implementation which is an important feature as many high-performance computing(HPC) systems do not have GPUs. This work has been integrated into ParaView.

2. Parallelization

2.1. Distributed Memory Data Parallel Sort-Last Rendering

Typically the sort-last rendering can be thought of as occurring in two main phases. The first phase contains a series of local graphics operations specific to the rendering algorithm. The second phase, called image compositing, exchanges and combines local renderings using a sorting algorithm into a single image on one process for display. The screen space surface LIC is atypical in that there are two sorting phases requiring inter-process communication with multiple purely local rendering operations intermixed. The first sort phase occurs during the transformation of vector data into screen space where overlapping regions of the screen space are exchanged and depth composited. The second sort phase performs the traditional image compositing reduction.

This is depicted in figure 1. Local graphics operations are represented by blocks while sorting operations requiring inter-process communication are represented by vertical bars. A vector field associated with surface geometry is input, a rendering stage transforms the vectors into screen space and performs surface rendering, a compositing stage then corrects the vector field in overlapping regions of the screen, a number of subsequent rendering stages compute the LIC in screen space, apply image processing filters, and combine the LIC with a pseduocolor plot, followed by the traditional image

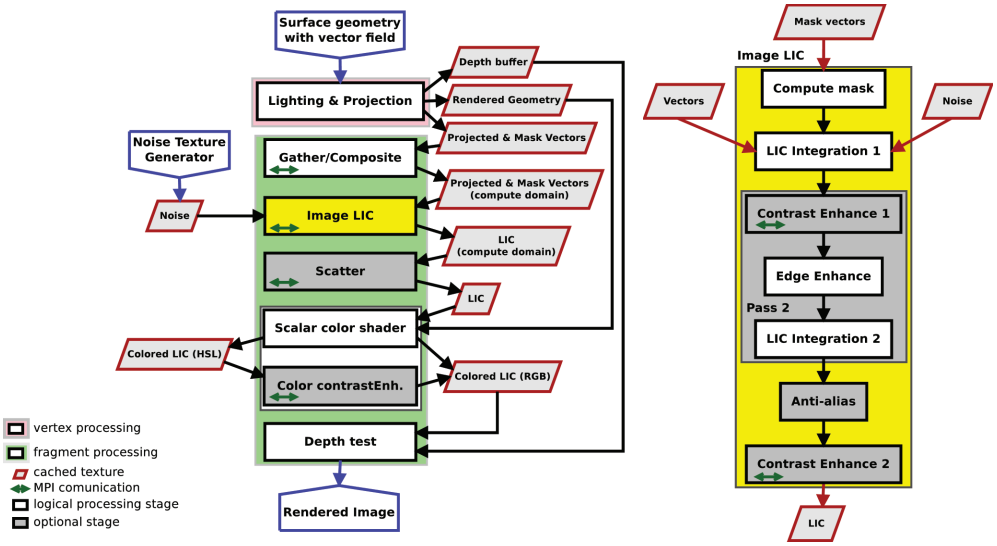

Figure 2. Pipeline. Left: Stages of the DPDMSLR screen surface LIC. Right: Breakout showing the two pass image LIC stage in detail.

compositing stage that reduces locally rendered images into the final result on a single process for viewing.

A high level schematic of our algorithm is shown in the left half of figure 2 with the image LIC stage shown in detail on the right. We have organized the algorithm as a sequential pipeline to deliver high performance in an interactive setting. The result of each stage is cached and will be re-used during interaction if neither that stage's inputs nor its control parameters have been modified. In the following sections we describe the parallelization of the stages of our pipeline.

2.2. Moving Vectors to Screen Space

To obtain the correct vector field in screen space during parallel execution as the vector data is transformed into screen space it needs to be exchanged and depth buffer composited where there is off-process overlap in screen space. This occurs in the gather/composite stage of the pipeline in figure 2. To locate the overlapping regions we first gather a list of screen space extents on all processes. Extents, a light weight data structure describing a region in screen space coordinates, are obtained by projecting the axis aligned bounding box for each block of data to be rendered. The collection of extents organized by process id constitutes the screen space domain decomposition upon which the LIC will be computed. Each process maintains a copy of this list and uses it to coordinate inter-process communication and steer computation as the rendering pipeline executes.

Once all processes have the list of extents organized by process id each can determine the communication necessary for the required point to point exchange of vector data. These exchanges include filling the guard pixel halos required for correct parallel computation. We use non-blocking communication operations so that communication can be overlapped with compositing of incoming data. We use GPU mapped memory for communication buffers to minimize memory overhead.

2.3. LIC Computation

We used a two pass image LIC (Okada & Lane 1997) with contrast enhancement(CE) stages to counteract inherent dynamic range reduction. Our CE stages are based on histogram normalization rather than equalization as the former is less costly to compute in parallel. We found that applying a CE stage on the L channel in HSL color space after the LIC is combined with pseudocolor plots generally increases dynamic range and improves perception of subtle color variations in the pseudocoloring. We included an anti-aliasing filter to remove some of the jagged pixelation in LIC streaks that the high-pass edge enhancing filter tends to introduce. The organization of these stages are shown on the right in figure 2.

When executed in parallel the CE stages require the computation of the global minimum and maximum intensities. This requires the min and max values to be fetched from the GPU and a reduction applied across all processes. When executed in parallel the LIC integration, edge enhancement, and anti-aliasing stages all require some number of guard pixels to produce correct results. To eliminate the need for intermediate guard pixel exchanges, which in addition to communication overhead necessitates moving data on and off from the GPU, we precompute the total number of guard pixels required for all stages. The exchanges that fill the halos are made at the same time vectors are composited in the gather/composite stage. Subsequent computations are nested such that upstream stages compute the solution in down stream stage's guard pixels. This avoids moving data on and off the GPU for communication at a slight increase in compositing and computation costs.

The anti-alias and edge enhancement stages are implemented with convolutions and require $\lfloor r/2 \rfloor$ guard pixels where r is the convolution radius. For the LIC computation the number of guard pixels required on a given extent is a function of the local vector field. We use the maximum magnitude of the vector field on each extent to ensure we have a sufficient number of guard pixels. We need to add the guard pixel halos prior to compositing, but on any given screen space extent the vector field is not correct until after compositing. This requires us to search for the maximum vector magnitude in both the extent under consideration and in overlapping regions of off process extents as well. The number of guard pixels for a given extent is given by

$$g(S_i) = \mathrm{maxv}(S_i) \cdot n_s \cdot d_s \cdot n_p + n_{ee} + n_{aa} \tag{1}$$

where S_i is one of the screen space extents, maxv is a function that computes maximum vector magnitude across the extent and all overlapping regions of other extents, d_s is the screen-space integration step size, n_s is the number of integration steps, n_p is the number of LIC passes, n_{ee} and n_{aa} are the number of guard pixels required for edge enhancement and anti-alias stages.

2.4. Screen Space Domain Decomposition

In the DMDPSLR setting, a reasonable data distribution for I/O and filter operations can result in highly unbalanced work load during rendering. User applied filtering operations can exacerbate load imbalances. For example slicing operations can leave many processes without any data. The computational resources on the processes without data go unused during rendering. This is usually not a problem as most rendering algorithms are relatively fast. However, the issue of good screen space load balancing is a concern because of the LIC's high computational cost. The communication, compositing, and

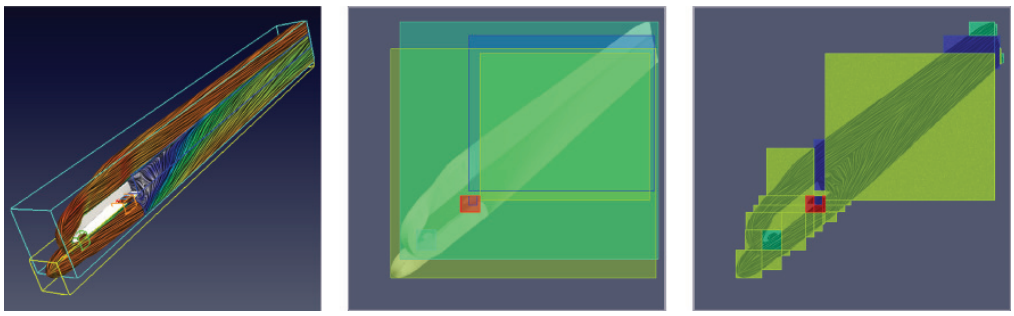

Figure 3. Domain decomposition. The physical space decomposition(left) is transformed into a screen space decomposition(center). Data in overlapping area between processes needs to be exchanged and composited and results in redundant computation. Making the decomposition disjoint(right) eliminates the need to exchange data and redundant computation.

computational costs of the screen space LIC are a function of the screen space domain decomposition. An additional complication of working in screen space is that work load and domain decomposition are strongly view dependent.

An illustrative example of some of the issues is shown in figure 3. In the left panel the surface LIC has been computed on a dataset with 7 blocks distributed across 4 processes. Axis aligned bounding boxes for each block have been included and colored by block id for reference. This domain decomposition results in good load balancing during filtering and I/O portions of the visualization pipeline. However, its transformation into screen space results in a large amount of redundant work and high communication costs during rendering of the surface LIC.

The initial screen space decomposition is obtained by projecting the axis align block bounding boxes into screen space. Each block maps to a screen space extent upon which surface LIC will be computed. We call this the "in-place" decomposition. Two issues that impact performance can arise with the in-place decomposition. First, screen space area that contains no data and thus contributes nothing to the result will be included in communication and compositing as vectors are transformed into screen space and in LIC computations. Second, after the transformation of vectors into screen space overlapping regions have the same vector data and as a result will contain identical results and thus the work done on all but one of the overlapping regions is redundant. These issues can be seen in the middle panel of figure 3 which shows the decomposition colored by process id. Many of the extents in this decomposition have large areas with no data and there is a lot of overlapping area that will result in redundant compositing and computation.

To address these inefficiencies we introduced a repartitioning stage into our vector transform which creates a disjoint screen space decomposition by assigning ownership of overlapping regions to one of the processes. To do this we examine the list of screen space extents one by one, from largest to smallest, and subtract from it the other extents. The subtraction operation is defined such that in cases where there is no overlap between the operands, both are unmodified. In the case where the right operand covers the left entirely the left operand is removed. In the case where the operands overlap, the overlapping region is removed from the left operand, spitting it into a number of smaller extents that cover the remaining area while the right operand is unmodified.

The results replace the original in our list of extents and the process continues. After the subtraction process the remaining set of extents is disjoint. The set of disjoint extents are then scanned and shrunk to tightly bound the data removing empty areas from subsequent communication, compositing, and computation. A pass is made to merge extents on the same process where they completely share an edge. We call the resulting set of screen space extents the "disjoint" decomposition.

Compared to the in-place decomposition the disjoint decomposition computes LIC once for each pixel and the communication and compositing costs are reduced. Data from overlapping regions is sent to the process owning the region rather than exchanged, reducing the communication by half. The right panel of figure 3 shows the disjoint decomposition for our example, where extents are colored by the owning process id.

While computing the disjoint decomposition is fast and can be done independently in parallel doing so is worthwhile only when there is a significant amount overlap in the original in-place decomposition. There are some common scenarios where there is little or no overlap in the in-place decomposition, for example in 2D datasets or after a 3D dataset has been sliced by a plane. We use a heuristic to automatically switch between the in-place and disjoint decomposition based on an estimate of the communication and compositing cost.

An estimate of the cost to move from one screen space domain decomposition to another is given by:

$$C(S) = \sum_{i=1}^{n_p} \sum_{j=1}^{n_e(i)} \left[\sum_{p=1, p\neq i}^{n_p} \sum_{q=1}^{n_e(p)} \frac{\text{area}(S_{ij} \cap S'_{pq})}{\text{area}(V)} \right] \tag{2}$$

where S and S' are sets of screen space extents in the two decompositions, the first index selects process, and the second index selects the specific extent local to that process, n_p is the number of processes, $n_e(i)$ is the number of extents local to process i, the $\cap$ operator takes the intersection of two extents yielding an extent which contains the overlapping area, the *area*() function computes the number of pixels in an extent, V is the extent that covers the screen. The constraint, $p \neq i$ on the first inner sum excludes local extents.

In plain language, equation 2 is the ratio of off process overlapping pixels to the total screen size and is proportional to the communication and compositing work required to move between S and S' screen space domain decomposition. Set $S = S'$ and you get an estimate for the cost of compositing S in place, which is the basis for our heuristic. We examine the in-place decomposition in this way to determine when it's worth moving to the disjoint decomposition. $C(S) <= 0.3$ selects the in-place domain decomposition, otherwise the disjoint decomposition is chosen.

For the in-place decomposition, shown in the middle panel of figure 3, $C(S) = 1.8$ and our heuristic selects the disjoint decomposition. To estimate the savings of communication and compositing work we evaluate $C(S)$ with S the in-place decomposition, shown in the middle panel, and S' the disjoint decomposition, shown in the right panel of figure 3. In this case $C(S) = 0.3$, which is a factor of 6 savings in communication and compositing overhead, in addition to the efficiency gained by computing LIC only once for each pixel and the elimination of many empty pixels from the computation, compared to using the in-place decomposition.

One complication we faced when integrating our algorithm into ParaView is that IceT does not support altering the screen space domain decomposition during render-

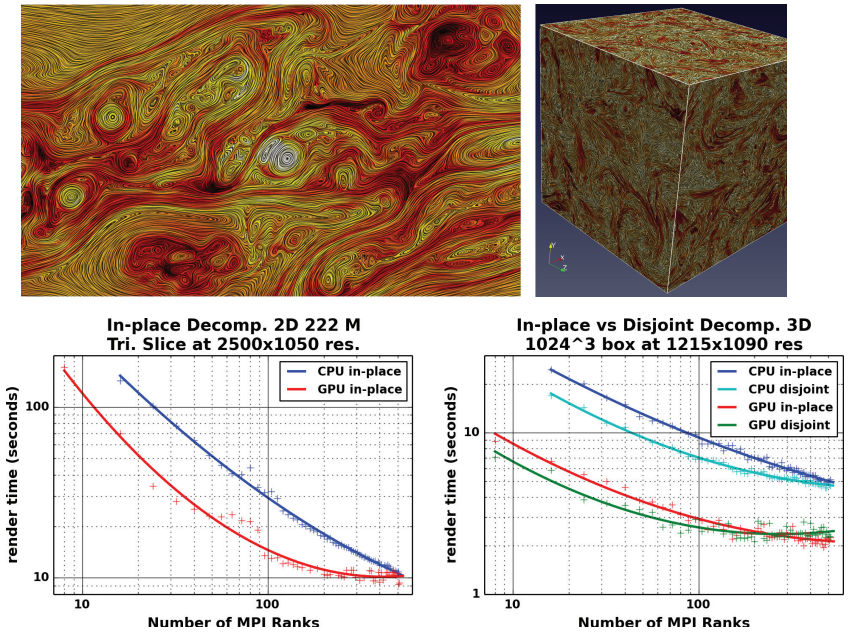

Figure 4. Strong scaling. Top row: Renderings as produced by the test configuration on 2D PIC and 3D MHD simulation of turbulent plasma. Bottom row: Rendering time as a function of number of processes with and without GPUs. Left: Runs with 2D dataset and in-place decomposition. Right: Runs on 3D dataset and in-place and disjoint decompositions.

ing. To work around this we added a scatter stage that moves the result back to the in-place decomposition. Because the regions are disjoint this is a simple copy operation implemented with point to point communication. With this additional communication the total communication costs for our algorithm are equal to or less than the in-place strategy. However, compositing costs are reduced by half, LIC computation costs are reduce by at least half on the overlapping area, and communication is spread out potentially reducing congestion.

3. Results

3.1. Scaling

We investigated the strong scaling behavior of our parallel algorithm and compare the in-place and disjoint domain decomposition strategies with and without GPUs. The runs on GPUs were made on TACC's Longhorn cluster where each compute node is equipped with 48GB RAM, two 4 core 2.5GHz Intel E5540 CPUs, two NVIDIA Quadro FX 5800 GPUs, and a QDR 4 GB/sec Infiniband network. Four processes were assigned to each node with two processes assigned to each GPU. The runs made without GPUs were made on NERSC's Cray XC30 Edison I, where each node is equipped with 64GB RAM, two 8 core Intel E5-2670 CPUs, and a Cray Aries 8 GB/sec network. On Edison Mesa 9.2.0 Gallium llvmpipe, a software implementation of OpenGL, was used. Eight processes were assigned to each node each with four rendering threads.

This was previously determined to deliver the best performance in single node benchmarks (Karimabadi et al. 2013).

The data used in our scaling study came from a 2D PIC simulation of a turbulent plasma computed on a $8192x16384$ grid (Karimabadi et al. 2013), and 3D MHD simulation of a turbulent plasma computed on a $1024x1024x1024$ grid. We set integration, shading, and contrast enhancement parameters for the runs to produce an aesthetic result on a 2560x1600 WQXGA 30" display. Note that about half the number of integration steps were needed for the 3D dataset. The renderings are shown in the top row of figure 4. Counterintuitively the 2D dataset is the more challenging from a rendering perspective. This is because only surface of the 3D dataset, comprised of $12E6$ triangles, is rendered compared with the entirety of the 2D dataset comprised of $222E6$ triangles.

The second row in figure 4 shows the strong scaling results with runs made on the 2D data on the left and 3D data on the right. Only the in-place decomposition was used with the 2D data because this decomposition is already disjoint. Both in-place and disjoint decomposition were used with the 3D data in independent runs. For both datasets the in-place decomposition scaled well to high process counts. With the 3D data, moving to the disjoint decomposition resulted in faster rendering but reduced scalability. On average moving to the disjoint decomposition produced a speed up of 22% for CPUs and 11% for GPUs. On the GPUs using the disjoint decomposition scaling ended around 200 processes, while on CPUs scaling ended around 512 processes. For the 2D dataset GPU scaling ended around 512 processes. Overall the algorithm scaled less well on GPUs which we attribute in large part to the extra step of transferring data across PCI bus during communication and in small part to the faster and fatter network pipes, and highly optimized MPI implementation on the Cray system. On average GPU's were faster than CPUs by 43% for the 2D dataset and by 62% for the 3D dataset.

3.2. Conclusions and Future Work

We have parallelized the screen space surface LIC for use in DMDPSLR frameworks, deployed our algorithm in ParaView, and investigated strong scaling behavior on two real world scientific datasets. Our algorithm eliminates redundant communication, compositing, and LIC computation by automatically moving to a disjoint screen space decomposition when appropriate. Perhaps unsurprisingly, we found that while GPUs are faster they face scalability issues at higher process counts which is likely as a consequence of the cost of moving data to and from the GPU. One direction for future work might include investigating strategies minimizing data movement to and from GPU. For example, a purely local contrast enhancement technique would reduce communication and data movement. Another idea for future work is to investigate alternative screen space domain decomposition strategies. For example the disjoint decomposition strategy we used works by eliminating unnecessary communication and computation but does nothing to ensure good load balancing. An alternative strategy could also add load balancing to ensure that each process has roughly the same amount of work. Finally, it would be worth modifying IceT to allow screen space domain decomposition to change during the render so that the scatter stage could be removed. This would further increase the speed up and improve the scalability of the disjoint decomposition.

References

Ayachit, U. 2015, The ParaView Guide: A Parallel Visualization Application (Kitware)

Bachthaler, S., Strengert, M., Weiskopf, D., & Ertl, T. 2006, Parallel texture-based vector field visualization on curved surfaces using gpu cluster computers

Cabral, B., & Leedom, L. C. 1993, in Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '93, 263

Childs, H., Brugger, E. S., Bonnell, K. S., Meredith, J. S., Miller, M., Whitlock, B. J., & Max, N. 2005, in Proceedings of IEEE Visualization 2005, 190

Karimabadi, H., Loring, B., O'Leary, P., Majumdar, A., Tatineni, M., & Geveci, B. 2013, in Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery (New York, NY, USA: ACM), XSEDE '13, 57:1

Karimabadi, H., Roytershteyn, V., Wan, M., Matthaeus, W. H., Daughton, W., Wu, P., Shay, M., Loring, B., Borovsky, J., Leonardis, E., Chapman, S. C., & Nakamura, T. K. M. 2013, Physics of Plasmas, 20

Laramee, R., Jobard, B., & Hauser, H. 2003, in Visualization, 2003. VIS 2003. IEEE, 131

Moreland, K., Kendall, W., Peterka, T., & Huang, J. 2011, in Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (New York, NY, USA: ACM), SC '11, 25:1

Okada, A., & Lane, D. 1997, in Proceedings of SPIE Vol. 3017 Visual Data Exploration and Analysis IV, 206

Stalling, D. 1997, LIC on Surfaces, Tech. rep., Konrad-Zuse-Zentrum fur Informationstechnik Berlin (ZIB)