Dataflow methods in HPC, visualisation and analysis

by

John A. Biddiscombe

Thesis

Submitted to the University of Warwick

for the degree of

Doctor of Philosophy

Warwick Manufacturing Group

August 2017

Contents

Acknowledgments

"Only the mediocre are always at their best"

Jean Giraudoux

The author would like to thank

CSCS, the Swiss National Supercomputing Centre
for the opportunity to work on the topics presented here.

Kurt Debattista and Alan Chalmers
for helping to prepare and supervise this work.

All of the co-authors of the papers presented in this thesis
for their valuable collaboration and input.

Mum and Dad
for everything you've done for me.

Olgypops
for her endless patience, cooperation and understanding
and waiting for 18 years for this thesis.

Declarations

This thesis consists of work that has been conducted in collaboration with several other authors who have given their consent to reproduce their material here.

The publication rights and copyright for each of the included papers is the property of the journal or proceedings for each of the respective articles, a copyright statement for each is included at the start of each chapter where the paper appears.

Statements have been made testifying to the contributions made by each author for each paper included in the thesis, they are included in this section.

Written statement of attribution regarding contributions to

"Time Dependent Processing in a Parallel Pipeline Architecture", IEEE Transactions on Visualization and Computer Graphics, Volume 13 Issue 6, November 2007, Pages 1376-1383, J. Biddiscombe, B. Geveci, K. Martin, K. Moreland, D. Thompson.

In support of the application to submit a PhD by Published work by

John Biddiscombe
Warwick Manufacturing Group, University of Warwick, Coventry CV4 7AL, UK
CSCS, Swiss National Supercomputing Centre, Via Trevano 131, Lugano 6900, Switzerland

We, the undersigned, hereby acknowledge that the following information is an accurate description of the contributions by each of the authors.

The paper describes the extension of the VTK pipeline model to handle time-dependent datasets and queries;

- The design of the information passing framework in the VTK library used in the paper was pre-existing prior to the work presented in the paper and forms the backbone of the VTK filter update mechanism, it was created by the authors Geveci, Moreland, Martin, Thompson as well as other collaborators not included here.
- The inclusion of new time based information keys and the design of how they interact via TemporalAlgorithms and TemporalDatasets was made jointly by all authors of the paper during the initial implementation and testing of the framework.
- The design and implementation of the time dependent filters described in the paper Temporal-Interpolator, -DataSetCache, -ShiftScale, -SnapToTimeStep, -StreamTracer, -PathTrails was the work of Biddiscombe.
- The principal editor of the written material and the creator of all the diagrams and pictures in the paper was Biddiscombe.

[redacted]

Written statement of attribution regarding contributions to

"Parallel Computational Steering for HPC Applications using In-memory HDF5 Files", IEEE Transactions on Visualization and Computer Graphics, vol. 18 Jun 2012, pp. 852-864, J. Biddiscombe, J. Soumagne, G. Oger, D. Guibert, J.G. Piccinali.

In support of the application to submit a PhD by Published work by

John Biddiscombe
CSCS, Swiss National Supercomputing Centre
Via Trevano 131
Lugano 6900
Switzerland

Warwick Manufacturing Group
University of Warwick
Coventry CV4 7AL
UK

We, the undersigned, hereby acknowledge that the following information is an accurate description of the contributions by each of the authors.

The paper describes the design and implementation of a interactive interface between the parallel visualization tool ParaView and an SPH fluid simulation code to enable computational steering and in-situ visualization. The work was carried out as part of the NextMuSE project funded by the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement 225967.

- The primary implementation of the Distributed Shared Memory (DSM) communication layer was the work of Soumagne with assistance from Biddiscombe.
- The SPH solver was the work of Oger and Guibert with assistance from Piccinali and others not mentioned here.
- The integration of the DSM into the solver was the work of Soumagne and Biddiscombe.
- Asynchronous updating mechanism between DSM/SPH/ParaView was the work of Biddiscombe with assistance from Soumagne.
- The development of the ParaView interface, automatic GUI integration, XML parsing and principal architect of the in-situ DSM visualization and steering mechanism was Biddiscombe.
- The written material was the work of Biddiscombe and Soumagne with images and benchmarks contributed by both.

[redacted]

Written statement of attribution regarding contributions to

"Practical parallel rendering of detailed neuron simulations",
EGPGV '13 Proceedings of the 13th Eurographics Symposium on Parallel Graphics and Visualization, Pages 49-56,
J. B. Hernando, J. Biddiscombe, B. Bohara, S. Eilemann, F. Schürmann.

In support of the application to submit a PhD by Published work by

John Biddiscombe
CSCS, Swiss National Supercomputing Centre
Via Trevano 131
Lugano 6900
Switzerland

Warwick Manufacturing Group
University of Warwick
Coventry CV4 7AL
UK

We, the undersigned, hereby acknowledge that the following information is an accurate description of the contributions by each of the authors.

The paper describes a comparison between the rendering capabilities and performance with neuron models of two parallel visualization tools, RTNeuron and ParaView.

- The design and implementation of the RTNeuron tool was the work of Hernando, Bohara, Eilemann, Schürmann and other contributors not included here.
- All benchmarking and testing of RTNeuron was made by Hernando, Bohara, Eilemann, Schürmann.
- Modules to load and perform parallel distribution and rendering of neuron models in ParaView were the work of Biddiscombe.
- All benchmarking of the ParaView software was the work of Biddiscombe.
- The written material was the work of all the authors with ParaView sections contributed by Biddiscombe.

[redacted]

Written statement of attribution regarding contributions to

"High-Performance Mesh Partitioning and Ghost Cell Generation for Visualization Software", EGPGV '16 Proceedings of the 16th Eurographics Symposium on Parallel Graphics and Visualization, J. Biddiscombe.

In support of the application to submit a PhD by Published work by

John Biddiscombe

Warwick Manufacturing Group	CSCS, Swiss National Supercomputing Centre
University of Warwick	Via Trevano 131
Coventry CV4 7AL	Lugano 6900
UK	Switzerland

We, the undersigned, hereby acknowledge that the following information is an accurate description of the contributions by each of the authors.

The paper describes the design of the zero-copy serialization layer in the HPX runtime;

- The mesh partitioning layer and ghost cell generation for VTK was the work of Biddiscombe.
- The author and creator of all the diagrams and pictures in the paper was Biddiscombe.

[redacted]

Written statement of attribution regarding contributions to

"Zero copy serialization using RMA in the HPX distributed task-based runtime",
14th International Conference on Applied Computing 2017,
J. Biddiscombe, T. Heller, A. Bikineev, H. Kaiser.

In support of the application to submit a PhD by Published work by

John Biddiscombe
Warwick Manufacturing Group
University of Warwick
Coventry CV4 7AL
UK

CSCS, Swiss National Supercomputing Centre
Via Trevano 131
Lugano 6900
Switzerland

We, the undersigned, hereby acknowledge that the following information is an accurate description of the contributions by each of the authors.

The paper describes the design of the zero-copy serialization layer in the HPX runtime;

- The serialization layer in HPX and chunk based archive design was the work of Kaiser, Bikineev and Heller and others not included here.
- The extension of the serialization benchmarking application to include HPX and the generation of the serialization benchmark figures was the work of Bikineev.
- The extension of the archive to support RMA chunks was the work of Biddiscombe.
- The one sided zero-copy network implementation using libfabrics was the work of Biddiscombe, with help from Heller.
- The implementation of rma_object and rma_vector was the work of Biddiscombe.
- The principal author and editor and the creator of all the diagrams and pictures in the paper was Biddiscombe.

[redacted]

Abstract

The processing power available to scientists and engineers using supercomputers over the last few decades has grown exponentially, permitting significantly more sophisticated simulations, and as a consequence, generating proportionally larger output datasets. This change has taken place in tandem with a gradual shift in the design and implementation of simulation and post-processing software, with a shift from simulation as a first step and visualisation/analysis as a second, towards in-situ on the fly methods that provide immediate visual feedback, place less strain on file-systems and reduce overall data-movement and copying. Concurrently, processor speed increases have dramatically slowed and multi and many-core architectures have instead become the norm for virtually all High Performance computing (HPC) machines. This in turn has led to a shift away from the traditional distributed *one rank per node* model, to *one rank per process*, using multiple processes per multicore node, and then back towards *one rank per node* again, using distributed and multi-threaded frameworks combined.

This thesis consists of a series of publications that demonstrate how software design for analysis and visualisation has tracked these architectural changes and pushed the boundaries of HPC visualisation using dataflow techniques in distributed environments. The first publication shows how support for the time dimension in parallel pipelines can be implemented, demonstrating how information flow within an application can be leveraged to optimise performance and add features such as analysis of time-dependent flows and comparison of datasets at different timesteps. A method of integrating dataflow pipelines with in-situ visualisation is subsequently presented, using asynchronous coupling of user driven GUI controls and a live simulation running on a supercomputer. The loose coupling of analysis and simulation allows for reduced IO, immediate feedback and the ability to change simulation parameters on the fly.

A significant drawback of parallel pipelines is the inefficiency caused by improper load-balancing, particularly during interactive analysis where the user may select between different features of interest, this problem is addressed in the fourth publication by integrating a high performance partitioning library into the visualization pipeline and extending the information flow up and down the pipeline to support it. This extension is demonstrated in the third publication (published earlier) on massive meshes with extremely high complexity and shows that general purpose visualization tools such as ParaView can be made to compete with bespoke software written for a dedicated task.

The future of software running on many-core architectures will involve task-based runtimes, with dynamic load-balancing, asynchronous execution based on dataflow graphs, work stealing and concurrent data sharing between simulation and analysis. The final paper of this thesis presents an optimisation for one such runtime, in support of these future HPC applications.

Chapter 1

Introduction

Visualisation and analysis go hand in hand with simulation; without graphs, images and videos made from the results generated by scientific simulations there would be significantly less discovery and insight in scientific computing. Unfortunately, generation of images from scientific data is not always a straightforward task, particularly so as the size of datasets generated by scientists has been growing consistently over the years, in line with the growing compute power that has followed Moore's law and in turn driven the IT and Big Data revolution that we are living through.

The problems associated with large dataset visualisation are a combination of those that occur in distributed computing with those associated with parallel rendering and stem from the following underlying causes:

- When datasets exceed the size of memory on a single workstation or the resources required to generate a pleasing image exceed the capabilities of a single node, it becomes necessary to process the data in parallel using multiple nodes in a cluster or supercomputer.
- Processing data in parallel requires communication and synchronization between nodes that significantly complicates software.
- As data becomes larger, it (generally) becomes more difficult to find regions of interest because the ratio of *interesting* to *average* numbers (or features) in the data becomes smaller.
- To make interesting data more visible and stand out from the sea of more mundane data, complex mapping or transformations may be required that require special treatment when run in a distributed parallel code.

- Increasing data sizes place more strain on filesystems that are impacted both by the simulation, and again by the visualisation during later post processing.

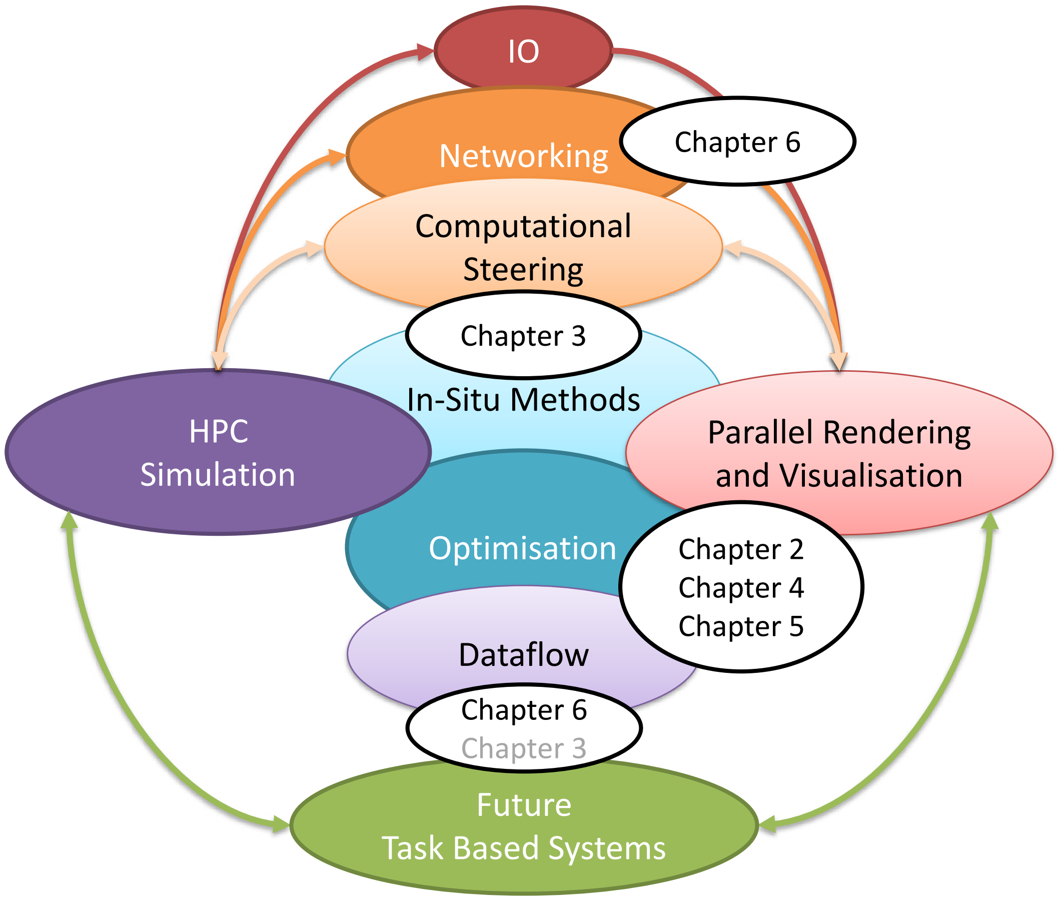


Figure 1.1: An illustration of how the main themes of the chapters of this thesis fit into the broader picture of simulation, visualisation and analysis. (It is not possible to confine each chapter to a single position since they all deal with optimisation and dataflow in some capacity).

These problem areas can be collected into broad categories as shown in figure 1.1, which illustrates how the chapters of this thesis overlap with those areas of research.

- **Networking** represents the major difference between a multi-node distributed application and a single node one; all communication between processes makes use of the network. In the area of **Parallel Rendering and Visualisation**, distributed pipelines (Chapter 2), image compositing (Chapter 4), load balancing (Chapter 5) all rely on high performance networking. Optimisation of the network layer itself is the subject of Chapter 6.

- **IO** has become a major bottleneck in the simulation-analysis workflow; huge datasets, saved to disk (then reloaded later), may add minutes or hours to compute runs. **In-Situ** analysis (Chapter 3) is one of the solutions to the IO problem, by analysing data at the point of creation, extracting useful information from it and only saving that which is necessary the total time spent on scientific discovery is dramatically reduced.

- **Computational steering** (Chapter 3) takes the principle of in-situ analysis one step further by allowing changes to the simulation to be made whilst it is still running and providing immediate feedback to the user.

- **Dataflow** represents the algorithmic building blocks and the connections between them, that are used when turning one piece of data into another, whether that data be initial boundary conditions, geometric/topological descriptions or any other methodological parameters. Dataflow represents the single binding theme of this thesis where all work intersects and it is the **Optimisation** of those building blocks and how they connect that each chapter explores.

- **Future** computer systems will have more cores, more accelerators, and be built with more heterogeneous components that in turn require more complex synchronization, thread control and scheduling between algorithmic dataflow blocks that subdivide work into **Tasks** that execute asynchronously. The optimisation of task communication over the network is the subject of Chapter 6.

1.1 Dataflow

Dataflow techniques have emerged as the dominant design paradigm for visualisation software with the two best known distributed parallel applications, VisIt and ParaView, being built upon the Visualization Toolkit (VTK, Schroeder et al. [1996]), a dataflow based visualisation library. VTK consists of hundreds of sources (readers, data generators), filters for processing datasets (slicing, contouring etc.) and sinks (renderers, writers) that consume data, these make up a huge pool of modular algorithms and connectible components that allow data of almost any kind to be manipulated and transformed from one type to another or rendered as images or graphs. In the wider field of data science, other technologies that make use of dataflow, such as MapReduce (Dean and Ghemawat [2008]), Apache Spark (Zaharia et al. [2016]) and more recently Google's Cloud Dataflow (Akidau et al. [2015]) have been widely adopted for large scale data processing due to their flexibility and applicability to a wide range of problems. In the field of AI, libraries such as TensorFlow (Abadi et al. [2016]) are built around the concept of dataflow graphs in exactly the same way and are revolutionising the software landscape and placing powerful learning tools in the hands of the masses.

At the heart of the dataflow principle is the idea that many small tasks that perform a limited or well defined function, may be chained together and combined to solve more complex problems (see Johnston et al. [2004] for a good introduction). One of the requirements (and great strengths) of this approach is that it encourages the adoption of a manageably small number of well defined data types to represent the intermediate states of the computation (graph edges) so that pipelines can be constructed from reusable code blocks (graph nodes) and each new algorithm or method added, increases the problem space that can be explored in a combinatorial way. Modular design of software is not new or revolutionary and the principles behind dataflow methods originate from the more fundamental realm of functional programming, where every operation (cf. filter) is a transformation of some input data/arguments and the resulting new data is produced as an output. Functional programming itself has seen a revival in popularity over recent years, in large part due to the changes in microprocessor architectures towards more cores per chip, which in turn leads to stronger thread safety requirements in programs. Functional programs should ideally have no external state – no global mutable variables – one of the principal requirements of robust thread-safe programming, however the adoption of thread-safe programming techniques in large libraries such as VTK has been slow due to the size and complexity of those libraries; their design and much of their implementation was already in place before the need for thread safety became necessary.

When VTK was first released in the 1990's, dual core processors were not available (IBM's Power-4 processor was the first dual-core general purpose CPU and was released in 2001, AMD and Intel produced their first x86 compatible dual-core processors in 2004 and 2005 respectively), dual socket machines were not commonplace and so multi-threaded programming was not widely used outside of the operating systems that hosted the users code and (in hardware form) in the graphics chips that did the work of rendering their images. Initial work on laying the foundations of multi-threading and parallel distributed operation in VTK were introduced in (Law et al. [1999] and Ahrens et al. [2000]) and it is in these developments that the concept of *information flow* into the VTK dataflow pipeline were first introduced –

information in this context may be considered as *meta-data* about the data being processed, the following section describes why it is useful and needed.

1.1.1 Information in Dataflow Systems

The concept of dataflow is straightforward, data is taken in by some filter or function, modified in some way and exported, where it is then consumed by another filter and the process repeats, with fan-out and fan-in of data paths producing a network of connections that form a graph. Generally (at least in visualisation), the graphs are acyclic and are referred to as Directed Acyclic Graphs (DAGs). In a visualisation pipeline, the sources that form the initial nodes of the graph are readers (or in-situ data generators) and the sinks that terminate the graph are renderers (or IO writers). Cyclic feedback loops within the graph are used frequently in AI systems such as TensorFlow, as they allow for concepts such as memory and feedback to be introduced, but in more traditional data-processing pipelines they introduce ambiguity and can trigger unwanted re-execution of the algorithms in the graph nodes if not treated specially.

The problem with a simple graph is that all data must be transmitted along the graph edges connecting the nodes, and the *type* of that data must conform to what is expected at the inputs to that node. A contour filter (for example) may expect a regular grid such as an image as input, but will generate an unstructured grid/mesh as output. A second filter expecting a regular grid cannot accept the unstructured data since the cells of the mesh in most systems will be represented by a different *type* than the cells in the image – an image requires only an x and y dimensions, origin and spacing between pixels, whereas the mesh requires explicit connectivity array to map vertices onto coordinates – very different definitions. A renderer may turn the mesh geometry back into a 2D image and return the type back to a regular grid once more – such transformations are common with filter pipelines. Within a powerful high-level programming language such as C++, the datatypes that represent the geometric *types* may be part of a class hierarchy that allows one to construct higher abstractions that hide some of these type transformations or provide automatic conversion from one type to another, removing some restrictions from the composability of the nodes within the graph and therefore making their construction more straightforward. All these considerations are taken into account in the design of a library such as VTK, but problems arise when the representation of *parameters* are included, in particular, *conditional* parameters.

To explain why parameters cause problems, consider figure 1.2 that shows a simple pipeline, the user may control parameters of each of the filters, or the

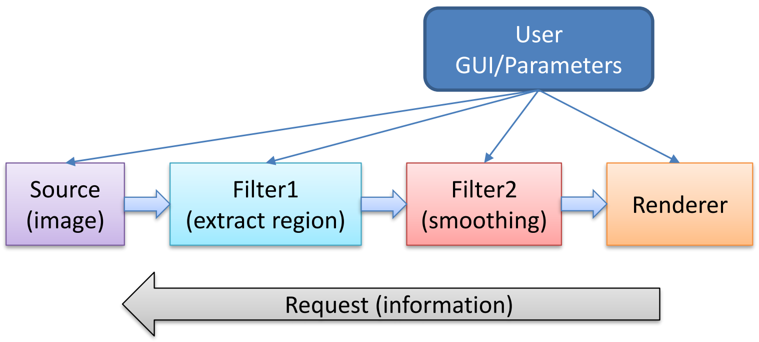


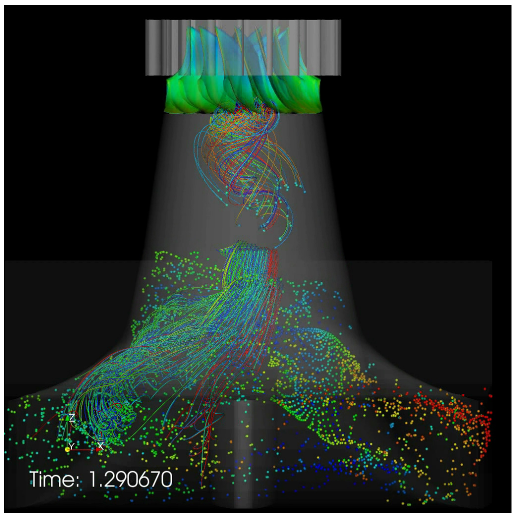
Figure 1.2: Each of the source nodes, or filters/renderers may have a large number of parameters, which may change the input or output requirements, or even data types. When the user interacts with a filter, information passing enables filters to modify themselves or their behaviour accordingly.

renderer. Data (an image in this example) is loaded initially and passed to an extract region filter that in turn crops the data and outputs a new image of smaller size, this is passed to a second filter that applies a smoothing operation in the form of a simple stencil. If the user sets the region of interest (ROI) on the extraction filter to some small subregion of the whole image – should the source reader export the whole image, or just the region that is needed for the ROI? Clearly the source reader could be more efficient if it only read the data that was actually needed further down the pipeline. The difficulty is that one must now pass parameters that belong to the extraction filter to the reader so that it can modify its output accordingly. The scenario can be made more complicated by including the smoothing filter – the user may select a smoothing option that applies a stencil or convolution operation requiring data from N nearest neighbours for each pixel – the ROI must be inflated by one or more pixels and the exported source region must be expanded accordingly. The user might even use another parameter to disable the visibility of the smoothed image in the renderer and it is now no longer necessary to read or process any data at all. The number of possible parameter changes that affect the dataflow balloons as more and more filters are added and it is clearly not possible to pass all of them to all filters and have them handle them correctly – especially if one considers that all parameters may have different types themselves (scalars, vectors or other structures). The solution adopted in VTK and it's parallel distributed implementation ParaView (Ahrens et al. [2005]) is to use *information keys* that are special messages that flow up and downstream, complementing the main dataflow and permitting a downstream filter to affect the behaviour of one upstream one by passing information/requirements about regions, pieces, ranges etc. and each filter may update/modify the information (even adding new information keys and values) depending upon its own needs before the main data update stage. Information flow is initiated via a *request* that is generated by the *executors* responsible for updating the pipeline when new data is required (such as when the user refreshes the screen to update the display).

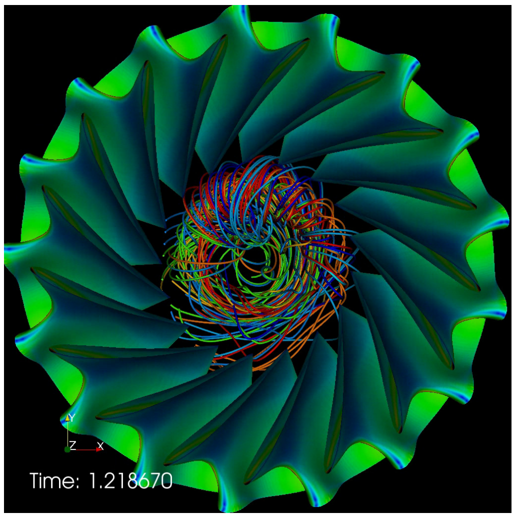
The solution implemented in VisIt (Childs et al. [2005]) uses a *contract* structure that holds fields for any dataset attributes or values that might be useful to other filters and each one may update the values in the contract prior to execution of the pipeline. The contract system is not as trivially generalizable as the one in ParaView that allows any key/value pair to be added to the information by any filter (or graph node) but it permits essentially the same kind of modifications to take place in the pipeline. It is important to note that the addition of extra messages flowing (sometimes bidirectionally) between nodes of the graph does not change the essential structure of the graphs into cyclic or undirected graphs since the main flow of data is unchanged – however, the communication between nodes adds tremendous flexibility and give them the opportunity to perform optimisations (data reduction for example) based on information supplied from others.

Chapter 2 presents the paper "Time Dependent Processing in a Parallel Pipeline Architecture" (Biddiscombe et al. [2007]) that explores this topic in more detail and uses the technique to extend the capabilities of ParaView to handle time dependent datasets in a consistent manner. The developments outlined in this paper allowed the creation of a series of filters that are still used today as part of the backbone of the pipeline in ParaView. The ability to modify time requests, create branches of pipelines with different timesteps and cache data between times is used in the comparative visualisation features as well as the animation controls provided by the ParaView front-end GUI. The implementation details of these features are generally hidden from the user, however some of the more visible developments are used actively – the best known of which is probably the vector field particle stream tracer that is used worldwide by researchers on a daily basis for visualisation of time dependent flows in areas of engineering and science from CFD to astrophysics.

The first industrial use of the time dependent pathline filter was in the visualisation of flows in a Francis turbine to identify vorticity and regions of cavitation (which damages turbine blades). The images of figure 1.3 show flow and vortex formation in a the turbine; they were particularly difficult to generate because the dataset consisted of over 50 individual high resolution mesh blocks representing different parts of the turbine, of which 34 were rotating relative to the frame of the turbine. Handing this case required special treatment of particles that passed from



(a) Turbulent flow in Francis turbine



(b) Close-up of vortex

Figure 1.3: Snapshots of pathlines generated in time dependent flow fields in a Francis turbine. (a) The turbulence below the main turbine blades causes uneven outflow that in turn reduces the efficiency of the turbine. Some particles are seeded midway through the flow to better show the asymmetrical flow. (b) A stable vortex forming right below the main turbine blades.

a stationary to a rotating block during an integration step and could not be done prior to the developments of Chapter 2. No other off the shelf software available at the time was able to produce images and videos of this kind from the datasets used.

The particle tracer was also used to create visualisations of flow in cerebrospinal fluid (CSF) to accompany the co-authored paper (Gupta et al. [2010]) which contains many images created using the new ParaView capabilities. Figure 1.4 shows a still image of the flow of CSF around the brain that when animated reveals the periodic motion of the fluid, synchronized to the heart beat, that transports chemicals around the brain and can be used in the modelling of drug delivery.

Numerous other time based filters have been developed since the time based pipeline work was completed, one example is the implementation of a custom interpolation filter for SPH (smoothed particle hydrodynamics) particles that was used in the extraction of vortex core-lines for another co-authored paper (Schindler et al. [2009]). The use of cubic interpolation that includes the particle velocities at two time steps as well as their positions produces a smoothly differentiable trajectory that leads to better convergence of the vortex core-line calculation developed in the paper. The pipeline model makes it easy to create custom filters that can be inserted into a visualisation pipeline to perform these kind of non-standard, appli-

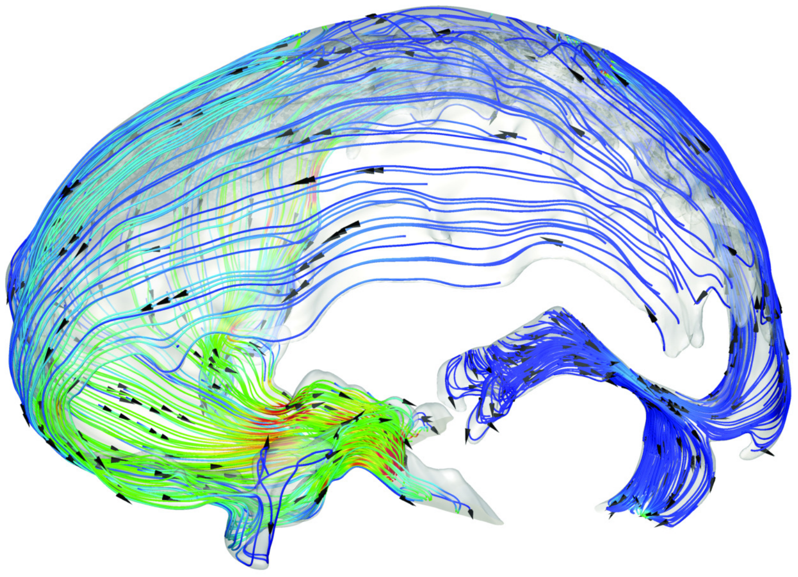


Figure 1.4: Flow visualisation of cerebrospinal fluid between the skull and brain and in the subarachnoid space. The beating heart causes pressure changes in blood flow and expansion/contraction of the brain tissue that triggers circulation of the fluid, transporting metabolites and hormones in the process.

cation specific operations, figure 1.5 shows the striking difference between the linear and hermitian interpolation scheme used.

1.2 In-Situ Visualisation and Computational Steering

The size of computer simulations in the field of HPC has been, and still is, continually growing as processors become more powerful and clusters/supercomputers increase in capabilities correspondingly. Managing the torrent of data produced by simulations has become one of the major challenges in the simulation community. In-situ analysis is the term used to refer to the generation of output statistics, plots or images that are derived from simulation results using the tools conventionally reserved for post processing data. The central idea being to reduce IO from simulation to file system, and then back again from file system to visualisation, as well as increasing the frequency at which analysis can be performed, since IO is frequently performed after N time steps rather than at every one to reduce IO needs. Over time, the approaches used for in-situ analysis have become known by the terms

- Loosely coupled, denoting that the simulation generates data that is sent to

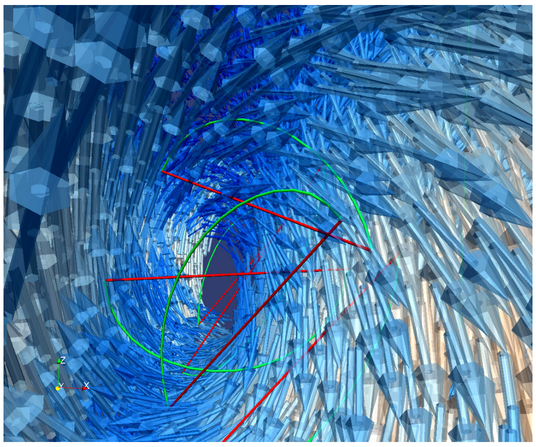


Figure 1.5: Illustration of cubic interpolation of particle paths. Arrows indicate positions and velocities of all particles and show a region of air under a wave rolling over. Red lines mark linearly interpolated paths of some of the particles, while green curves indicate the corresponding cubically interpolated paths. (Image © IEEE. Reprinted, with permission)

separate nodes reserved for analysis to be further processed.

- Tightly coupled, where data remains on the same nodes on which it has been generated and is then processed in a second stage by analysis code.
- Hybrid coupled, some data reduction or initial processing may take place on compute nodes and the remaining data is then forwarded on to others for further processing.

Each strategy has its own advantages and disadvantages;

- Loose coupling places a minimal burden on the simulation which can continue unaffected – but it forces the user to make a possibly expensive copy of the data on analysis nodes which must be allocated in addition to the compute nodes.
- Tightly coupled methods may require the simulation to stop whilst post processing takes place and may cause excessive memory requirement on the compute nodes and impact performance through contention for system resources. This problem is particularly notable if the analysis code does not scale as well as the original simulation code, resulting in resources being wasted.
- Hybrid approaches may take the best, and possibly the worst of both worlds and are the most complex to implement and deploy.

The first major work in this area for the HPC community was the loose coupling framework DataStager (Abbasi et al. [2009]), it was built upon a number of other libraries including the Adaptable IO Services (ADIOS, Lofstead et al. [2008]) that were not used widely outside of the US national labs where they were developed. A much more widely adopted parallel IO library is the Hierarchical Data Format, HDF5 (The HDF Group [2000-2017]) and this was used as the basis for a loosely coupled interface between simulation and ParaView (Biddiscombe et al. [2011]), this paper was later extended and published as Chapter 3, "Parallel Computational Steering for HPC Applications using HDF5 Files in Distributed Shared Memory" (referred to as DSM henceforth) (Biddiscombe et al. [2012]) – the paper introduces a DSM based approach to reroute file IO into memory instead of disk and expose the memory as a file to the application. In the same year as the initial paper, the VisIt community released a tightly coupled in-situ library *libsim* (Whitlock et al. [2011]) and a few year later a tightly coupled in-situ ParaView library was released called *Catalyst* (Ayachit et al. [2015]).

The work of Chapter 3 differs from libsim and Catalyst and is important for several reasons:

- The ParaView/DSM interface is built on top of a custom HDF5 file driver implying that any simulation that uses the HDF5 IO library can be integrated into the in-situ visualisation framework without requiring any modifications to the code – only relinking to a modified HDF5 lib is needed.
- Since only a linking step is needed, Fortran, C, C++ based simulation codes can all be accommodated equally well (other languages could be supported but have not been tested).
- The HDF5 file exposed by the DSM file driver (H5FDDSM) behaves exactly like a high speed file on a parallel filesystem and so can be read from or written to by either side of the communication link – this permits computational steering as well as visualisation since commands and data can be sent back from the ParaView GUI and read by the application.
- Custom GUI interfaces to control steerable objects or define visualisable ones can be built for the simulation using a simple XML syntax.

The dataflow theme continues in the computational steering work of Chapter 3, when outputs from the simulation are connected to pipelines in ParaView and these are used to generate new datasets that are returned back to the simulation for use in subsequent time steps (direct coupling of solvers is also possible in this way).

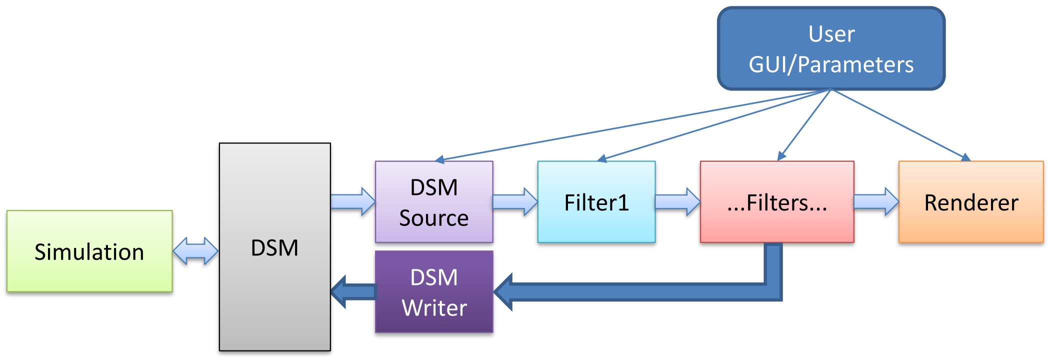


Figure 1.6: The DSM is a bi-directional module in the pipeline, it permits dataflow from simulation to user, and also allows steering commands and data to be returned.

Figure 1.6 shows how the pipeline of figure 1.2 is modified by the presence of the DSM, forming a bridge between the simulation and the conventional visualisation.

ParaView has been used as an interface for a user to control the simulation by taking outputs from the simulation, combining them with user parameters from controls, then generating new geometries that are fed back to the simulation to steer it, experiments involved the steering of boats, control of wave generator geometries and the design of turbine blades and buckets – principally using SPH solvers that are well suited to handling adaptive/dynamic meshes. In figure 1.7 is an example of the steering framework being used to control a pelton turbine simulation. The design of the bucket shape is critical to minimize the splash from the water jet rebounding and colliding with the incoming jet as this perturbs the shape and reduces momentum. This plot shows the change in velocity and torque from the turbine as the water flow is increased.

Several difficulties arose when controlling simulations in this way – the main one was that the time steps output in simulation time are typically tiny, of the order of milli or microseconds and are generated in seconds or tens of seconds of user time – this makes it very difficult for interactive (human operated) controls to be used to directly steer a simulation and instead it was found that connecting scripted python routines to outputs and animating variables using the time dependent controls discussed previously gave more consistent and predicable behaviour.

1.2.1 MPI – one rank per node, one rank per process

A second problem that was found with the DSM implementation was not with controlling simulations but with the complexity of the implementation itself which used MPI for all communication – between two applications that were themselves

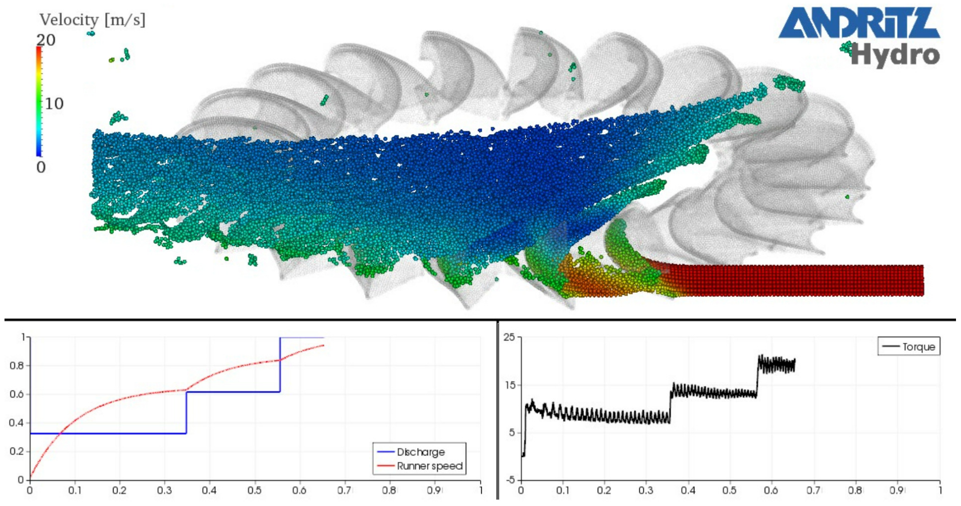
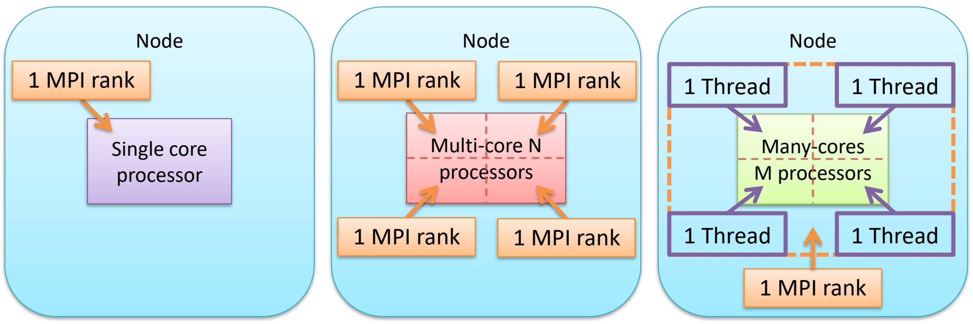


Figure 1.7: A snapshot of the steering system in use whilst monitoring and controlling a simulation of a pelton turbine. The user increased the water discharge in steps and measured the velocity of the turbine and torque produced.

using MPI communication internally. Allowing an entirely asynchronous shared memory file with parallel access that can be opened and closed arbitrarily by either simulation or analysis, required a complex (parallel) locking mechanism to ensure that only one side of the link could modify the file contents at a time and a complex message scheduling algorithm to distribute pieces of the file in an MxN fashion (since the number of simulation nodes might not match the number of analysis nodes and random access of the file is permitted and enforced by using random or block cyclic access). This in turn required a multi-threaded design to allow efficient handling of many messages coming from a large number of nodes in arbitrary ordering.

The latest MPI-3 standard (MPI Forum [2012]) emphasises and encourages the use of asynchronous non-blocking communications, one-sided transfers using remote memory access (RMA) and improvements to multi-threaded performance in applications where *any* thread may perform communication. In MPI terminology, each rank of an application is a single process, which may consist of one or more threads of execution. When multiple cores are available on a node, it is possible to run multiple MPI ranks on that node where each rank is bound to a core, and the memory used by that core is isolated from the memory used by other cores/ranks. This has the effect of improving performance of applications due to good cache reuse and minimized cross memory-bus traffic when multiple sockets are present. As the number of cores on a chip is steadily increasing and memory hierarchies are



(a) 1 single-threaded rank per node (b) N single-threaded ranks per node (c) 1 multi-threaded rank per node

Figure 1.8: Illustration of the relationship between nodes, ranks, threads and cores. (a) A single core node with a single MPI rank, the simplest configuration and the dominant usage mode for older processors. (b) A multicore node may have 1 rank assigned to each core, each rank represents a separate process with an independent memory space. (c) A multicore node may be assigned a single rank or process with the rank using multi-threading in a single address space shared between all threads.

becoming more complex, the trend in software design is moving away from many ranks on a node towards a single MPI rank/process consisting of many threads of execution on each node. This tends to increase the complexity of the software due to thread safety issues, but allows a much easier sharing of data between cores on the same node, where previously they would have been different ranks and had to communicate using the MPI interface. Figure 1.8 gives a simple illustration of the evolution from a single rank per node, to many and then back to one again.

ParaView relies on the single rank per process model - each time a user instantiates a filter pipeline, N copies of the pipeline are created, whether they share a node or are distributed among nodes. Each core on a node is a separate rank in an MPI program and communicates with other ranks using MPI. A user may create a single ParaView server on a multicore node and then a use multi-threaded filter on that node, but so far only a small fraction of the available filters can benefit from this. To solve this problem, a new implementation of the VTK filter design has been created, VTK-m (Moreland et al. [2016]) that inherits most of its design from three earlier attempts at creating multi-threaded and/or GPU enabled filters, EAVL (Meredith et al. [2012]), DAX (Moreland et al. [2011]) and PISTON (Lo et al. [2012]). The design of VTK-m is centred around the implementation of Data Parallel Primitives (DPP) the use of parallel (in the sense of threads or vectorized instructions) implementations of algorithms to decompose visualisation tasks into high parallelisable building blocks along the original design of VTK. The discussion of this topic will be revisited in the final part of this introduction and Chapter 6. The next section looks instead at another of the problems that arise during interactive visualisation.

1.3 Load-balancing, rendering and information

When interacting with large datasets, a common operation is to extract regions by dragging a selection area on screen, or selecting one or more blocks of data from many by some attribute or threshold. When this happens the data that might have originally been well load-balanced across processes, may become wildly imbalanced with some processes holding no data and others containing $1/N^{\text{th}}$ of the data (as originally allocated). In many cases (for example, when the number of analysis nodes is small anyway and some wasted CPU cycles can be ignored), an imbalance is of little consequence, however there are many occasions when maintaining load-balance for either analysis or rendering is important. The work of Chapter 4 and Chapter 5 was motivated by these requirements. Chapter 4 presents "Practical parallel rendering of detailed neuron simulations" that used an early implementation of the work described in Chapter 5 "High-Performance Mesh Partitioning and Ghost Cell Generation for Visualisation Software" most significantly, the load-balancing of very large meshes to improve interactive visualisation particularly when transparency of those meshes made sort-last compositing the most favourable rendering mode (the full text of Chapter 4 discusses the trade-offs between sort-first and sort-last for transparency).

The question in need of an answer in the paper of Chapter 4 was, "could a general purpose visualisation tool (eg. ParaView), designed to work with almost any data and many workflows, compete with a custom built, single-purpose visualisation tool?" (in this case RTNeuron, Hernando et al. [2008], and Hernando [2011]): with the secondary question being "how much effort would be required to do it?". The answer turned out to be "yes", the ParaView implementation of neuron rendering was able to perform as well as RTNeuron for most cases, though RTNeuron had many options for neuron visualisation that were not available to ParaView (they can be implemented if so desired).

In order to achieve good results with ParaView it was necessary to replace the data distribution filter with a higher performance implementation based on the Zoltan (Boman et al. [2007]) library from the Trilinos project. This produced a huge improvement in load-balancing, but rendering still performed poorly due to bad communication between the load-balancing modules and the rendering ones.

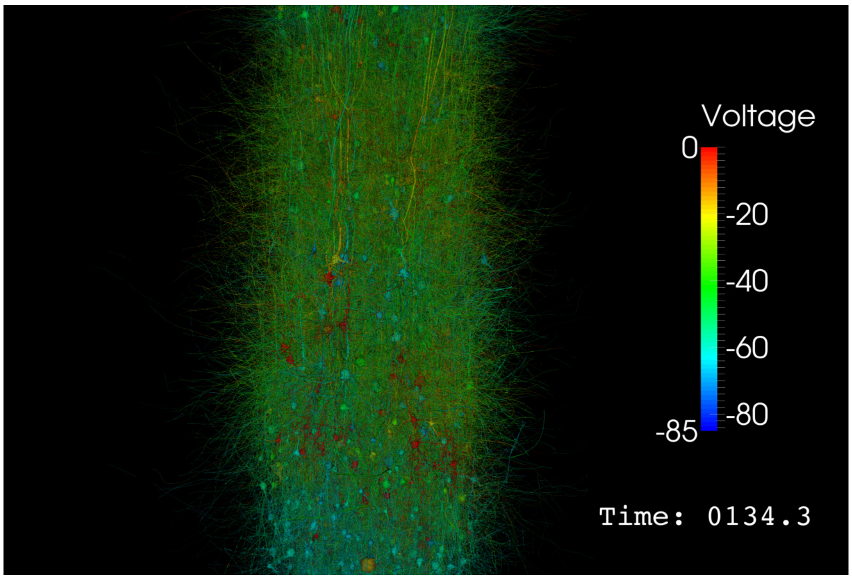


Figure 1.9: 5K neuron circuit rendered with per vertex transparency showing electrical activity (in millivolts) - regions that are not active are transparent, allowing active regions to be seen more easily through the dense mesh.

The solution to this problem was to make use of information flow in the pipeline to pass the geometric bounding boxes of individual process pieces of data down the rendering pipeline and then modify the rendering layer to use the information to order the compositing of pieces and give correct transparent blending. Transparent rendering of surfaces requires ordered (back-to-front usually) rendering, or depth peeling of data to give the correct results, and this ordering includes the compositing of individual images from nodes that have been rendered correctly individually.

With load balancing and compositing optimized, there remained only the actual geometry rendering that required performance improvements – to solve this problem, the data parallel PISTON library (mentioned in the previous section) was used to create a data-parallel sort and render filter that could run on the GPU instead of the CPU and then pass the generated images to the network for parallel compositing. At the same time, the rendering filter was extended to support 4 channel RGBA values so that transparency could be added on a per vertex basis – as required by the neuron visualisation – previously, ParaView supported only RGB values per vertex with a single alpha transparency value per object. With these 4 major enhancements (load-balancing, compositing, rendering, transparency), the results were comparable to RTNeuron despite the work taking around 6 months of development, whereas RTNeuron had been in continuous development for over 5 years at the time.

1.3.1 Particle data

The development of the Zoltan based load-balancing filter was driven partly by the desire to handle large neuron meshes, and also by the need to handle very large particle based datasets. One problem was that there was no capability in ParaView to generate ghost cells for particle data which was necessary when analysing huge (tens of billions of cells) datasets on distributed processes. The data distribution filter in ParaView can generate ghost cells for unstructured meshes, but not for particle data, and this is essential when one wishes to interpolate between points to compute resampled density, mass or other properties between particles when those particles overlap process boundaries. Chapter 5 presents work that took the initial implementation used for neuron rendering and made it more robust, more feature complete and tightly integrated into the ParaView framework.

Some of the analysis operations on particle data (eg. SPH) requires finding N nearest neighbours and computing a weighted sum of their contributions which can be a time consuming and compute intensive task – the ability to extract regions of interest from load-balanced data, then perform a second load-balancing step to redistribute the new data where it can be processed in parallel by all nodes in the job (and not just those on which the interesting data was first loaded) dramatically increases the speed of analysis.

1.4 Task based programming

The connection between the final paper in this collection, entitled "Zero Copy Serialization using RMA in the Distributed Task-Based HPX Runtime" (appearing as Chapter 6) and the previous papers, may not seem obvious since the other papers deal principally with visualisation enhancements and performance improvements to ParaView. To understand the connection, consider the new VTK library, VTK-m that is built upon data-parallel primitives: these primitives consist of basic algorithms such as sort, copy, inclusive scan, exclusive scan, reduce by key, lowerbound, upperbound and several others. The majority of these algorithms have now been standardised as part of the C++17 extensions for parallelism [The C++ Standards Committee, 2017, §algorithms.parallel] and so become part of the core C++ language (under the namespace `std::parallel`) and can be invoked in standard code to run algorithms on multiple cores. The remaining algorithms in VTK-m that have not been standardised may be constructed from those that have. Also standardised as part of the C++11 language is the asynchronous call `std::async` [The C++ Standards Committee, 2011, §futures.async], that allows a function to be invoked on a separate thread and return a `std::future` that is a thread synchronisation primitive for a result that has not yet been generated – enabling *tasks* to be spawned that perform computations in the background, on separate threads, whilst the function that launched it continues with other work.

1.4.1 Dataflow through futurization

Limitations in the C++11 definition of `future` make it difficult to use for task based programming directly, however a number of additions have been proposed for C++20 and beyond that extend the features of `future`s significantly by using *continuations*; `future`s are monadic data structures, they can be composed together and used to chain operations in the following manner:

```
future_2 = future_1.then(new_function, arg1, arg2, ...);
```

This permits a style of programming known as Continuation Passing Style (CPS – Appel and Jim [1989]) where futures may be used as inputs to other functions that themselves return futures. The beauty of this system is that the continuations attached to `future`s via the `future.then` function are not executed until the first `future` has completed, making non-blocking asynchronous multi-threaded code straightforward and simple to write with an easy to learn syntax. A number of other functions such as `future.when`, `future.when_all`, `future.when_any`, and a `shared_future`, make it possible to build up arbitrary dataflow graphs from the futures. A `shared_future` may be used to connect one task to several others, and the `future.when(...)` constructs allow the joining of several futures into one so that only when all required tasks have completed can the next one begin. CPS ensures that tasks that have data dependencies do not start executing until all their dependencies are satisfied and this approach has been shown to reduce unnecessary waiting and avoid latency (Syme et al. [2011]).

1.4.2 HPX

Whilst the extensions to C++ `future`s to enable CPS are not likely to be available until the time-frame of C++20, the HPX runtime system for parallelism and concurrency (Kaiser et al. [2017]) has already implemented them, as well as the extensions to `std::parallel` algorithms, so they may be used today. HPX goes further than the current C++ proposals, by extending the asynchronous API to distributed operations using an Active Global Address Space (AGAS) to reference objects on remote nodes (Kaiser et al. [2015]). AGAS works as a kind of distributed key-value store holding Ids to objects across a global address space that permits tasks to be launched on remote nodes and return `future` results from those nodes. Objects may be held locally as normal C++ data structures, or may be registered (using a unique name) with AGAS and then accessed remotely via the AGAS Ids. This means that code designed to run on a single node using multiple threads may easily be extended to run on multiple nodes and multiple threads using the same API.

The VTK-m filter framework is designed in such a way that multiple implementations may be created as backends on which to execute code using a *Device Adaptor* abstraction that hides implementation details of the threading layer from the filter designer. Versions already exist for GPU execution using CUDA (Nickolls et al. [2008]) and on the CPU using a serial implementation and multi-threaded using Thread Building Blocks (TBB – Reinders [2007]). This author has implemented an HPX backend that implements the *Device Adaptor* using the `hpx::` versions of the `std::parallel` algorithms and permits all visualisation filters to be executed using the HPX runtime. However, developing high performance asynchronous distributed VTK-m style filters in HPX that can pass datasets and pieces around, requires a high performance network layer within HPX and this is what is presented in the paper of Chapter 6.

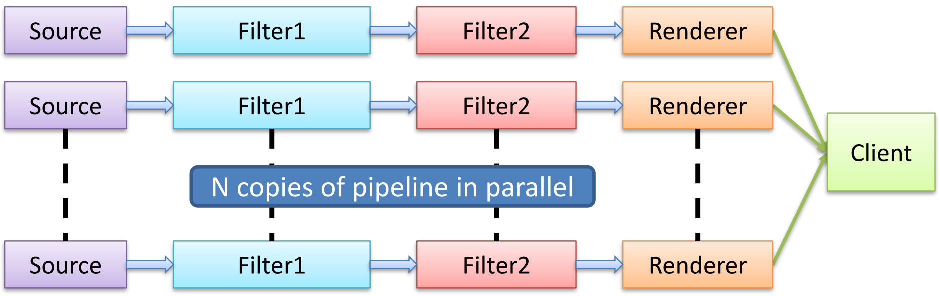


Figure 1.10: ParaView relies on N copies of each pipeline - one per process, synchronisation is explicit and enforced at the filter level (when filters use MPI) and all data/information requests are mirrored on all ranks.

1.4.3 Task based visualisation pipelines of the future

The traditional filter pipeline that has existed in VTK/ParaView for the last 15-20 years (as shown in figure 1.10) is built upon N copies of the pipeline being instantiated and coordinated by a server instance running on each rank. The client communicates requests to each server synchronously and each server delivers the *piece* of the result that it is responsible for. It has been demonstrated already that there are problems with this design arising from the need to load-balance the filters and to keep CPU/GPU resources fully occupied. One of the goals of this thesis is to outline a design of visualisation pipelines for the next generation of software that can address these problems.

With distributed task based visualisation filters, there is no longer any need for N parallel pipelines with identical copies of each filter running in synchronized steps on each node. They may instead be replaced by a single instance of each filter that acts as a coordinator for that particular algorithmic module in the workflow. This coordinating filter-task can spawn as many or as few sub-tasks as needed on whichever nodes are holding the data of interest – those sub-tasks are still instances of the filter in question, but instead of communicating with the client/server mechanism directly, they communicate with the master copy of the relevant filter, the flow of information occurs through the filter-tasks of the main pipeline rather than through all sub-tasks. An operation on a large dataset that spans all nodes can spawn tasks on all nodes, whereas an operation on a smaller subset, needs only to create a reduced set of sub-tasks on the nodes that have the data of interest. The decision about how many tasks to create should be based based on the number of *pieces required* (or available) to process a particular dataset and not decided up front by the server based on the *initial job size* alone. In the case of an in-situ visualisation, we can assume that every node will initially contains some data, but the filters instantiated at later parts of the pipeline may not span all nodes if data reduction has taken place, and dynamic redistribution may modify the nodes doing work as the simulation progresses and changes. Most importantly, work can move to where the data is when it is more efficient than moving the data to free resources.

The HPX AGAS system, operating as a key value store, provides a mechanism to find pieces of data and send work to them, or to retrieve a piece of data and send it as a parameter to a function located elsewhere. It also provides a mechanism to query how much work lies in the queues of any node to see which are idle and which are oversubscribed (see Grubel et al. [2016]), this allows dynamic load balancing based on the actual work being done rather than always using a fixed set of nodes, as is the case in the current ParaView implementation. The only requirement to make this possible is that each node producing data registers it with AGAS to give it a handle (Id) that can be passed around the system and accessed by any task on any node using the handle identifier.

Figure 1.11 illustrates an architectural view of how such a pipeline would span nodes and make use of thread pools. The front-end GUI (or command-line, etc.) still sees a visualisation pipeline as before, but messages and dataflow through the

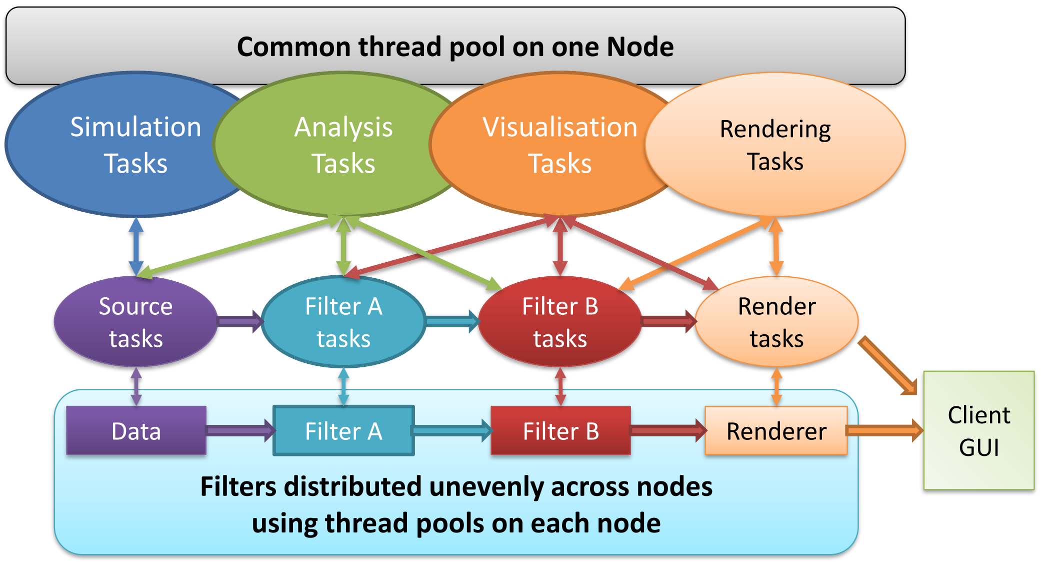


Figure 1.11: Using a task based approach, there is not necessarily any need for a copy of the pipeline on each node. A single filter, source or renderer *task* can coordinate as many or as few sub tasks as required to perform an operation. The pipeline itself needs only to exist *once* and distributed operation and synchronisation of each filter is local only to that filter. There is no need for a server to directly coordinate all branches of the pipeline individually.

pipeline are now only communicated along a single path of principal filter-tasks that in turn coordinate the activities of each of the sub-tasks that they are responsible for. Placement and numbers of sub-tasks may vary for each filter depending upon needs.

As well as providing a much more flexible approach to dataflow, a task based design using a work-stealing runtime allows seamless integration of in-situ processing with the simulation providing the simulation is written using *the same runtime* as the analysis. Simulation tasks and analysis tasks *share the same work queues* and *share the same data*, using reference counting of data handles to ensure that once data has been processed it can be deleted or reused as needed. If the simulation decides that no analysis is needed on a certain time step, references are dropped, no analysis tasks are created and the simulation continues. If analysis is needed, then tasks can be created, they hold onto the data they require until they are done, and then when they complete, data references are dropped and memory is cleaned up. Work stealing between simulation and analysis tasks that are running on the same queues solves the problem of *tightly coupled* applications slowing each other down by forcing one to wait for the other – with work stealing of tasks, simulation can

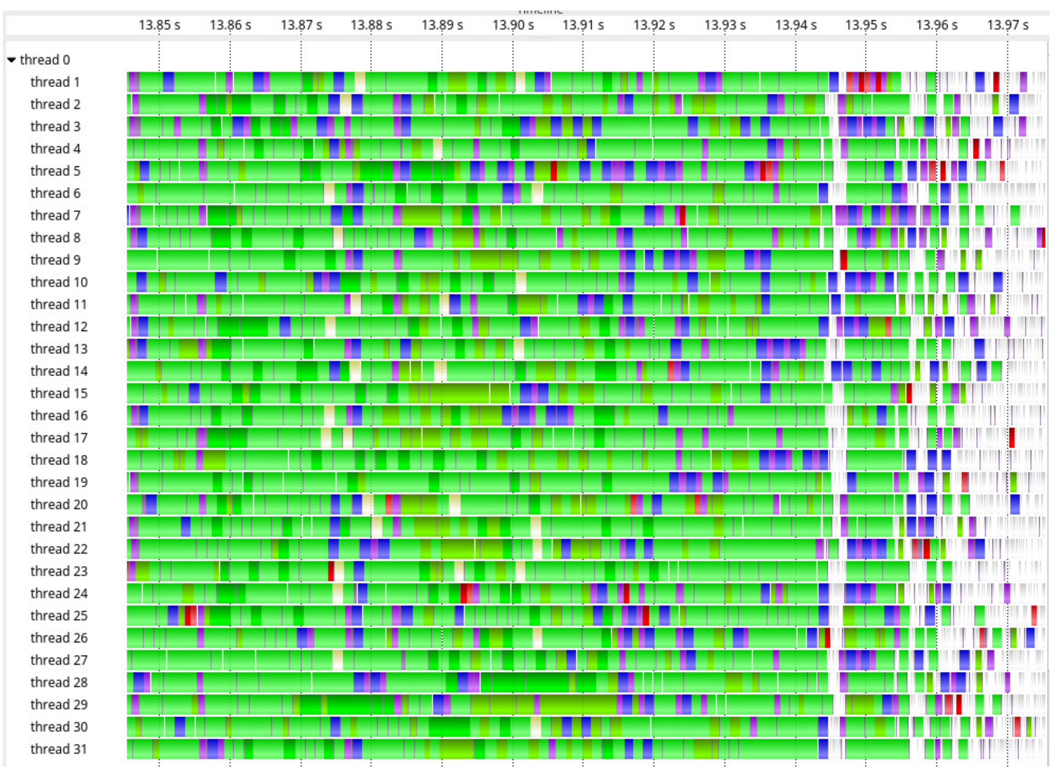


Figure 1.12: A plot of many different simulation tasks coded by colour. Each task runs on a separate thread assigned to a particular core. As soon as a task completes, the next one is taken and executed, leading to very efficient use of the CPU resources - until (in this example) the end of an iteration, where new tasks cannot be generated yet and blank/idle time starts to appear in the task plot. During these idle periods, other work can be performed if it is available.

continue and the user may control the priority of analysis tasks to minimize their impact.

Figure 1.12 shows an example of a task based simulation that creates many thousands of sub-tasks on each iteration of its solver, however, at the end of each iteration, the amount of work to be done decreases and cores begin to sit idle waiting for additional work. during these idle moments, visualisation tasks can be spawned to make use of the wasted resources. On each individual node, analysis tasks can even be placed on the queues of the cores that have generated the data they require, improving data-locality by cache reuse as well as correct placement on the desired node.

There is a further advantage to thread aware pipelines (using an HPX style dataflow) that interplays with the scheduling (or execution) of individual nodes of the graph and is demonstrated very clearly in Vo et al. [2011]. A pull driven pipeline (ie. one that executes filters by traversing from sink back to source) cannot easily execute two branches of a fork-join graph simultaneously because the execution trigger is made from the downstream filter to its upstream source. Since each filter passes its execution trigger upstream to its parent, multiple threads are required causing a race as the two triggers progress on the two paths back to the (common, in this example) source of the pipeline where they both wait for a filter to execute and then trigger their respective pipeline branches to execute. With a push driven pipeline, as soon as a filter completes, it can trigger it's downstream dependencies directly – executing on N threads for N branches of the pipeline becomes trivial. The HPX dataflow execution model using CPS triggering naturally follows this pattern and can easily be integrated into thread pools and schedulers to minimize wasted CPU waits and to steal work from idle cores as need be.

The development therefore of a high speed serialization and network layer that can handle the passing of large datasets (using RMA), in distributed multi-threaded pipelines with low latency information exchange for HPX is a major step towards making this not only possible, but feasible.

1.4.4 Future work

It is clear that the vision of a task-based visualisation pipeline operating in parallel in a distributed setting requires considerable development before it can be fulfilled. The next steps towards this goal are

- Handling of information flow. A ParaView pipeline always has each filter instantiated on each node, so information messages flowing between filters always pass through shared memory within the node. Only synchronization messages (using MPI) pass between nodes. In a distributed pipeline, these informational messages may pass between nodes since there is no longer any guarantee that connected filters are on the same node. The `hpx::async` messaging will be used for this.

- Distributed filters. With individual filters as distributed objects, the synchronization and update mechanism of the pipeline, will flow in a tree like pattern rather than as a series of linear parallel updates. Together with the removal of explicit synchronization between nodes participating in the pipeline, asynchrony is introduced to the update phase and each connected part of the pipeline will need to pass `future` objects rather than datasets directly.
- Load-balancing. Currently there do not exist any task-based counterparts to the Zoltan partitioning software. To enable truly dynamic pipelines, this will need to be developed.
- Compositing and rendering. Compositing of images using the HPX distributed run time has been tested (in Biedert et al. [2017]), but does not make use of the improved network capabilities implemented here. This work will need to be extended and integrated with the ideas presented.

There are a great many opportunities for new research and development in the field of task-based visualisation beyond those stated above. Whilst the VTK-m design of filters using parallel primitives opens up the door to easy implementation with libraries such as HPX, the filters themselves are not always optimal and can be further improved with better task based designs.

1.5 Contribution

In this thesis, a collection of papers have been presented that demonstrate ways of optimising the performance of dataflow based visualisation, analysis and simulation software in the HPC community. The contributions of this work are:

- The demonstration of information (or meta-data) flow within a pipeline based visualisation framework, to solve the problem of handling large time-dependent datasets and the implementation of filters to enable visualisation of flows that were not previously possible, produce comparative views of data and arbitrary manipulations of time within the VTK/ParaView software environment.

- To provide an interface between information flow and a high-quality mesh partitioning library, that can be used to improve load balancing of parallel pipelines in the VTK/ParaView environment.

- To combine the load balancing mechanism with the rendering engine so that high depth complexity meshes can be rendered with full transparency in parallel without duplicating the geometry on every node.

- To also provide a mechanism to generate ghost cells for parallel distributed algorithms using the load balancing pipeline and leveraging the information flow techniques.

- To show how to loosely couple a traditional visualisation pipeline with a live simulation using in-memory files that can be implemented without changes to the simulation software and can additionally be used to steer the software interactively if the simulation is modified accordingly.

- To improve the speed of data transmission and remote function invocation in a distributed dataflow based runtime using RMA and a novel serialization strategy to minimize data duplication and movement.

The developments and contributions to ParaView that have been described are used by researchers and engineers on a daily basis around the world and the improvements made to the HPX library will accelerate the adoption of distributed asynchronous task based runtimes that are seen by many as the future programming model for extreme scale systems. The common theme running through all these developments is the quest for efficient low overhead transfers of data and execution of filters/algorithms between modular components of a software system. Additionally, the foundations have been laid for a next generation analysis framework that will integrate tightly with simulation, allow for dynamic load balancing and scheduling to maximize performance and minimize resource usage.

Chapter 2

Time Dependent Processing in a Parallel Pipeline Architecture

The following paper
Time Dependent Processing in a Parallel Pipeline Architecture

Published in
IEEE Transactions on Visualization and Computer Graphics
Volume 13 Issue 6, November 2007

Chapter 3

Parallel Computational Steering for HPC Applications using HDF5 Files in Distributed Shared Memory

The following paper
Parallel Computational Steering for HPC Applications using HDF5 Files in Distributed Shared Memory

Published in
IEEE Transactions on Visualization and Computer Graphics
Volume: 18, Issue: 6, June 2012

Chapter 4

Practical parallel rendering of detailed neuron simulations

The following paper
Practical parallel rendering of detailed neuron simulations

Published in the proceedings of the
Eurographics Symposium on Parallel Graphics and Visualization 2013

Chapter 5

High-Performance Mesh Partitioning and Ghost Cell Generation for Visualization Software

The following paper
High-Performance Mesh Partitioning and Ghost Cell Generation for Visualization Software

Published in the proceedings of the
Eurographics Symposium on Parallel Graphics and Visualization 2016

Chapter 6

Zero Copy Serialization using RMA in the Distributed Task-Based HPX Runtime

The following paper
Zero Copy Serialization using RMA in the Distributed Task-Based HPX Runtime

Published in the proceedings of the
14th International Conference on Applied Computing 2017

Bibliography

Thesis Introduction

Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016. URL `https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf`.

Hasan Abbasi, Matthew Wolf, Greg Eisenhauer, Scott Klasky, Karsten Schwan, and Fang Zheng. DataStager: scalable data staging services for petascale applications. In *HPDC '09: Proceedings of the 18th ACM international symposium on High performance distributed computing*, pages 39–48, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-587-1. doi: http://doi.acm.org/10.1145/1551609.1551618.

James Ahrens, Charles Law, Will Schroeder, Ken Martin, Kitware Inc, and Michael Papka. A parallel approach for efficiently visualizing extremely large, time-varying datasets. Technical report, Los Alamos National Laboratory, 2000.

James Ahrens, Berk Geveci, and Charles Law. Paraview: An end-user tool for large data visualization. *The Visualization Handbook*, 717, 2005.

Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, and Sam Whittle. The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proc. VLDB Endow.*, 8(12):1792–1803, August 2015. ISSN 2150-8097. doi: 10.14778/2824032.2824076. URL `http://dx.doi.org/10.14778/2824032.2824076`.

A. W. Appel and T. Jim. Continuation-passing, closure-passing style. In *POPL '89: Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 293–302, New York, NY, USA, 1989. ACM. ISBN 0-89791-294-2.

Utkarsh Ayachit, Andrew Bauer, Berk Geveci, Patrick O'Leary, Kenneth Moreland, Nathan Fabian, and Jeffrey Mauldin. Paraview catalyst: Enabling in situ data analysis and visualization. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, ISAV2015, pages 25–29, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-4003-8. doi: 10.1145/2828612.2828624. URL `http://doi.acm.org/10.1145/2828612.2828624`.

Tim Biedert, Kilian Werner, Bernd Hentschel, and Christoph Garth. A Task-Based Parallel Rendering Component For Large-Scale Visualization Applications. In Alexandru Telea and Janine Bennett, editors, *Eurographics Symposium on Parallel Graphics and Visualization*. The Eurographics Association, 2017. ISBN 978-3-03868-034-5. doi: 10.2312/pgv.20171094.

Erik Boman, Karen Devine, Robert Heaphy, Bruce Hendrickson, Vitus Leung, Lee Ann Riesen, Courtenay Vaughan, Umit Catalyurek, Doruk Bozdag, William Mitchell, and James Teresco. *Zoltan 3.0: Parallel Partitioning, Load Balancing, and Data-Management Services; Developer's Guide*. Sandia National Laboratories, Albuquerque, NM, 2007. Tech. Report SAND2007-4749W.

Hank Childs, Eric S. Brugger, Kathleen S. Bonnell, Jeremy S Meredith, Mark Miller, Brad J Whitlock, and Nelson Max. A contract-based system for large data visualization. In *Proceedings of IEEE Visualization 2005*, pages 190–198, 2005.

Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

P. Grubel, H. Kaiser, K. Huck, and J. Cook. Using intrinsic performance counters to assess efficiency in task-based parallel applications. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1692–1701, May 2016. doi: 10.1109/IPDPSW.2016.115.

Juan B. Hernando. *Interactive Visualization of Detailed Large Neocortical Circuit Simulations*. PhD thesis, Facultad de Informática, Universidad Politécnica de Madrid, 2011.

Juan B. Hernando, Felix Schürmann, Henry Markram, and Pedro de Miguel. RT-Neuron, an application for interactive visualization of detailed cortical column simulations. *XVIII Jornadas de Paralelismo, Spain*, 2008.

Wesley M. Johnston, J. R. Paul Hanna, and Richard J. Millar. Advances in dataflow programming languages. *ACM Comput. Surv.*, 36(1):1–34, March 2004. ISSN 0360-0300. doi: 10.1145/1013208.1013209. URL `http://doi.acm.org/10.1145/1013208.1013209`.

Hartmut Kaiser, Thomas Heller, Daniel Bourgeois, and Dietmar Fey. Higher-level parallelization for local and distributed asynchronous task-based programming. In *Proceedings of the First International Workshop on Extreme Scale Programming Models and Middleware*, ESPM '15, pages 29–37, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3996-4. doi: 10.1145/2832241.2832244. URL `http://doi.acm.org/10.1145/2832241.2832244`.

Hartmut Kaiser, Bryce Adelstein-Lelbach, Thomas Heller, Agustin Berge, and John Biddiscombe et.al. HPX V1.0: The C++ Standards Library for Parallelism and Concurrency, 2017. `http://dx.doi.org/10.5281/zenodo.4455628`.

C. Law, W. Schroeder, and K. Martin. A multi-threaded streaming pipeline architecture for large structured data sets. In *Proceedings of IEEE Visualization 1999*, pages 225–232. ACM Press, 1999.

Li-ta Lo, Christopher Sewell, and James Ahrens. PISTON: A Portable Cross-Platform Framework for Data-Parallel Visualization Operators. In Hank Childs, Torsten Kuhlen, and Fabio Marton, editors, *Eurographics Symposium on Parallel Graphics and Visualization*. The Eurographics Association, 2012. ISBN 978-3-905674-35-4. doi: 10.2312/EGPGV/EGPGV12/011-020.

Jay F. Lofstead, Scott Klasky, Karsten Schwan, Norbert Podhorszki, and Chen Jin. Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS). In *CLADE '08: Proceedings of the 6th international workshop on Challenges of large applications in distributed environments*, pages 15–24, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-156-9. doi: http://doi.acm.org/10.1145/1383529.1383533.

Jeremy S. Meredith, Sean Ahern, Dave Pugmire, and Robert Sisneros. EAVL: The Extreme-scale Analysis and Visualization Library. In Hank Childs, Torsten Kuhlen, and Fabio Marton, editors, *Eurographics Symposium on Parallel Graphics* *and Visualization*. The Eurographics Association, 2012. ISBN 978-3-905674-35-4. doi: 10.2312/EGPGV/EGPGV12/021-030.

K. Moreland, C. Sewell, W. Usher, L. t. Lo, J. Meredith, D. Pugmire, J. Kress, H. Schroots, K. L. Ma, H. Childs, M. Larsen, C. M. Chen, R. Maynard, and B. Geveci. Vtk-m: Accelerating the visualization toolkit for massively threaded architectures. *IEEE Computer Graphics and Applications*, 36(3):48–58, May 2016. ISSN 0272-1716. doi: 10.1109/MCG.2016.48.

Kenneth Moreland, Utkarsh Ayachit, Berk Geveci, and Kwan-Liu Ma. Dax toolkit: A proposed framework for data analysis and visualization at extreme scale. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 97–104. IEEE, 2011.

MPI Forum. MPI: A Message-Passing Interface Standard. Version 3, September 4th 2012. available at: `http://www.mpi-forum.org` (Dec. 2009).

John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. *Queue*, 6(2):40–53, March 2008. ISSN 1542-7730. doi: 10.1145/1365490.1365500. URL `http://doi.acm.org/10.1145/1365490.1365500`.

James Reinders. *Intel Threading Building Blocks*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, first edition, 2007. ISBN 9780596514808.

W. Schroeder, K.W. Martin, and B. Lorensen. *The Visualization Toolkit: An Object-oriented Approach to 3D Graphics*. The Visualization Toolkit: An Object-oriented Approach to 3-D Graphics. Prentice Hall PTR, 1996. ISBN 9780131998377. URL `https://books.google.ch/books?id=MrCiQgAACAAJ`.

Don Syme, Tomas Petricek, and Dmitry Lomov. The f# asynchronous programming model. In *Proceedings of the 13th International Conference on Practical Aspects of Declarative Languages*, PADL'11, pages 175–189, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-18377-5. URL `http://dl.acm.org/citation.cfm?id=1946313.1946334`.

The C++ Standards Committee. ISO International Standard ISO/IEC 14882:2011, Programming Language C++. Technical report, Geneva, Switzerland: International Organization for Standardization (ISO)., 2011. `http://www.open-std.org/jtc1/sc22/wg21`.

The C++ Standards Committee. ISO/IEC Draft International Standard 14882, Programming Language C++. Technical report, Geneva, Switzerland: International Organization for Standardization (ISO)., 2017.

The HDF Group. Hierarchical data format version 5, 2000-2017. URL `http://www.hdfgroup.org/HDF5`.

H. T. Vo, J. L. D. Comba, B. Geveci, and C. T. Silva. Streaming-enabled parallel data flow framework in the visualization toolkit. *Computing in Science Engineering*, 13(5):72–83, Sept 2011. ISSN 1521-9615. doi: 10.1109/MCSE.2011.88.

Brad Whitlock, Jean M. Favre, and Jeremy S. Meredith. Parallel in situ coupling of simulation with a fully featured visualization system. In Torsten Kuhlen, Renato Pajarola, and Kun Zhou, editors, *Eurographics Symposium on Parallel Graphics and Visualization*, pages 101–109. Eurographics Association, 2011. ISBN 978-3-905674-32-3. doi: 10.2312/EGPGV/EGPGV11/101-109.

Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark: A unified engine for big data processing. *Commun. ACM*, 59(11):56–65, October 2016. ISSN 0001-0782. doi: 10.1145/2934664. URL `http://doi.acm.org/10.1145/2934664`.

Author Bibliography

Journal

John Biddiscombe, Berk Geveci, Ken Martin, Kenneth Moreland, and David S. Thompson. Time dependent processing in a parallel pipeline architecture. *IEEE Trans. Vis. Comput. Graph.*, 13(6):1376–1383, 2007. doi: 10.1109/TVCG.2007.70600. URL `http://dx.doi.org/10.1109/TVCG.2007.70600`.

John Biddiscombe, David Graham, and Pierre Maruzewski. Visualization and analysis of sph data. *Ercoftac Bulletin*, 76(LMH-ARTICLE-2008-006):9–12, 2008.

John Biddiscombe, Jérome Soumagne, Guillaume Oger, David Guibert, and Jean-Guillaume Piccinali. Parallel computational steering for HPC applications using HDF5 files in distributed shared memory. *IEEE Trans. Vis. Comput. Graph.*, 18(6):852–864, 2012. doi: 10.1109/TVCG.2012.63. URL `http://dx.doi.org/10.1109/TVCG.2012.63`.

Sumeet Gupta, Michaela Soellinger, Deborah M. Grzybowski, Peter Boesiger, John Biddiscombe, Dimos Poulikakos, and Vartan Kurtcuoglu. Cerebrospinal fluid dynamics in the human cranial subarachnoid space: an overlooked mediator of cerebral disease. i. computational model. *Journal of The Royal Society Interface*, 7(49):1195–1204, 2010. ISSN 1742-5689. doi: 10.1098/rsif.2010.0033. URL `http://rsif.royalsocietypublishing.org/content/7/49/1195`.

Guillaume Oger, David Le Touzé, David Guibert, Matthieu De Leffe, John Biddiscombe, Jérome Soumagne, and Jean-Guillaume Piccinali. On distributed memory mpi-based parallelization of SPH codes in massive HPC context. *Computer Physics Communications*, 200:1–14, 2016. doi: 10.1016/j.cpc.2015.08.021. URL `http://dx.doi.org/10.1016/j.cpc.2015.08.021`.

R.J. Powell, A.R. Birks, W.J. Bradford, C.L. Wrench, and J. Biddiscombe.

Using transponders with the ers-1 and topex altimeters to measure orbit altitude to 3 cm. *Advances in Space Research*, 13(5):61 – 67, 1993. ISSN 0273-1177. doi: http://dx.doi.org/10.1016/0273-1177(93)90528-J. URL `http://www.sciencedirect.com/science/article/pii/027311779390528J`.

Benjamin Schindler, Raphael Fuchs, John Biddiscombe, and Ronald Peikert. Predictor-corrector schemes for visualization of smoothed particle hydrodynamics data. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1243–1250, 2009. doi: 10.1109/TVCG.2009.173. URL `http://dx.doi.org/10.1109/TVCG.2009.173`.

Refereed Conference

J. A. Biddiscombe. Modelling line of sight availability for high frequency urban radio networks. In *1999 IEEE MTT-S International Topical Symposium on Technologies for Wireless Applications (Cat. No. 99TH8390)*, pages 105–110, Feb 1999. doi: 10.1109/MTTTWA.1999.755137.

John Biddiscombe. High-Performance Mesh Partitioning and Ghost Cell Generation for Visualization Software. In Enrico Gobbetti and Wes Bethel, editors, *Eurographics Symposium on Parallel Graphics and Visualization*. The Eurographics Association, 2016. ISBN 978-3-03868-006-2. doi: 10.2312/pgv.20161181.

John Biddiscombe, Jérome Soumagne, Guillaume Oger, David Guibert, and Jean-Guillaume Piccinali. Parallel computational steering and analysis for HPC applications using a paraview interface and the HDF5 DSM virtual file driver. In Torsten Kuhlen, Renato Pajarola, and Kun Zhou, editors, *Eurographics Symposium on Parallel Graphics and Visualization, EGPGV 2011, Llandudno, Wales, UK, 2011. Proceedings*, pages 91–100. Eurographics Association, 2011. ISBN 978-3-905674-32-3. doi: 10.2312/EGPGV/EGPGV11/091-100. URL `http://dx.doi.org/10.2312/EGPGV/EGPGV11/091-100`.

John Biddiscombe, Thomas Heller, Anton Bikineev, and Hartmut Kaiser. Zero Copy Serialization using RMA in the Distributed Task-Based HPX runtime. In *14th International Conference on Applied Computing*. IADIS, International Association for Development of the Information Society, 2017.

K. H. Craig, M. P. M. Hall, L. R. Norbury, Y. Seville, M. J. Willis, J. A. Biddiscombe, and T. G. Hayton. Propagation research for millimetrewave cellular systems. In *Tenth International Conference on Antennas and Propagation (Conf. Publ. No. 436)*, volume 2, pages 383–386 vol.2, Apr 1997. doi: 10.1049/cp:19970405.

Stefan Eilemann, Fabien Delalondre, Jon Bernard, Judit Planas, Felix Schürmann, John Biddiscombe, Costas Bekas, Alessandro Curioni, Bernard Metzler, Peter Kaltstein, Peter Morjan, Joachim Fenkes, Ralph Bellofatto, Lars Schneidenbach, T. J. Christopher Ward, and Blake G. Fitch. Key/value-enabled flash memory for complex scientific workflows with on-line analysis and visualization. In *2016 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2016, Chicago, IL, USA, May 23-27, 2016*, pages 608–617. IEEE Computer Society, 2016. ISBN 978-1-5090-2140-6. doi: 10.1109/IPDPS.2016.23. URL `http://dx.doi.org/10.1109/IPDPS.2016.23`.

Juan Hernando, John Biddiscombe, Bidur Bohara, Stefan Eilemann, and Felix Schürmann. Practical parallel rendering of detailed neuron simulations. In Fabio Marton and Kenneth Moreland, editors, *Eurographics Symposium on Parallel Graphics and Visualization, Girona, Spain, 2013. Proceedings*, pages 49–56. Eurographics Association, 2013. ISBN 978-3-905674-45-3. doi: 10.2312/EGPGV/EGPGV13/049-056. URL `http://dx.doi.org/10.2312/EGPGV/EGPGV13/049-056`.

Mark L. Sawley, John Biddiscombe, and Jean M. Favre. Advanced visualization of large datasets for discrete element method simulations. In *Discrete Element Methods (DEM) '07, Brisbane, Australia, 26-29 August 2007*, 2007.

Felix Schürmann, Fabien Delalondre, Pramod S. Kumbhar, John Biddiscombe, Miguel Gila, Davide Tacchella, Alessandro Curioni, Bernard Metzler, Peter Morjan, Joachim Fenkes, Michele Franceschini, Robert S. Germain, Lars Schneidenbach, T. J. Christopher Ward, and Blake G. Fitch. Rebasing I/O for scientific computing: Leveraging storage class memory in an IBM bluegene/q supercomputer. In Julian Martin Kunkel, Thomas Ludwig, and Hans Werner Meuer, editors, *Supercomputing - 29th International Conference, ISC 2014, Leipzig, Germany, June 22-26, 2014. Proceedings*, volume 8488 of *Lecture Notes in Computer Science*, pages 331–347. Springer, 2014. ISBN 978-3-319-07517-4. doi: 10.1007/978-3-319-07518-1_21. URL `http://dx.doi.org/10.1007/978-3-319-07518-1_21`.

Jérome Soumagne and John Biddiscombe. Computational steering and parallel online monitoring using RMA through the HDF5 DSM virtual file driver.

In Mitsuhisa Sato, Satoshi Matsuoka, Peter M. A. Sloot, G. Dick van Albada, and Jack Dongarra, editors, *Proceedings of the International Conference on Computational Science, ICCS 2011, Nanyang Technological University, Singapore, 1-3 June, 2011*, volume 4 of *Procedia Computer Science*, pages 479–488. Elsevier, 2011. doi: 10.1016/j.procs.2011.04.050. URL `http://dx.doi.org/10.1016/j.procs.2011.04.050`.

Jérome Soumagne, John Biddiscombe, and Jerry Clarke. An HDF5 MPI virtual file driver for parallel in-situ post-processing. In Rainer Keller, Edgar Gabriel, Michael M. Resch, and Jack Dongarra, editors, *Recent Advances in the Message Passing Interface - 17th European MPI Users' Group Meeting, EuroMPI 2010, Stuttgart, Germany, September 12-15, 2010. Proceedings*, volume 6305 of *Lecture Notes in Computer Science*, pages 62–71. Springer, 2010. ISBN 978-3-642-15645-8. doi: 10.1007/978-3-642-15646-5_7. URL `http://dx.doi.org/10.1007/978-3-642-15646-5_7`.

Jérome Soumagne, John Biddiscombe, and Aurélien Esnard. Data redistribution using one-sided transfers to in-memory HDF5 files. In Yiannis Cotronis, Anthony Danalis, Dimitrios S. Nikolopoulos, and Jack Dongarra, editors, *Recent Advances in the Message Passing Interface - 18th European MPI Users' Group Meeting, EuroMPI 2011, Santorini, Greece, September 18-21, 2011. Proceedings*, volume 6960 of *Lecture Notes in Computer Science*, pages 198–207. Springer, 2011. ISBN 978-3-642-24448-3. doi: 10.1007/978-3-642-24449-0_23. URL `http://dx.doi.org/10.1007/978-3-642-24449-0_23`.

David Le Touzé, John Biddiscombe, Andrea Colagrossi, Erwan Jacquin, Francis Leboeuf, Jean-Christophe Marongiu, Nathan J. Quinlan, Andrea Amicarelli, Matteo Antuono, Daniel A. Barcarolo, Mihai Basa, Joelle Caro, Matthieu De Leffe, Nicolas Grenier, Pierre-Michel Guilcher, Matthieu Kerhuel, Fang Le, Libor Lobovský, Salvatore Marrone, Adam Marsh, Guillaume Oger, Etienne Parkinson, and Jérome Soumagne. Next-generation multi-mechanics simulation engine in a highly interactive environment. In Elisabeth Giacobino and Rolf Pfeifer, editors, *Proceedings of the 2nd European Future Technologies Conference and Exhibition, FET 2011, Budapest, Hungary, May 4-6, 2011*, volume 7 of *Procedia Computer Science*, pages 292–293. Elsevier, 2011. doi: 10.1016/j.procs.2011.09.077. URL `http://dx.doi.org/10.1016/j.procs.2011.09.077`.