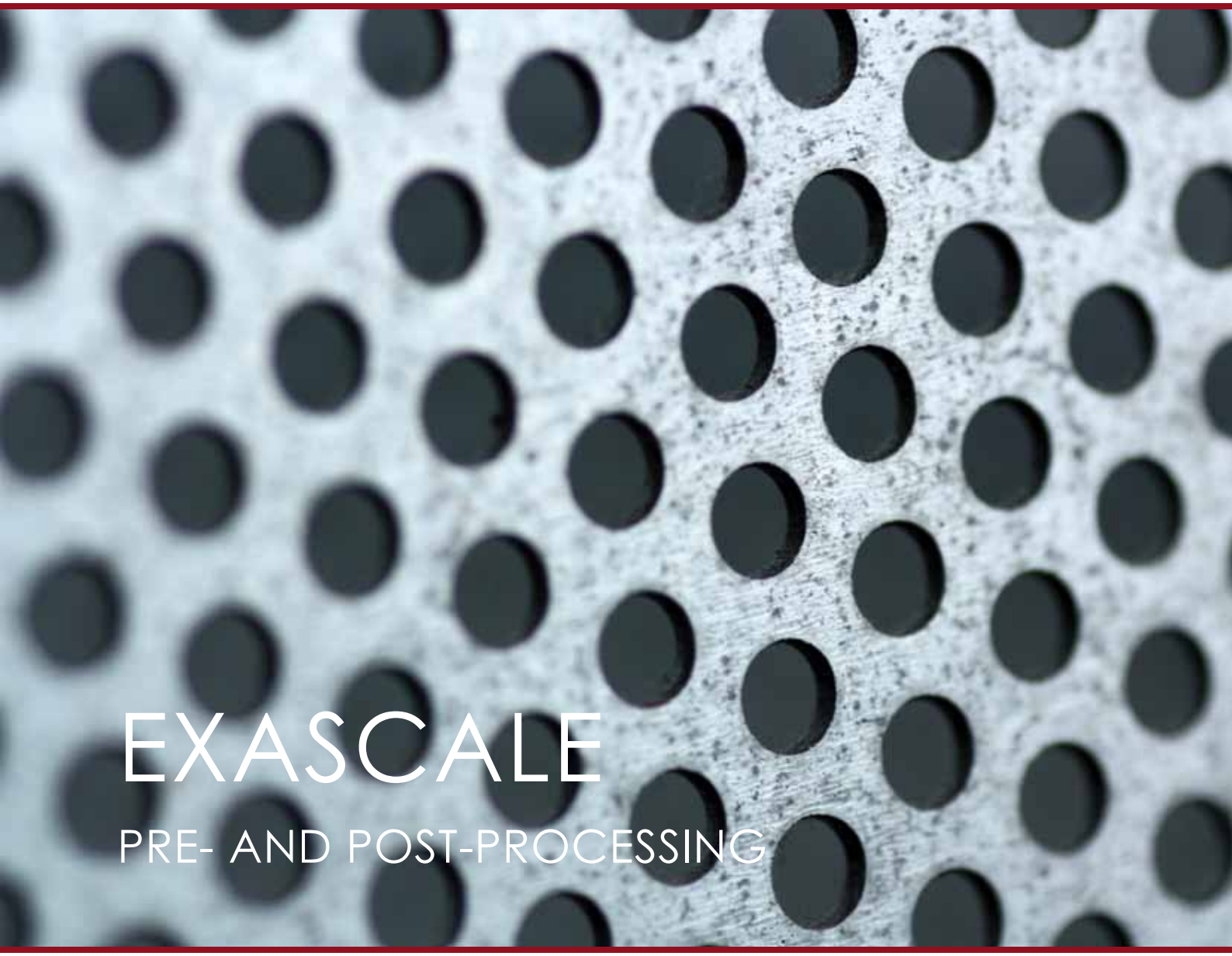
CRESTA



EXASCALE

PRE- AND POST-PROCESSING

Strategy and Software Technology
CRESTA White Paper

Authors: Achim Basermann (DLR), Gregor Matura (DLR), Fang Chen (DLR),
Andreas Gerndt (DLR), Martin Aumüller (USTUTT), Derek Groen (UCL)

Editors: Lorna Smith, Catherine Inglis (UEDIN)

Collaborative Research Into Exascale Systemware, Tools and Applications (CRESTA)
ICT-2011.9.13 Exascale computing, software and simulation

SEVENTH FRAMEWORK PROGRAMME

FOREWORD

BY **DR ACHIM BASERMANN**, FROM DLR, GERMAN AEROSPACE CENTER, SIMULATION AND SOFTWARE TECHNOLOGY, GERMANY, AND "USER TOOLS" WORK PACKAGE LEADER IN CRESTA.

Today's large-scale simulations deal with complex geometries and numerical data on an extreme scale. As computation approaches the exascale, it will no longer be possible to write and store full-sized result data sets. In-situ data analysis and scientific visualisation provide feasible solutions to the analysis of complex large-scale simulations. To bring pre- and post-processing to the exascale we must consider modifications to data structure and memory layout, and address latency and error resiliency.

For pre-processing, it is crucial to have a load-balancing strategy that supports multiple simulation phases and includes their costs in order to calculate a data distribution that leads to an optimal performance for the full simulation. For distributed post-processing, in-situ processing is a key concept in order to perform scalable on-the-fly data analysis and user interaction to on-going simulations. Remote hybrid rendering (RHR) is suitable to access remote exascale simulations from immersive projection environments over the Internet. RHR decouples local interaction from remote rendering and thus guarantees smooth interactivity during exploration of large remote data sets.

In this white paper, we present strategies, algorithms and techniques for pre- and post-processing in exascale scenarios. With software prototypes developed in CRESTA and integrated into CRESTA applications, we demonstrate the effectiveness of our pre- and post-processing concepts for extremely parallel systems.

ABOUT CRESTA

BY **PROFESSOR MARK PARSONS**, COORDINATOR OF THE CRESTA PROJECT AND EXECUTIVE DIRECTOR AT EPCC, THE UNIVERSITY OF EDINBURGH, UK.

The Collaborative Research into Exascale, Systemware Tools and Applications (CRESTA) project is focused on the software challenges of exascale computing, making it a unique project. While a number of projects worldwide are studying hardware aspects of the race to perform 1018 calculations per second, no other project is focusing on the exascale software stack in the way that we are.

By limiting our work to a small set of representative applications we hope to develop key insights into the necessary changes to applications and system software required to compute at this scale.

When studying how to compute at the exascale it is very easy to slip into a comfort zone where incremental improvements to applications eventually develop the necessary performance. In CRESTA, we recognise that incremental improvements are simply not enough and we need to look at disruptive changes to the HPC software stack from the operating system, through tools and libraries to the applications themselves. From the mid-1990s to the end of the last decade, HPC systems have remained remarkably similar (with performance increases being delivered largely through the increase in microprocessor speeds). Today, at the petascale, we are already in an era of massive parallelism with many systems containing several hundred thousand cores. At the exascale, HPC systems may have tens of millions of cores. We simply don't know how to compute with such a high level of parallelism.

CRESTA is studying these issues and identifying a huge range of challenges. With the first exascale system expected in the early 2020s, we need to prepare now for the software challenges we face which, we believe, greatly outnumber the corresponding hardware challenges. It is a very exciting time to be involved in such a project.

CRESTA WHITE PAPERS

BY **DR LORNA SMITH**, PROJECT MANAGER FOR THE CRESTA PROJECT AND GROUP MANAGER AT EPCC, THE UNIVERSITY OF EDINBURGH, UK.

CRESTA is preparing a series of key applications for exascale, together with building and exploring appropriate software – systemware in CRESTA terms - for exascale platforms. Associated with this is a core focus on exascale research: research aimed at guiding the HPC community through the many exascale challenges.

Key outcomes from this research are CRESTA's series of white papers. Covering important exascale topics including new models, algorithms, techniques, applications and software components for exascale, the papers will describe the challenges and current state of the art and propose solutions and strategies for each of these topics.

Handling pre- and post- processing on exascale platforms will be a significant challenge, but one that many applications will need to overcome. This white paper considers this challenge, investigating and describing new techniques and software to address these challenges.

CONTENTS

1 EXECUTIVE SUMMARY 1

2 INTRODUCTION 2

2.1 GLOSSARY OF ACRONYMS 2

3 EXASCALE PRE-PROCESSING 3

3.1 LOAD BALANCING 3

3.1.1 Multiple Stages 4

3.1.2 Repartitioning 4

3.1.3 Initialisation 4

3.2 REALISATION 4

3.2.1 Usability 5

3.2.2 Algorithms for Load-Balancing 8

3.2.3 Extensions 9

4 EXASCALE POST-PROCESSING 10

4.1 INTERACTIVE VISUALISATION ON EXASCALE SYSTEMS 10

4.1.1 In-Situ, Live Processing where the Simulation Happens 10

4.1.2 Rendering 11

4.1.3 User interaction 11

4.1.4 Post-Processing System Architecture 11

4.1.5 Application Study: Co-Design with HemeLB 13

4.2 REMOTE HYBRID RENDERING IN EXASCALE SIMULATION SCENARIOS 16

4.2.1 Motivation 16

4.2.2 Design & Implementation 18

4.2.3 First Experiences 22

4.2.4 Open Challenges 23

5 CONCLUSIONS 24

6 REFERENCES 25

INDEX OF FIGURES

Figure 1: A sample graph (without weights); [3] 5

Figure 2: Array content of distributed CSR format for the sample graph using 3 processors; [3] 5

Figure 3: PPStee flow chart 6

Figure 4: HemeLB runtimes for data set data_02M with plain HemeLB and HemeLB with PPStee using ParMETIS PTScotch and Zoltan on HECToR 7

Figure 5: Calculation time of HemeLB on ARCHER for geometry aneurysm_0.025mm with PPStee using one of three partitioning libraries 8

Figure 6: Partitioning time of HemeLB on ARCHER for geometry aneurysm_0.025mm with PPStee using one of three partitioning libraries 9

Figure 7: Interactive post-processing system architecture for exascale systems 12

Figure 8: An In-situ monitoring with only a screen showing the network image streamed to the front-end application 12

Figure 9: An Interactive post-processing architecture with a user interacting on the front-end 13

Figure 10: Time measurement for generating image with resolution 128x128 13

Figure 11: Time measurement for generating image with resolution 256x256 14

Figure 12: Time measurement for generating image with resolution 512x512 14

Figure 13: Time measurement for generating image with resolution 1024x1024 14

Figure 14: Latency (Front-end frame rates) for image display at front-end 14

Figure 15: User interacting with the aneurysm data in front of a power-wall 15

Figure 16: A pure remote vs. a remote hybrid rendering workflow 17

Figure 17: Local context information (left), remote simulation data (middle), fused image shown to the user (right) 17

Figure 18: Typical network topology for a remote visualisation task 18

Figure 19: Contribution of nodes in different colours (left) and final composited image (right) of IHS pump turbine test case 20

Figure 20: Reference image for depth buffer compression quality assessment 22

Figure 21: Depth buffer compression quality – left: original image, middle: with compressed depth, right: differences highlighted in red 22

INDEX OF TABLES

Table 1: Compression ratio and quality for lossy GPU based depth compression for the image in Figure 20 22

1 EXECUTIVE SUMMARY

Today's large-scale simulations deal with complex geometries and numerical data on an extreme scale. As computation approaches the exascale, it will no longer be possible to write and store full-sized result data sets. In-situ data analysis and scientific visualisation provide feasible solutions to the analysis of complex large-scale simulations. To bring pre- and post-processing to the exascale we must consider modifications to data structure and memory layout, and address latency and error resiliency.

Load balancing is a crucial pre-processing task on extremely parallel systems. Here, our focus is on a load balancing strategy that supports multiple simulation phases and includes their costs to calculate a data distribution that leads to an optimal performance for the full simulation.

The software library PPStee developed in CRESTA already incorporates this idea. It was explicitly designed to support multiple simulation phases. The user provides communication costs of a simulation phase represented as edge weights of a graph corresponding to the simulation data. The according computation costs of the phase are matched to vertex weights. As multiple weight sets can be included, the partitioning of the simulation data is calculated to achieve an optimal load balance covering the full simulation cycle. PPStee supports various partitioning tools, repartitioning and improved initial data distributions by exploiting knowledge from previous simulation runs. PPStee was successfully integrated into several large-scale fluid simulation codes. Here, we demonstrate the flexibility of PPStee with the hemodynamic simulation code HemeLB.

In-situ processing has become a key concept in exascale data post-processing and visualisation. Waiting for a simulation to finish and writing out huge amounts of simulation output is no longer a viable solution for data analysis. Instead, visualisation and data analysis must happen when and where a certain simulation step has been carried out, as the so-called in-situ processing.

Our in-situ processing system provides scalable distributed post-processing. This system supports on-the-fly data analysis and user interaction to on-going simulations. Here, we demonstrate the feasibility of our system by an online-monitoring scenario with the hemodynamic simulation code HemeLB.

Remote hybrid rendering (RHR) is used to access remote exascale simulations from immersive projection environments over the Internet. The display system may range from a desktop computer to an immersive virtual environment such as a CAVE. The display system forwards user input to the visualisation cluster, which uses highly scalable methods to render images of the post-processed simulation data and returns them to the display system. The display system enriches these with context information rendered locally, before they are shown. RHR decouples local interaction from remote rendering and thus guarantees smooth interactivity during exploration of large remote data sets.

Here, we discuss strategies, algorithms and techniques for RHR in exascale scenarios and present performance measurements for a prototype developed in CRESTA. For performance analysis, the prototype has been instrumented to collect timing information, compression ratios and image quality metrics.

2 INTRODUCTION

Large-scale simulations on extremely parallel systems require highly scalable pre- and post-processing software for load balancing as well as data analysis and visualisation.

In this white paper, we present strategies, algorithms and techniques for pre- and post-processing in exascale scenarios. With software prototypes developed in CRESTA and integrated into CRESTA applications, we demonstrate the effectiveness of our pre- and post-processing concepts for extremely parallel systems.

Section 3 tackles exascale pre-processing with a focus on load balancing by suitable partitioning (cf. detailed discussion in Section 3.1). Section 3.2 describes a software approach for achieving load balance in large-scale multi-phase simulations and demonstrates the flexibility of this software with the hemodynamic simulation code HemeLB.

Section 4 is devoted to exascale post-processing concepts. While Section 4.1 treats strategies for interactive visualisation on exascale systems, Section 4.2 presents remote hybrid rendering (RHR) methods in exascale simulation scenarios. Software approaches and a case study for hemodynamic simulation regarding interactive visualisation are described in Sections 4.1.4 and 4.1.5, respectively. RHR software and performance are discussed in Sections 4.2.2 and 4.2.3, respectively.

In Section 5 we draw conclusions on how we believe pre- and post-processing should be adapted to the requirements of exascale simulation.

2.1 Glossary of Acronyms

API	APPLICATION PROGRAMMING INTERFACE
CPU	CENTRAL PROCESSING UNIT
CRESTA	COLLABORATIVE RESEARCH INTO EXASCALE SYSTEMWARE, TOOLS AND APPLICATIONS
CSR	COMPRESSED SPARSE ROW
CUDA	COMPUTE UNIFIED DEVICE ARCHITECTURE
DLR	DEUTSCHES ZENTRUM FÜR LUFT UND RAUMFAHRT (GERMAN AEROSPACE CENTER)
EC	EUROPEAN COMMISSION
GPU	GRAPHICS PROCESSING UNIT
GPGPU	GENERAL PURPOSE GRAPHICS PROCESSING UNIT
HPC	HIGH PERFORMANCE COMPUTING
IHS	INSTITUTE OF FLUID MECHANICS AND HYDRAULIC MACHINERY
JPEG	JOINT PHOTOGRAPHIC EXPERTS GROUP
MPI	MESSAGE PASSING INTERFACE
MTTF	MEAN TIME TO FAILURE
OPENGL	OPEN GRAPHICS LIBRARY
PSNR	PEAK SIGNAL-TO-NOISE RATIO
RFB	REMOTE FRAMEBUFFER
RHR	REMOTE HYBRID RENDERING
SIMD	SINGLE INSTRUCTION MULTIPLE DATA
UCL	UNIVERSITY COLLEGE LONDON
USTUTT	UNIVERSITY OF STUTTGART
VNC	VIRTUAL NETWORK COMPUTING
VR	VIRTUAL REALITY

3 EXASCALE PRE-PROCESSING

Traditionally, all the tasks done before the real simulation starts are called pre-processing tasks. This includes file I/O operations to read in the necessary data, preparation of this data as well as adjustments according to the simulation needs, e.g., mesh manipulations of any kind. Also, partitioning and distribution of the data is a vital task of the pre-processing phase of a simulation to gain a good load balance.

In the exascale regime, we deal with simulations that are inherently large, complex and time-consuming. We see a shift from check pointing that becomes unaffordable to more and more interactivity. As the arising amount of data is quite large, post-processing methods like result analysis or visualisation have to be included into the simulation.

Thus, the challenges of pre-processing in the exascale regime are evident. The simulation data must still be arranged and prepared. This data management must suit the needs of all parts of the simulation core and all other computations before the simulation starts and after results are computed. In addition to these simulation-internal needs, the data distribution must lead to an adequate performance of the full simulation, including all its subparts.

Here, the performance is directly related to the execution costs. Beside the costs of the solver part that obviously need the best available optimisation, the total load balance is essential. It must include all subparts of the simulation as they may differ substantially in their costs for communication and computation. To achieve an even better performance result, load balance might additionally include real-time system details like information provided by a fault tolerance or system monitoring framework.

Accordingly, our focus is on a load balancing that supports multiple simulation phases and includes their costs to calculate a data distribution that leads to an optimal performance for the full simulation. We illustrate some aspects of such a load balancing before we describe important details that have to be considered for an implementation.

3.1 Load Balancing

As pointed out above, load balancing is a crucial purpose of pre-processing. Yet, it is not sufficient to optimise the data distribution only for the solver. Other simulation parts like initial mesh manipulations, subsequent result analysis or in-situ visualisation impose further constraints. Each of these multiple stages possesses a different distribution of load in terms of communication and computation and needs a matching distribution of data. Thus support for multiple stages in calculating the data distribution of the simulation is mandatory.

The increasing interactivity introduces another vital aspect that tackles load balancing. Exascale simulations run in cycles to be able to provide intermediate potentially visual results for the user. As the user can intervene and change significant simulation parameters that, consequently, change the load of all stages, the pre-processing must be adapted repeatedly. This leads directly to optimisation opportunities for the computation of the initial data partitioning. The evaluation of the data distribution from previous simulation cycles can trigger or improve a repartitioning. Once implemented, this utilisation of previous load statistics also allows for a better initialisation of the first cycle in a new simulation run.

3.1.1 Multiple Stages

When calculating a data distribution for an exascale simulation, pre-processing must cover all stages of the simulation to achieve a good load balance. Data initialisation with file I/O, mesh generation and mesh manipulations may contribute to the simulation cycle as well as the core algorithms, a subsequent result analysis, an in-situ visualisation of the results or a preparation of data used for remote rendering. Each of these stages can have its own distribution pattern of the simulation data and its own load footprint regarding costs of communication and computation.

The software library PPStee [1][2][3] already incorporates this idea. It was explicitly designed to support multiple simulation stages. The user provides communication costs of a simulation stage represented as edge weights of a graph corresponding to the simulation data. The according computation costs of the stage are matched to vertex weights. As multiple weight sets can be included, the partitioning of the simulation data is calculated to achieve an optimal load balance covering the full simulation cycle.

3.1.2 Repartitioning

Now, if we assume that we have a simulation that runs in cycles and gets balanced according to its various stages, we can improve the balancing as the simulation run progresses. We can evaluate the load balance of previous cycles, mix it with expectations based on current user input and use this to improve the partitioning of the next cycle.

In practice, all partitioning libraries supported in PPStee provide the feature to calculate a partitioning together with the costs needed for the creation of this distribution based on an initial partitioning. Thus PPStee is capable not only of delivering an optimal partitioning but also of deciding whether it pays off to actually use this partitioning.

3.1.3 Initialisation

As discussed above, we can exploit a measurement of the partitioning quality of previous simulation cycles and achieve a better load balance for the next cycle. Analogously, we can use this measurement to improve even the first partitioning call of a simulation. Normally, this first call would calculate a partitioning from scratch. However, if we re-use partitioning data from a previous simulation run, the partitioning library provides a better partitioning result and even saves calculation time compared to a zero-initialised partitioning call. This improved initialisation is, e.g., in PPStee, equivalent to a repartitioning call and therefore implicitly included.

3.2 Realisation

In practice, the realisation of an intermediate software layer that ensures an optimal load balance and covers the full simulation run requires consideration of some important details.

For a start, the data format that is used to exchange information on costs of communication and computation has to be carefully chosen. This data format must be compact and be flexible enough to support a broad spectrum of data layouts used in simulations. The former property helps to keep the overhead of the additional pre-processing software layer small in terms of additional data processing time and memory consumption. The latter property ensures compatibility with application scenarios in various simulation codes so that integration of advanced pre-processing methods into existing codes is simple.

Another important aspect is the support for a diversity of partitioning libraries that actually compute the balanced distribution of simulation costs and data. As each of the widely used partitioning libraries, i.e., ParMETIS [4], PTScotch [5] and Zoltan [6], implement a distinct method to achieve a balanced partitioning, it is surely favourable to test each one on the data of a simulation to find the most suitable candidate for the given data. Also, this support for different partitioning libraries demands the capability to easily swap the partitioning library without any significant code adjustments.

Beyond these simulation-related issues, pre-processing and, specifically, load balancing benefit from additional system-relevant information, e.g. provided by system monitoring. Particularly in the exascale regime, support for a fault tolerance framework is indispensable as the simulation has to react in real-time to any faulty system components.

3.2.1 Usability

We now want to describe in more detail how to provide a software package that is both easy to use and easy to implement. We illustrate the aspects mentioned above that apply to completely new simulations as well as to existing simulation codes.

3.2.1.1 Data Format

Firstly, the data format that is used to exchange information on costs of communication and computation must be compact. Graph data structures are suitable. This lowers the additional burden imposed on the memory. Such graph data structures might even be allocated already in the simulation code and thus eliminate the need for extra space completely.

Secondly, the data format must be versatile in different ways. The internal layout of data varies a lot among simulation codes. Additionally, each stage of a given simulation can have its own view of the simulation data. Thus it is necessary to implement a format that is compatible with a wide range of data layouts.

Lastly, we state the obvious fact that the format must be parallelisable. A distribution to huge amounts of cores should both be possible and not diminish the other properties.

A good candidate for a data format that supports all desired properties is the graph format proposed by and used in ParMETIS [4]. For a further, convenient analysis, Figure 1 shows a sample graph that is assembled to the CSR-like format depicted in Figure 2 (CSR: Compressed Sparse Row). Here, "vtxdist" contains the information on the global distribution of the vertices, "xadj" encodes the number of edges, or adjacencies, for the vertices, and "adjncy" holds these neighbour data. We clearly see the parallel structure and almost no redundant content. Solely the global vertex distribution is repeated, but this saves some expensive collective communication operations in the course of the simulation. Also through its compact layout, this format is widely compatible with other, more sophisticated data structures.

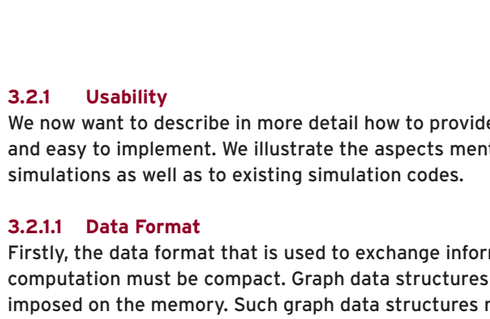


Figure 1: A sample graph (without weights); [3]

Processor 0:	xadj	0 2 5 8 11 13
	adjncy	1 5 0 2 6 1 3 7 2 4 8 3 9
	vtxdist	0 5 10 15
Processor 1:	xadj	0 3 7 11 15 18
	adjncy	0 6 10 1 5 7 11 2 6 8 12 3 7 9 13 4 8 14
	vtxdist	0 5 10 15
Processor 2:	xadj	0 2 5 8 11 13
	adjncy	5 11 6 10 12 7 11 13 8 12 14 9 13
	vtxdist	0 5 10 15

Figure 2: Array content of distributed CSR format for the sample graph using 3 processors; [3]

PPStee relies on this data format to exchange costs and connectivity. Here, the vertices represent computation units and carry a vertex weight that correlates to the computation cost of this unit in a specific phase. The edges represent communication patterns and provide communication costs via their respective edge weights.

3.2.1.2 Integration

We want to illustrate the integration of a pre-processing layer for load balancing using the example of PPStee. Basically, the procedure of integration into an existing code consists of three steps that require only minor code changes (cf. Figure 3. A detailed description including an example can be found in [1]). Firstly, the simulation data is prepared which should be easily possible given the versatile data format (cf. last section). Secondly, we submit graph data and costs of all existing phases as graph weights to PPStee. We subsequently trigger the partitioning calculation and receive a partitioning that we can now use to distribute simulation data accordingly.

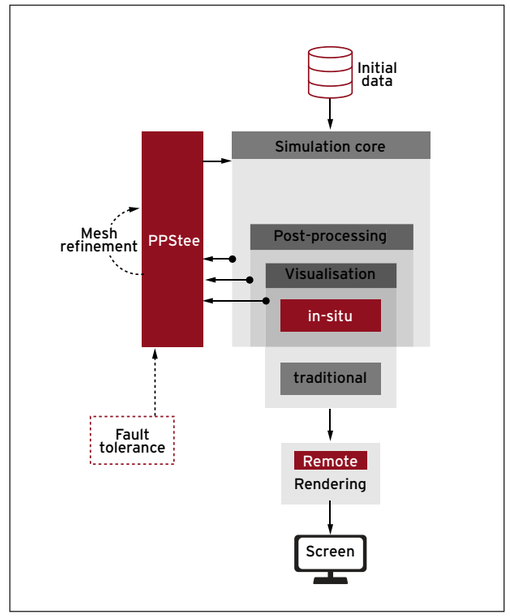


Figure 3: PPStee flow chart

The example shows that it is particularly easy to integrate PPStee into a simulation. Especially if an existing code already uses a partitioning library, the position of the necessary code changes is predefined by the position of the call to the given partitioning library. Thus only minor code changes are required for an integration. Additionally, we note that the interference with the architecture of the simulation or its data structures is usually low.

3.2.1.3 Overhead

When designing software that introduces a new layer in the application, one has to pay attention to overhead that is implicitly brought in. Regarding a pre-processing package that uses partitioning libraries to calculate an optimal distribution of costs and data, two major sources of overhead arise. The overhead implied by the calculation of the partitioning in one of the partitioning libraries will be tackled in the next section in detail. On the other hand, we can investigate the overhead of the software layer per se. e.g., an improper handling of data or too many unnecessary operations can impose run time or memory deficiencies.

The analysis of such an overhead becomes clean and easy if we eliminate the time of the call to the partitioning library in the measurement of the run time. For an illustration, we use HemeLB [7] together with PPStee. The unmodified version of HemeLB already uses the partitioning library ParMETIS to obtain an optimised distribution pattern for its simulation data. We easily integrated PPStee into HemeLB (as described in section 3.2.1.2, and in [2]). We now trigger different partitioning libraries and compare the run time to the run time of the unmodified version of HemeLB. Figure 4 depicts the run time results obtained on HECToR [8] for a HemeLB geometry called "data_02M" and a number of cores between 32 and 2048.

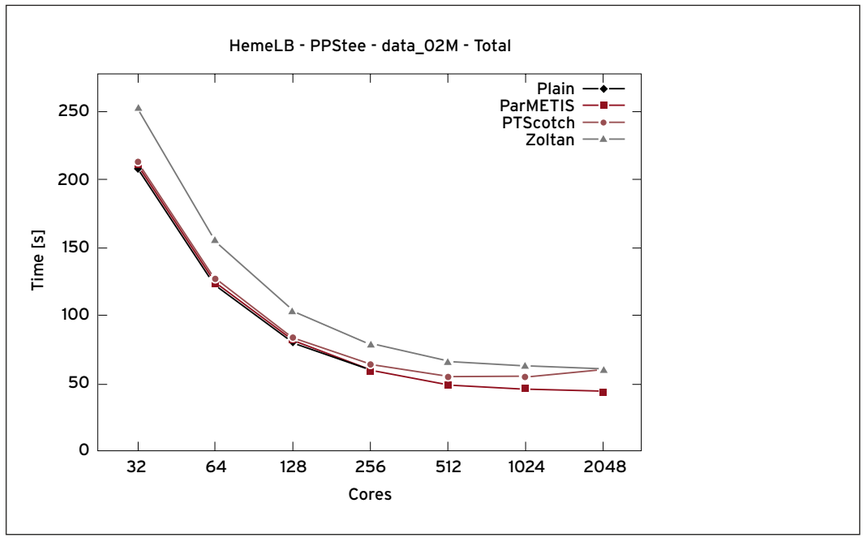


Figure 4: HemeLB runtimes for data set data_02M with plain HemeLB and HemeLB with PPStee using ParMETIS PTScotch and Zoltan on HECToR

In the run time measurements, we observe both described overheads. The comparison of plain HemeLB versus HemeLB using PPStee with ParMETIS shows almost no difference. Since both versions use the same partitioning library, we conclude that the overhead of PPStee is low enough to be insignificant for the simulation. On the other hand, the other two plotted curves show that there are significant differences in the calculation time of the partitioning libraries. As already mentioned, we describe these issues in the following section.

3.2.2 Algorithms for Load-Balancing

The algorithms that are used to calculate the partitioning play a key role in a software package for load balancing and hence in pre-processing in general. However, each partitioning library implements its own method for the computation of the partitioning. e.g., PTScotch [5] uses a divide and conquer approach, while ParMETIS [4] is based on a parallel multilevel k-way algorithm. On the other hand, we are confronted with a variety of simulation codes. Each code uses its own data structures and software architecture in general. The differences are significant.

To illustrate this encounter of algorithm and simulation, we again address HemeLB with PPStee. We have already seen how easily PPStee is integrated into an existing code. In addition, PPStee provides a simple mechanism for the choice of the partitioning library that is used. Since PPStee uses a versatile data format (cf. section 3.2.1.1) that is compatible with all three major partitioning libraries, only one parameter must be changed to specify one or the other partitioning library.

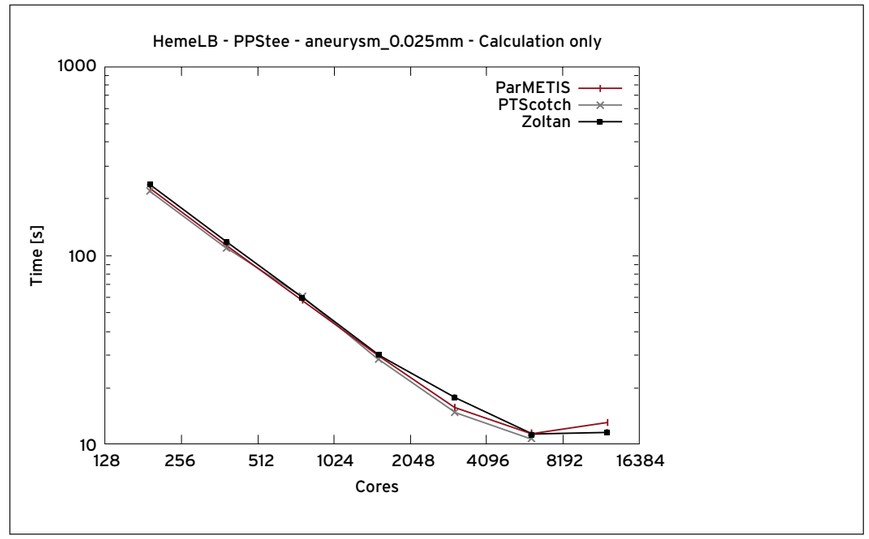


Figure 5: Calculation time of HemeLB on ARCHER for geometry aneurysm_0.025mm with PPStee using one of three partitioning libraries

Figure 5 depicts the results of run time measurements for HemeLB with PPStee using one of the three partitioning libraries. Here, only the calculation around the simulation kernel is measured and, thus, the results directly show the quality of the partitioning. The measurements were obtained on ARCHER [9] with a varying number of cores between 192 and 12k. We observe an equal partitioning quality for all three partitioning libraries up to 1,536 cores. Starting at 3,072 cores, we see differences in the obtained quality. For example, ParMETIS produces a better partitioning for 3,072 cores than Zoltan, but, for 12k cores, the ParMETIS partitioning loses performance compared to Zoltan. This behaviour reflects that the partitioning quality depends on an interaction of the simulation and the partitioning library and, additionally, on the number of cores.

The mutual dependence of simulation and partitioning algorithm becomes even more evident when we compare the time that is needed to calculate the partitioning. This partitioning time is not significant for a very long simulation where simulation data is partitioned only once at the beginning. However, if the simulation is interactive and needs on-going repartitioning, e.g., after a couple of time steps, or the simulation is rather short with a huge amount of data to be partitioned, partitioning time is going to be an issue. We, therefore, compare the partitioning time for the same simulation runs that were performed for Figure 5, i.e., simulation runs with HemeLB and PPStee using one of the three partitioning libraries ParMETIS, PTScotch or Zoltan. Figure 6 depicts the measured timings.

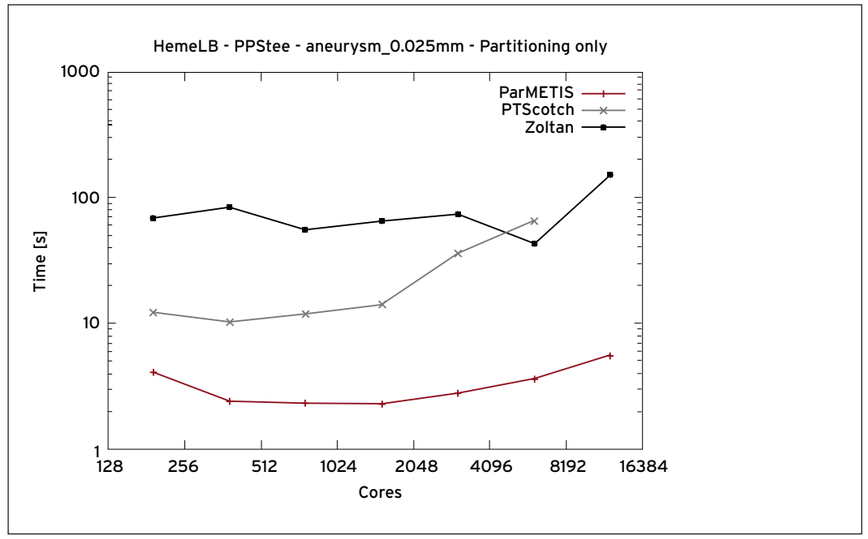


Figure 6: Partitioning time of HemeLB on ARCHER for geometry aneurysm_0.025mm with PPStee using one of three partitioning libraries.

We point out three observations. Firstly, ParMETIS is, at least for a simulation run of HemeLB and with this specific data set, faster than the other two partitioning libraries. Secondly, PTScotch is slower than Zoltan on 6,144 cores even though it was significantly faster up to 1,536 cores. Thirdly, none of the partitioning libraries scales well or, to be more precise, all of them even show an increase in runtime beginning at 1,536 cores. It is doubtful that this situation will change for another simulation code or another data set.

In conclusion, it is beneficial to support a variety of partitioning libraries and, therewith, partitioning algorithms. Together with a simple method to change the partitioning library that is used, the integration of multiple partitioning libraries leads to a comfortable situation for the user. The user can easily test which partitioning library best matches the given simulation and then select the most suitable one.

3.2.3 Extensions

In addition to the simulation-related issues discussed above, there are some system-related issues that must be considered in pre-processing. For example, the system or the system administrator of an exascale machine might reduce the clock speed of some parts of the machine, potentially to save energy. Or the specific system architecture and node structure of a future exascale system might lead to a time-dependent node performance that has to be considered. Another significant point is the permanently decreasing Mean Time To Failure (MTTF) for huge machines. As MTTF drops below the runtime of a simulation, the usage of techniques for fault tolerance such as fault detection and post-fault recovery is unavoidable.

It is necessary that this kind of information is available on a system through an API to system monitoring or a fault tolerance framework that is directly connected to the hardware. However, pre-processing has to include such information to obtain a good load balance for the simulation. Thus, a mechanism for repartitioning is inevitably required and has to be integrated. After all, this additional effort leads to a pre-processing that faces the growing requirements for interactivity of exascale simulations.

4 EXASCALE POST-PROCESSING

4.1 Interactive Visualisation on Exascale Systems

While numerical simulation is developing towards exascale, it is no longer a viable solution to store all simulation data to disk. Limited system I/O capacity hinders the simulation from intermediate result output. Therefore, it has become a common practice for large simulations to throw away results from intermediate time steps. To prevent simulation failure at an early stage, in-situ data analysis and visualisation is becoming a necessity to enable domain experts to monitor whether a simulation is running smoothly and to obtain first insight into the resulting data [10][11].

Complicated visualisation algorithms are often unnecessary and time-consuming, but provide a first insight into the on-going simulation. Rather than carrying out conventional post-processing after a simulation is finished, domain experts are eager to use their domain knowledge to steer the visualisation to focus on regions that they consider as important and to identify critical information that might lead to a modification of the next simulation design. Therefore, we consider user interaction as a key component of our system.

For exascale computers, visualisation and post-processing systems and algorithms have to be re-designed to accommodate highly parallel data processing and interactive user investigation. During the CRESTA project, we designed and implemented a distributed, in-situ parallel post-processing system (cf. 4.1.4). Compared to existing solutions, this system is more flexible and powerful, permitting in-situ post-processing with the distributed simulation, supporting on-demand data analysis, and interactive exploration with current instances of simulation data [3].

4.1.1 In-Situ, Live Processing where the Simulation Happens

In-situ (a Latin phrase for on site or in position) processing has become a key concept in exascale data analysis and visualisation. Waiting for a simulation to finish and writing out huge amounts of simulation output is no longer a viable solution for data analysis. Instead, visualisation and data analysis must happen when and where a certain simulation step has been carried out, as the so-called in-situ processing. Our in-situ processing system was an extension of previous work by Virachocha [12], which involved a distributed post-processing system. We extended this system to running simulations, providing on-the-fly data analysis and user interaction to the on-going simulations. To validate the feasibility of our system, we integrated our post-processing software into the HemeLB application [7] (cf. 4.1.5 for details).

4.1.1.1 Sharing Memory with Simulation Processes

To access the simulation data in-situ and perform on-demand filtering, we need to access the main-memory of each computing node. Since the numerical simulation is running as a parallel application, each process holding data has to pass the current simulation data to the cohabitant Viracocha process. A solver specific data extractor utilised by the Viracocha data manager is integrated into HemeLB. To guarantee high bandwidths and low latencies we have chosen to couple both parallel applications in the process-space (in-situ). Therefore, Viracocha is executed concurrently by threading within each solver process. Viracocha is based on the master/slave paradigm where two major types of instances exist. The Viracocha slave is in charge of algorithm execution including data access while the Viracocha master is responsible for receiving filtering requests and scheduling algorithm execution of the Viracocha slaves.

4.1.1.2 Feature extraction on the fly

Once the master receives a filtering request, the simulation is shortly disrupted after the actual iteration to access data snapshots and to apply desired data conversion. Then the simulation proceeds while the snapshot is used by the filtering operation to extract user-defined features. We have chosen a snapshot approach where the simulation is only interrupted for a negligible time and Viracocha does not interfere further with the simulation process. Altogether, this concurrent execution and easily extendable data exchange allow for quick integration into other simulation codes.

4.1.2 Rendering

After the feature has been extracted, we free the simulation master to proceed to the next simulation step. Extracted features will be streamed to a rendering and user interaction front-end, where graphical representation will be rendered, and user interactions are provided. This front-end application can be a lightweight single desktop or an additional cluster system, depending on the given size of the extracted features.

4.1.2.1 Smaller features, single front-end

As mentioned before, we often do not need complicated visualisation algorithms to obtain a first insight into the running simulation. In most cases, showing the simulation mesh, extracting an isosurface, or extracting a cut-plane of the whole data would already be sufficient for domain experts to carry out investigations. In such cases, extracted features result in a minimal amount of data that can fit into the memory of a single machine.

4.1.2.2 Larger features, additional hardware for rendering

In certain cases, extracted features of the simulation step are so large and complicated that a single machine cannot handle these. In order to provide scalable solutions for rendering solutions, we place a scalable parallel rendering system between the feature extraction and the user's front-end. On top of this parallel rendering technique, our developed rendering application makes heavy use of multi-threading per process to prepare the data rendering.

4.1.3 User interaction

Interactive visualisation provides the user with the ability to interact with the provided visual representation, browse through data and make decisions. It is one of the key features in an exascale post-processing system.

4.1.3.1 Hardware setup

Our application allows the user to interact with the in-situ visualisation in a virtual environment. Using a fly-stick in front of a powerwall display, the user is able to perform tasks including sending requests to couple or decouple data filtering, choosing data mapping and rendering algorithms, and navigating through the resulting visualisation. As alternatives, we also support a single desktop as front-end using mouse and keyboard as input devices that send out interaction commands.

4.1.3.2 Benefit of allowing user interaction

Involving human experts in the post-processing loop allows in-depth analysis of the current simulation step. It also enables knowledge-driven data inspections of the data. Using the knowledge of domain experts, decisions and conclusions are made more easily, thus resulting in a more efficient data analysis.

4.1.4 Post-Processing System Architecture

In the following, we briefly describe the system architecture of our post-processing tools. We will elaborate on the system layout in a detailed manner with respect to the HemeLB application in the later sections.

Figure 7 demonstrates a general post-processing system for an exascale system. To avoid moving data around, the visualisation shares the same process as the simulation. At the same location, simulation output will be cached and visualised. A master node will control and collect not only simulation results, but also visualisation output, and send these back to the user front-end.

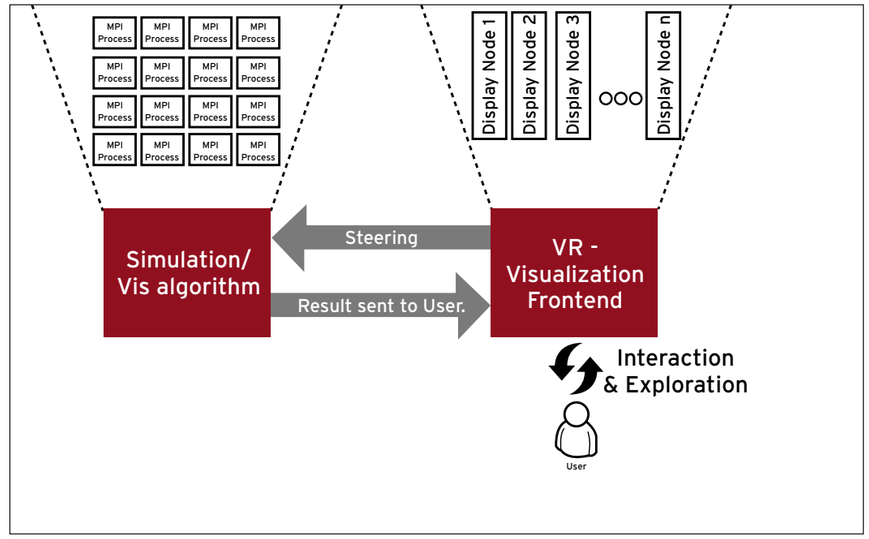


Figure 7: Interactive post-processing system architecture for exascale systems

For the purpose of in-situ monitoring, the user front-end can be just a single display (see Figure 8) that demonstrates the run-time results of the current simulation step. In this type of system, only an image with given resolution is transmitted to the front screen and thus provides a first insight into the running simulation. The advantage of such a set-up is the minimal amount of data moved over the network. It results in low latency between the remote systems and the front monitor.

To inspect the data in a more detailed way, a front-end can also be a more complex system that utilises virtual reality techniques that allow interactive user interaction with the data (see Figure 9). Within this approach, a set of data or images needs to be collected and stored somewhere by the scheduler, which allows for interactive exploration requests sent by the VR (Virtual Reality) front-end.

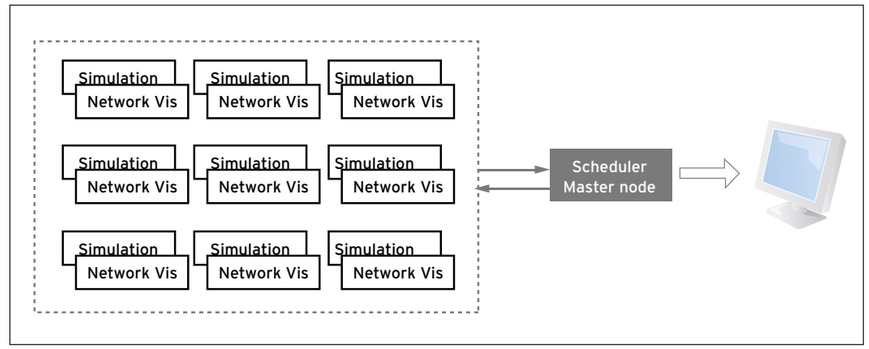


Figure 8: An In-situ monitoring with only a screen showing the network image streamed to the front-end application

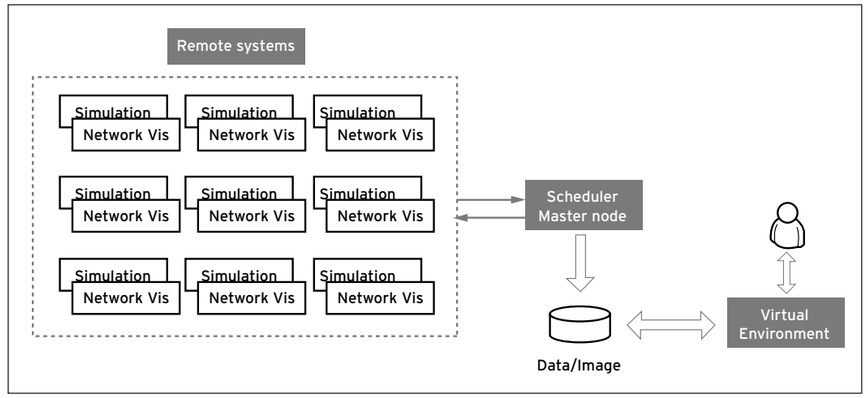


Figure 9: An Interactive post-processing architecture with a user interacting on the front-end

The difference between the two systems is that the former only sends rendered images over networks from the remote system to the front-end and thus minimises data movement. The latter system requires more communication between the front-end and remote systems, but allows in-depth and intuitive exploration of the current simulation time step. For monitoring a rapid running simulation process the former architecture is recommended, while for in-depth analysis of the simulation output the second is more suitable.

4.1.5 Application Study: Co-Design with HemeLB

4.1.5.1 Online-monitoring for a remotely located running simulation

An online-monitoring tool was implemented in the programming language Python which is able to monitor a large simulation that is running remotely on a cluster system without pausing or writing out data to disk (principle cf. Figure 8). For this demonstration, a HemeLB simulation was running on a cluster system in real time, and based on the already implemented volume mapping from HemeLB, the online-monitoring client accessed the produced network image rendered with a resolution of 1024x1024, transferred it to the front-end and displayed it as on the monitoring window.

4.1.5.2 Post-Processing of the HemeLB Simulation

Two aspects of the the online-monitoring tool with the HemeLB simulation were benchmarked, using 4 to 256 cores on a T-Systems cluster at Munich EIP Data Center. First, we benchmarked the performance and time needed to perform one simulation step and generated one network image. Then we measured how the image resolution affected the frame-rates on the front-end.

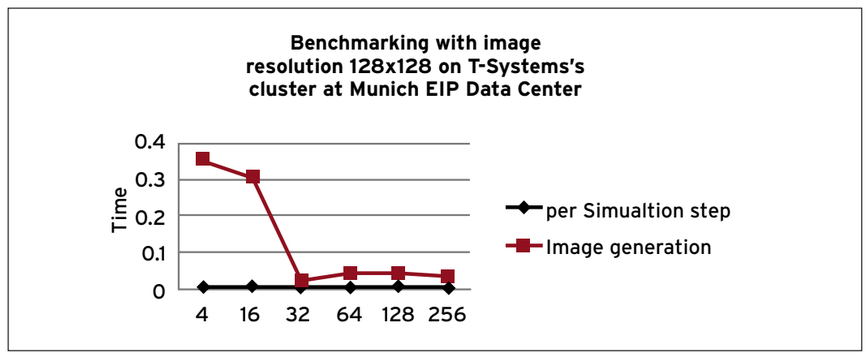


Figure 10: Time measurement for generating image with resolution 128x128

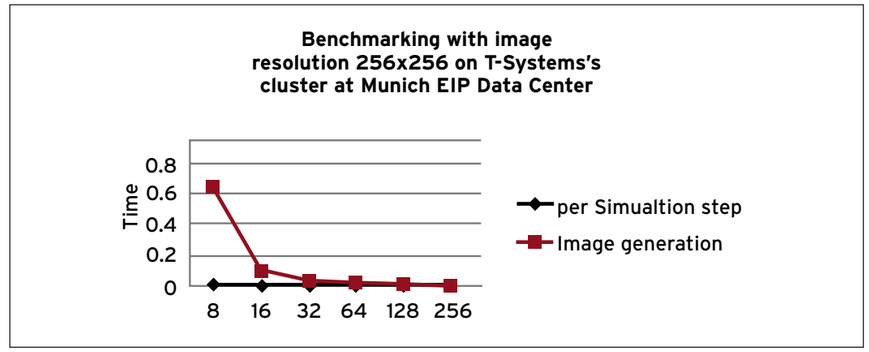


Figure 11: Time measurement for generating image with resolution 256x256

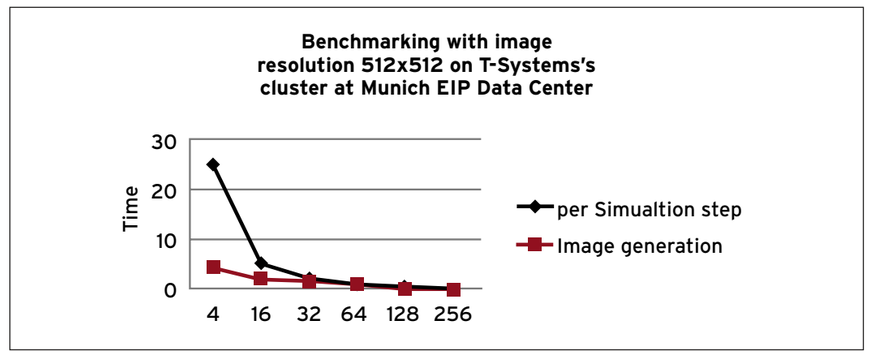


Figure 12: Time measurement for generating image with resolution 512x512

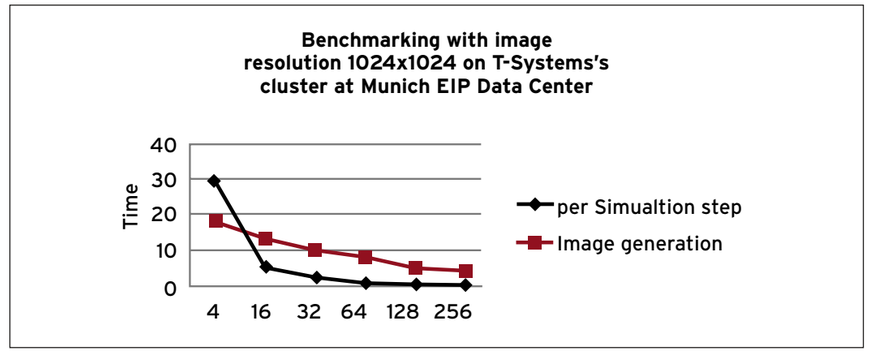


Figure 13: Time measurement for generating image with resolution 1024x1024

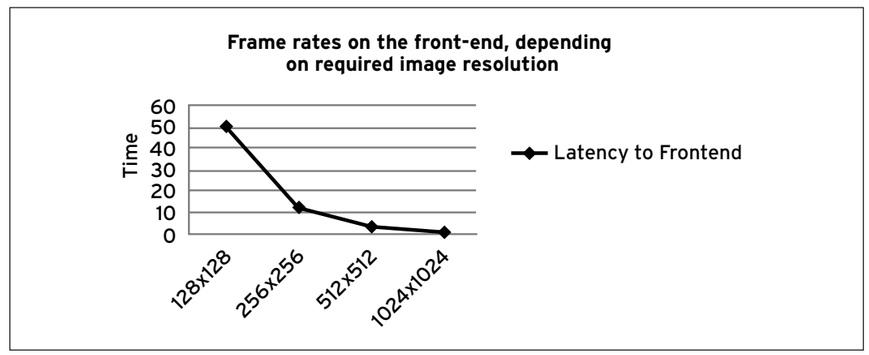


Figure 14: Latency (Front-end frame rates) for image display at front-end

We measured the time that is needed to composite an image for the front-end with image resolution 128x128, 256x256, 512x512 and 1024x1024 pixels, respectively (cf. Figure 10, Figure 11, Figure 12, Figure 13). For each given image resolution, we also measured the latency from remote system to front-end in terms of frame rates, see Figure 14.

Comparing Figure 11, Figure 12, and Figure 13 to Figure 10, the time needed for generating an image increased as the required image resolution increased. At resolution 128x128 and 256x256, the scaling curve for image generation did not decrease dramatically. This is due to the fact that at smaller resolutions, the image generation is quickly finished and the time needed to collect the data as well as communication among cores remains more or less the same. While going to a higher image resolution (Figure 13), we can observe that there is an obvious decreasing trend in the time needed for image generation with more computational cores.

We observe that with more cores, the computation time for simulation and image generation decreases. However, the non-linear decrease is expected due to the fact that, with increasing number of cores, more time is needed to collect the image from each single core and compose them. Moreover, with the increased image resolution that is required by the front-end (online-monitoring client), the frame rates on the front-end decrease (Figure 14).

In the following figure and video (Figure 15), we demonstrate the interactive exploration tool developed at DLR for analysing an aneurysm geometry. In front of a power-wall, the user is able to interact with the aneurysm dataset, seed particles in the blood flow and trace the dynamics of the blood within the aneurysm. The stereo view in front of the power-wall together with an interacting fly-stick enables the user to naturally navigate through the dataset, allowing intuitive and in-depth exploration of the blood simulation output.

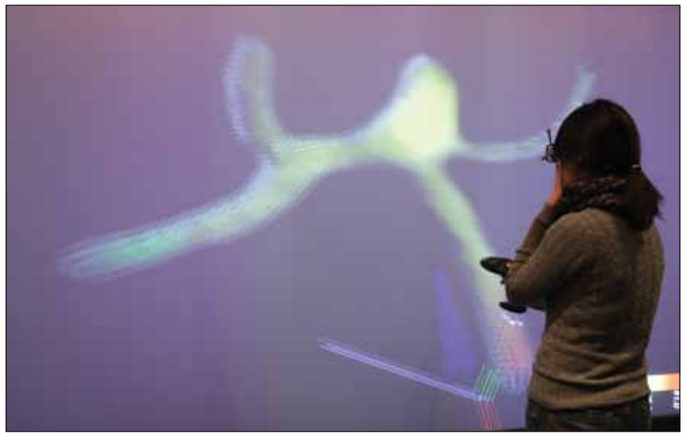


Figure 15: User interacting with the aneurysm data in front of a power-wall. Please contact the authors if you wish to access this video.

4.2 Remote Hybrid Rendering in Exascale Simulation Scenarios

Remote hybrid rendering (RHR) is used to access remote exascale simulations from immersive projection environments over the Internet. The display system may range from a desktop computer to an immersive virtual environment such as a CAVE [13]. The display system forwards user input to the visualisation cluster, which uses highly scalable methods to render images of the post-processed simulation data and returns them to the display system. The display system enriches these with context information rendered locally, before they are shown. RHR decouples local interaction from remote rendering and thus guarantees smooth interactivity during exploration of large remote data sets.

The protocol for RHR only sends viewing parameters, derived from user interaction and head tracking, from client to server, which responds with 2.5D images, which are merged with locally rendered content. This design enables the cooperation of lightweight renderers with display programs that contain most of the application logic and interaction handling. This allows for easy integration of RHR with a multitude of applications that operate on a 3-dimensional domain. The sole requirement is that the application is able to generate colour images together with depth data describing the distance of the visible pixels to the viewer.

4.2.1 Motivation

Output data from simulations can be large. The Institute of Fluid Mechanics and Hydraulic Machinery (IHS) at the University of Stuttgart uses OpenFOAM [14] to simulate the flow in an entire hydro turbine. Based on the estimated requirement for a dependable simulation of about 1 billion nodes for the whole turbine, the size of a single time step is about 1/4 TB. Storing a full turbine rotation with steps of one degree requires about 90 TB. Transferring that amount of data across a high-speed link (10 GigE) for off-line processing on a user workstation would take more than one day – and would require huge amounts of local storage and processing power. This shows that for exascale data the traditional way of transferring the post-processed geometry data to the display system for local rendering is not possible anymore. In comparison, streaming uncompressed HD-resolution (1920x1080 pixels) images at 30 frames/s for a whole day would require less than 15 TB of bandwidth – and the image the user is interested in is available immediately, not just after a lengthy preparatory transfer. Additionally, employing image compression techniques can significantly reduce this amount of data without incurring a visible loss. This technique of transmitting rendered images instead of post-processed data to the display system is called remote rendering. The significantly lowered bandwidth and processing requirements of remote rendering allow for the efficient use of remote compute resources by a much larger user base.

Head-tracked immersive virtual environments, where the rendering is constantly updated according to the user's current head position, require high frame rates and low reaction latencies to achieve a high sensation of presence and to avoid motion sickness [15]. These immersive visualisation environments provide more intuitive ways for specifying the location of regions of interest, cutting planes, seed points for particle traces, or reference points for isosurface extraction than desktop-based systems. We aim to enable users to experience exascale simulations in such immersive environments over the Internet.

To improve frame rate and reaction times, we try to decouple interaction from network latencies as far as possible, but still without requiring the transfer of huge amounts of data to the client. Only extracted features from simulation results are rendered either directly on the simulation host or on a remote visualisation cluster employing scalable methods. "Context information", however, such as essentially static geometry e.g. turbine shapes, interaction cues for the parameters controlling the visualisation algorithms applied on the visualisation cluster and menus are rendered locally, at a rate independent of the remote rendering. As both remotely and locally rendered images are composited for final display, we call this technique "remote hybrid rendering" (RHR) or "hybrid remote rendering" [16]. This compositing usually takes pixel depth into account, but it might also use opacity information. Figure 16 illustrates the differences between a pure remote rendering and a remote hybrid rendering visualisation pipeline.

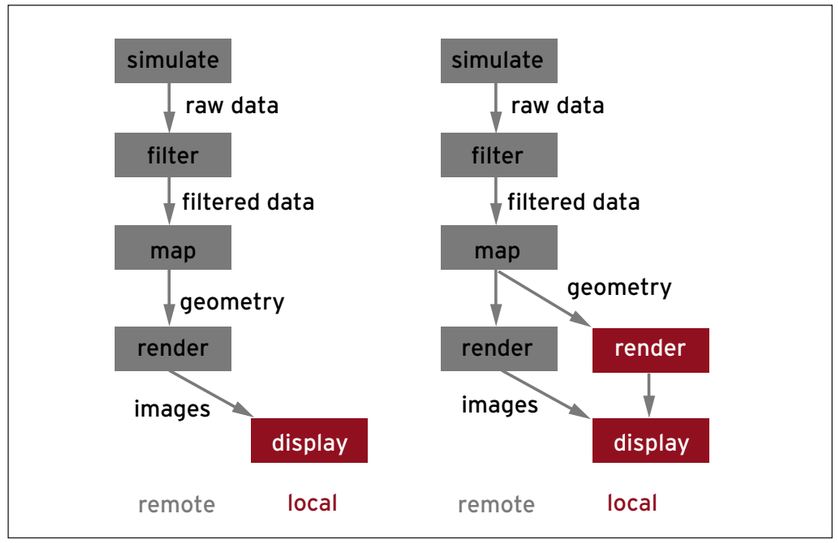


Figure 16: A pure remote vs. a remote hybrid rendering workflow

Figure 17 shows a visualisation of the simulated airflow around a car and illustrates how the image presented to the user results from local context information and remote simulation data. The remote system is used for post-processing the results of the flow simulation and rendering the corresponding visualisations, such as streamlines as well as a plane cutting through the flow field colourised according to air pressure. The local system renders context information. This comprises the menu and interaction elements, e.g. for moving the cutting plane. Also the static geometry of the car is rendered locally. In a final step before displaying the result, locally and remotely rendered images are composited taking into account the distance to the viewer of the geometry object contributing to the pixel's colour: The closer pixel of the two images is copied into the final image.

All the interactive features of the visualisation system are available even though parts of the rendering are delegated to a remote system e.g. new seed points for streamlines can be placed by interacting with the visualisation. Only the fact that the remote parts of the image are updated less frequently makes this visualisation distinguishable from a purely local visualisation.

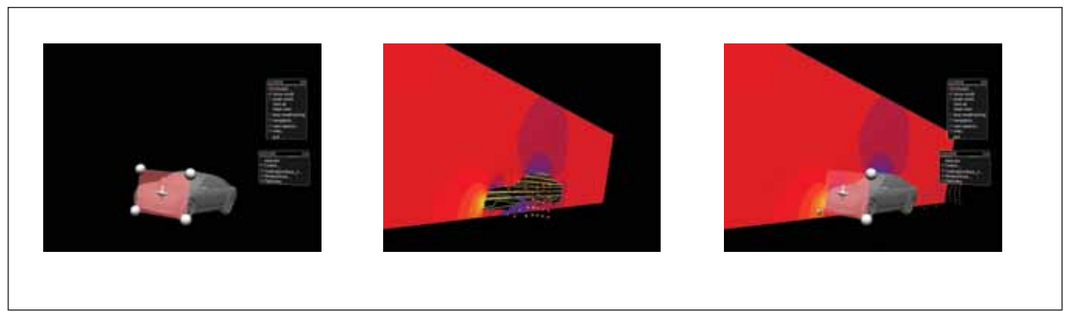


Figure 17: Local context information (left), remote simulation data (middle), fused image shown to the user (right)

4.2.2 Design & Implementation

4.2.2.1 Requirements

4.2.2.1.1 Considerations for Exascale Systems

The environments to which we try to adapt our remote hybrid visualisation system are comprised of the following parts:

- a remote exascale compute resource;
- possibly a remote visualisation cluster, tightly coupled to the compute resource;
- a local display system.

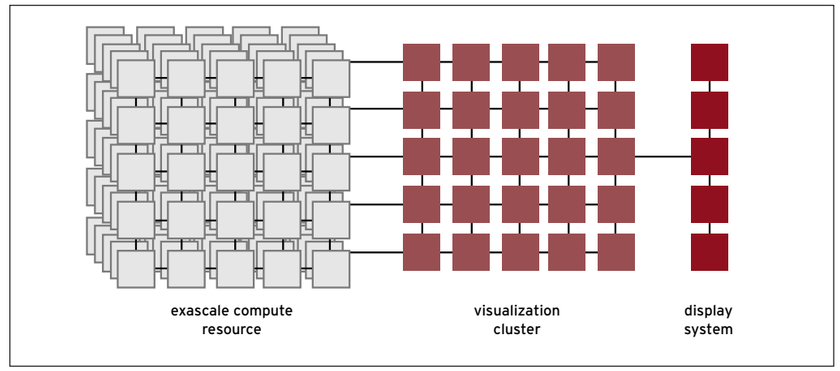


Figure 18: Typical network topology for a remote visualisation task

In some cases, e.g. when there are GPUs inside each node of the exascale system or with CPU based rendering, the compute system and the visualisation system might be the same resource and the GPUs might be used for both simulation and visualisation. For all other cases, we assume a high-bandwidth low-latency link of a quality comparable to the exascale cluster interconnect between compute and visualisation system. The network connection between the remote visualisation cluster and the display system will provide considerably lower bandwidth and higher latency. While it is desirable to have a higher quality link between visualisation and display, this will not always be possible in the case where remote hybrid rendering is used, as this connection will usually be across the Internet.

The network infrastructure might allow for direct connections from each node of the visualisation cluster to each node of the display system, but in the general case the network topology or firewalls prohibit this. Hence, we design our system to cope with a point-to-point connection between the head node of the visualisation cluster and the head node of the local display system. The protocol should keep the number of simultaneous network connections to a minimum; the establishment of a connection should be possible from client to server and vice versa in order to cater for all possible circumstances.

Sort-last [17] has been selected as the method for parallelising the render process, as this allows the rendered to be scaled with the application in a data parallel setting. This means that flat pixel images as present in a framebuffer are the result of the rendering phase. The available data for remote rendering is one colour value including opacity per pixel together with possibly one depth value. Remote sort-last parallel rendering in a system with the described network topology provides the best performance if compositing happens on the visualisation cluster, as this saves bandwidth on the slower link between visualisation and display system. This also ensures that remote hybrid rendering can be composed with scalable rendering methods.

However, rendering context information locally requires a final compositing step in the display system. Depending on the context information to be shown and the rendered data, this requires depth or opacity in addition to the colour information for each image pixel.

4.2.2.1.2 Requirements for Immersive Display Systems

The display system might be a traditional desktop computer. The focus of this work, however, is to enable access to remote exascale visualisations from within immersive projection systems. These are distinguished from desktop systems by:

- input devices which record their position and orientation and input methods which exploit this additional information;
- tracking of the user's head position and continuously updating the rendered image according to the changing point of view (POV);
- 3D stereoscopic imagery, where each eye is presented with an image that is adapted to its position;
- multiple display surfaces, which are used for enhancing the resolution (e.g. in powerwalls, where several screens are tiled in one plane to form a larger display area) or to surround the viewer with images (e.g. in a CAVE, where the sides of a cube around the viewer are used as projection surfaces).

It is sufficient to serve one display system at a time. Such a system, however, might possibly consist of several display surfaces, each of which may be a stereographic display. Updates to different display surfaces have to be synchronised in order to enable correct 3D stereoscopy across all surfaces. This might incur longer latencies, when the images for all tiles are not available at the display client at the same time, but this synchronisation is vital for immersive display environments. With our design, where all data transfer is funnelled through the head nodes of the local and remote systems, synchronisation between the nodes attached to a tiled display naturally happens in the client application. On the other hand, re-projection of 2.5D images [19] according to current viewing parameters automatically brings all tiles into a matching state, so that synchronisation becomes unnecessary.

4.2.2.1.3 Performance Requirements

Communication overhead should be minimal. Network round trips, e.g. waiting for acknowledgement of successful delivery of messages, have to be avoided in order to guarantee good performance. For local area connections, TCP based protocols have proven superior, whereas in wide area networks, UDP based protocols seem to have an advantage [18]. We expect the principal use case to be from within local area networks or within networks providing a similar connection quality, such that we prefer TCP to UDP.

In order to be able to balance visual accuracy with performance, the protocol has to allow for different encodings and compression algorithms. For accommodating changing network circumstances (bandwidth and latency variations), these have to be switchable at run-time. Compression should not visibly decrease image quality for either line drawings or images with huge amount of gradients, e.g. from volume rendering.

The protocol should not hinder the off-load of suitable tasks, like image compression or decompression, to accelerators, such as GPUs. This mostly concerns the image codecs to be used. Hence, we want to allow for the easy addition of new codecs. This also allows for using codecs adapted to the requirements of the processing of the transmitted data on the display system, e.g. when the 2.5D image is re-projected. Additionally, this allows the system to profit easily from algorithmic improvements available in new video codecs, such as H.265 [20], as soon as GPGPU solutions for real-time compression at high resolutions are available.

4.2.2.2 Protocol

The purpose of the protocol for remote hybrid rendering is to define the communication between the visualisation cluster and the local display system. i.e., the protocol for hybrid remote rendering connects the rendering stage to the display stage of the post-processing phase of the visualisation pipeline. The data to be sent comprises viewing matrices, lighting configuration, desired image resolution and current animation step from client to server, which sends colour and depth images in response.

Integrating the remote rendering facility with the application might enable further optimisations, as the application has more knowledge about which data is important. The application might choose to update the significant regions more often or at lower compression level with higher image fidelity. However, as application independence is also a goal of this system, we do not take into account solutions that require tight coupling with the application, such as described above e.g IBRAC [21], or as implemented in Visapult [22] into account.

Based on an assessment of the requirements listed above, RHR was implemented as extensions to the RFB protocol [23], as it allows for backward compatibility with regular VNC clients by building on the extensible protocol implementation LibVncServer [24].

4.2.2.3 Implementation

4.2.2.3.1 Local Display Client

The client for remote hybrid rendering is implemented as a plug-in to OpenCOVER [25], the virtual reality renderer of the visualisation system COVISE [26] and its data-parallel successor Vistle [27], which is currently being developed. It retrieves both colour image and depth data from the server and renders these as an additional node in its scene graph. This achieves compositing of remote and local content. During each frame, the current values of the matrices describing the positions of the user's head and hand are sent to the server. In addition, the results of user interactions, e.g. new seed points for particle traces, are transmitted to the server.

While viewing the colour image generated by a RHR server is possible with any VNC viewer, taking advantage of the compositing of local and remote data requires a specially adapted VNC client.

4.2.2.3.2 Remote Rendering Server

For the RHR server, there are two implementations: one is realised as a plug-in for OpenCOVER. As such, it is compatible with COVISE and Vistle. The other implementation is a lightweight rendering module for Vistle, which uses the CPU for interactive ray casting.

Both server implementations can make use of a cluster of rendering resources by means of sort-last parallel rendering: a complete image is composited from renderings of all parts of decomposed data sets. This requires 2.5D image data (colour and depth) for each partial image. The final image is obtained by selecting the colour of each pixel from the partial image with the smallest corresponding depth value, i.e. that is closest to the viewer. This step is executed by the IceT compositor framework [28], a library that provides highly efficient algorithms for combining images, by exploiting MPI (Message Passing Interface).

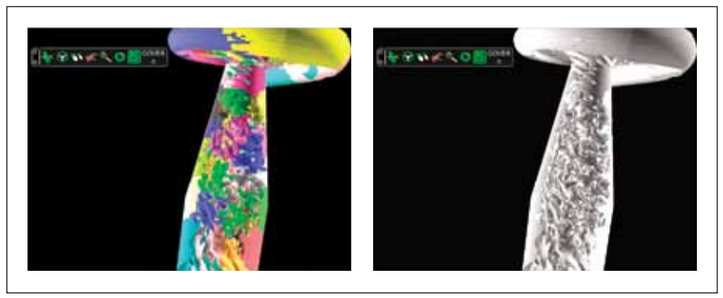


Figure 19: Contribution of nodes in different colours (left) and final composited image (right) of IHS pump turbine test case.

OpenCOVER uses a plug-in for this purpose, while compositing is an integral part of the Vistle ray caster. As the ray caster does not depend on GPU support, it allows scaling with the simulation even when there are no GPUs in the compute nodes. Figure 19 shows the contributions to the final image from individual nodes in different colours together with the final composited image.

The RHR servers provide a full implementation of a VNC server: every VNC client can connect to it and interact with the visualisation with keyboard and mouse. For implementing this functionality, the library LibVNCServer [24] has been used.

For remote hybrid rendering, it has been augmented with the following features:

- transmission of depth data (z-buffer) from server to client for enabling compositing with image contributions rendered on the client;
- reception of 3D viewer and pointer positions sent by client;
- reception of interaction data sent by client.

For compressing depth data the snappy entropy compressor library is used [29]. For CPUs and CUDA capable GPUs, we implemented a method for lossy depth compression similar to DirectX texture compression, which operates orthogonal to the entropy encoding. The development of our own algorithm for depth compression was necessary, as we did not find a high bandwidth compression algorithm for image data with more than 8 bit precision per channel. Although VNC has mechanisms for sending colour images, we added our own extension for sending JPEG compressed image tiles for being able to synchronise colour and depth frames, which is necessary for correct compositing of local and remote images.

When rendering with OpenGL, image data has to be transferred from GPU to CPU memory before compositing. We employ two methods for copying the image data from GPU to CPU: one that relies purely on the OpenGL API call glReadPixels, and another one that employs CUDA for the transfer from GPU to CPU memory. Especially on gaming class hardware, resorting to CUDA provides better performance [30].

The CPU based data-parallel ray casting render module for Vistle is based on the ray tracing framework Embree [31], which makes use of the SIMD units of CPUs to reach interactive frame rates. The sole purpose of this render module is to provide the remote hybrid rendering service. Because of this, a rather lightweight implementation was possible, as most of the application logic resides in the RHR client.

4.2.2.4 Controlling RHR Behaviour

There are several ways of influencing remote hybrid rendering behaviour.

- Data Distribution: The user can choose how to split the data between local and remote systems. On the one extreme, only interaction elements such as menus are rendered locally, while all the simulation data is kept on the remote system. If the local system is powerful enough, a large part of the static geometry can be rendered locally for providing lower interaction latency with these parts of the data.
- Image Quality: Image quality can be traded for bandwidth reduction and higher frame rates. Less precise lossy compression reduces bandwidth requirements. And reducing the resolution of the rendered image both reduces bandwidth requirements while simultaneously reducing the load on the image generation pipeline.
- Composition: For high image fidelity, it is possible to combine the remote image with the local elements in a pose that corresponds to the viewing parameters used during remote image generation. Another possibility is to overlay the remote content with local imagery for the current viewing parameters. A third possibility is to warp or re-project the remote image based on the available depth data according to the current head position, thereby generating the lowest latency for head movements. However, this comes at the cost of holes in the warped surface and polygons that are shaded according to a previous viewer position.

4.2.3 First Experiences

4.2.3.1 Performance of the Prototype

In order to allow for performance measurements, the prototype has been instrumented to collect timing information, compression ratios and image quality metrics. The implementation of the prototype is based on LibVNCServer [24]. Colour image transfer relies solely on what is provided by LibVNCServer. Support for depth was implemented as a VNC extension.

With LibVNCServer's default of using JPEG compression for colour images, transmitting one Full HD frame (1920x1080 pixels) required about 1 MB for both colour and depth images. Frame rate has been measured for Full HD frames when transmitting from one system with a Quadro FX 5800 GPU to another over a QDR Infiniband network. The images received from the remote server have been updated at a rate of about 20 Hz, while the local renderer updated its contents at a rate of ca. 50 Hz. This shows that the goal of decoupling local and remote update rates has been achieved. The low frame rate of 20 Hz is not due to the required bandwidth of about 20 MB/s, but mostly due to the slow depth buffer read-back performance of about 40 Hz on the Quadro FX 5800. Latency has been measured to increase by 0.1s for a Full HD frame.

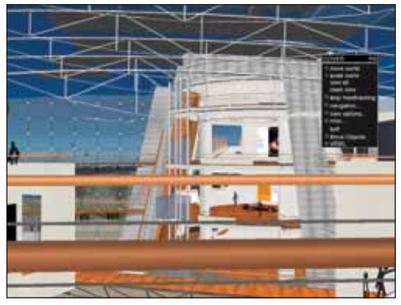


Figure 20: Reference image for depth buffer compression quality assessment.

bits/pixel		2	3	4	6	8
24 bits min/max	compressed size	20.8%	25.0%	29.1%	37.4%	45.8%
	PSNR (dB)	67.7	69.4	77.6	86.6	97.4

Table 1: Compression ratio and quality for lossy GPU based depth compression for the image in Figure 20

Table 1 shows the compression ratios and qualities for the 24 bit depth image corresponding to the colour image shown in Figure 20. The peak signal-to-noise ratio (PSNR) is relatively high compared to codecs for colour images. The visual errors, however, resulting from wrong reconstructed depth values differ from the errors in colour image compression: based on the depth value of a pixel, its colour value is chosen from either the remote colour image or the local rendering. Hence, a pixel is either displayed correctly or in a completely unrelated colour. As these artefacts can appear and disappear from frame to frame, they might be more noticeable than the PSNR suggests. Figure 21 illustrates these artefacts.

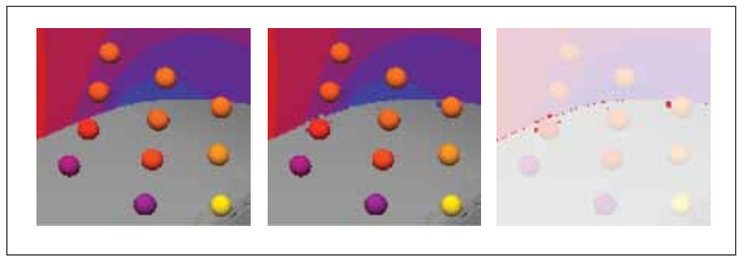


Figure 21: Depth buffer compression quality – left: original image, middle: with compressed depth, right: differences highlighted in red

4.2.3.2 Lessons learned

4.2.3.2.1 Pure Remote Rendering vs. Remote Hybrid Rendering

Classic remote rendering couples a large server application on the remote system to a small display client on the local system. With remote hybrid rendering, this situation is reversed: all the application logic can reside on the local system, and the server application is only responsible for image generation according to updated view points from the client. The result is a lean server, which can be easily integrated with different applications, especially if the application already includes its own renderer. This benefits from massively parallel systems, where the remote application is replicated across many nodes and should make RHR very well suited for in-situ visualisation.

4.2.3.2.2 Choice of RFB as Base Protocol

When designing the system, VNC's RFB protocol seemed to be a good choice as a base protocol for server/client communication. During further development, however, we replaced all parts of the VNC protocol with our own implementation, such that RFB merely served as a transport channel for our own protocol. Fortunately, this only incurs an overhead of one byte per request. Using direct socket communication instead of introducing LibVNCServer as another layer would have allowed for more control of TCP behaviour. However, RFB still provides backward compatibility with regular VNC clients.

4.2.4 Open Challenges

Based on the experience gained with implementing and using the current tool for remote hybrid rendering, we see the following gaps where the software could be improved:

- commonalities between the server-side implementations of RHR (OpenCOVER plug-in and Vistle CPU ray caster) should be identified to reduce code duplication and to provide a basis for integrating RHR support into other rendering software;
- framebuffer read-back using GPGPU methods for retrieving images compatible with compositing for lower PCI Express bandwidth and lower latency should be integrated;
- while the lossy depth compression is a considerable improvement over only entropy based compression, bandwidth requirements and latency could benefit from further improvements of the compression algorithms for depth data, e.g. by exploiting inter-frame coherence.

5 CONCLUSIONS

The discussions presented in this white paper serve as a roadmap for designing pre- and post-processing tools for large-scale simulation codes at exascale. For case studies we used the hemodynamic simulation code HemeLB.

We presented the architectural challenges at exascale pre- and post-processing and sketched a closed-loop-system for the co-design of large-scale simulations and pre- and post-processing systems or libraries. In-situ processing and computational steering will be two major directions when working at exascale. Copying data between a simulation cluster and a dedicated smaller scale visualisation cluster becomes impossible. For numerical simulations at exascale the combination of techniques such as multi-phase load balancing, in-situ processing, computational steering and multi-resolution data structures are crucial to perform an interactive exploration of these simulations.

Regarding pre-processing for exascale simulation, multi-phase load balancing, scalable partitioning, repartitioning and the coupling to fault tolerance frameworks is crucial. Scalable partitioning is still a challenge today.

Interactivity is a main challenge for exascale post-processing, in particular for involving human experts into the post-processing loop. The latter allows in-depth analysis of the current simulation step. It also enables knowledge-driven data inspection.

Remote hybrid rendering (RHR) allows access of remote exascale simulations from immersive projection environments over the Internet. RHR decouples local interaction from remote rendering and thus guarantees smooth interactivity during exploration of large remote data sets. Thus effective RHR techniques are in particular crucial for involving human experts in an adequate way into the post-processing loop and for enabling convenient computational steering.

6 REFERENCES

[1] CRESTA Deliverable 5.1.3, Pre-processing: first prototype tools for exascale mesh partitioning and mesh analysis, will be available at http://www.cresta-project.eu/featured-articles/publications-deliverables/

[2] CRESTA Deliverable 5.1.4, Pre-processing: revision of system, data format and algorithms definition for exascale systems, will be available at http://www.cresta-project.eu/featured-articles/publications-deliverables/

[3] Chen, F; Flatken, M.; Basermann, A; Gerndt, A; Hetherington, J.; Kruger, T.; Matura, G.; Nash, R.W., "Enabling In Situ Pre- and Post-processing for Exascale Hemodynamic Simulations - A Co-design Study with the Sparse Geometry Lattice-Boltzmann Code HemeLB," High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion, pp. 662-668, 10-16 Nov. 2012, doi: 10.1109/SC.Companion.2012.91

[4] ParMETIS, Parallel graph partitioning and fill-reducing matrix ordering, http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview

[5] PTScotch, Software package and libraries for sequential and parallel graph partitioning, static mapping, and sparse matrix block ordering, and sequential mesh and hypergraph partitioning, http://www.labri.fr/perso/pelegrin/scotch/

[6] Zoltan, Data-Management Services for Parallel Applications, http://www.cs.sandia.gov/Zoltan/Zoltan_phil.html

[7] D. Groen, J. Hetherington, H.B. Carver, R.W. Nash, M.O. Bernabeu, P.V. Coveney, Analysing and modelling the performance of the HemeLB lattice-Boltzmann simulation environment, Journal of Computational Science 4 (5), 412-422, 2013, http://ccs.chem.ucl.ac.uk/hemelb

[8] HECToR UK National Supercomputing Service, http://www.hector.ac.uk

[9] ARCHER UK National Supercomputing Service, http://www.archer.ac.uk

[10] K. K. Ma, C. Wang, H. Yu, and A. Tikhonova, "In-situ processing and visualization for ultrascale simulations", Journal of Physics: conference series, 78(1), pp. 12-43, 2007.

[11] K.-L. Ma et al., Next-Generation Visualization Technologies: Enabling Discoveries at Extreme Scale, Davis, CA: SciDAC Review, 2009.

[12] A. Gerndt, B. Hentschel, M. Wolter, T. Kuhlen, and C. Bischof, "VIRACOCHA: An Efficient Parallelization Framework for Large-Scale CFD Post-Processing in Virtual Environments", Proceedings of the ACM/IEEE SC2004 Conference, 2004, doi: 10.1109/SC.2004.66.

[13] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti, "Surround-screen projection-based virtual reality: the design and implementation of the CAVE," Proceedings of the 20th annual conference on Computer graphics and interactive techniques, p. 142, 1993.

[14] OpenFOAM: http://www.openfoam.org/

[15] M. Usoh, K. Arthur, M. Whitton, R. Bastos, A. Steed, M. Slater, and F. Brooks, "Walking > walking-in-place > flying, in virtual environments," SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques, Jul. 1999.

[16] C. Wagner, M. Flatken, F. Chen, A. Gerndt, C. Hansen, and H. Hagen, "Interactive Hybrid Remote Rendering for Multi-pipe Powerwall Systems," in Virtuelle und Erweiterte Realität - 9. Workshop der GI-Fachgruppe VR/AR, C. Geiger, J. Herder, and T. Vierjahn, Eds. Aachen: Shaker Verlag, 2012, pp. 155–166.

[17] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs, "A sorting classification of parallel rendering," Computer Graphics and Applications, IEEE, vol. 14, no. 4, pp. 23–32, 1994.

[18] B. Jeong, J. Leigh, A. Johnson, L. Renambot, M. Brown, R. Jagodic, S. Nam, and H. Hur, "Ultrascale Collaborative Visualization Using a Display-Rich Global Cyberinfrastructure," Computer Graphics and Applications, IEEE, vol. 30, no. 3, pp. 71–83, 2010.

[19] D. Pajak, R. Herzog, E. Eisemann, K. Myszkowski, and H.-P. Seidel, "Scalable Remote Rendering with Depth and Motion-flow Augmented Streaming," Computer Graphics Forum, vol. 30, no. 2, pp. 415–424, 2011.

[20] G. J. Han, J. R. Ohm, W.-J. Han, W.-J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," Circuits and Systems for Video Technology, IEEE Transactions on, no. 99, p. 1, 2012.

[21] I. Yoon and U. Neumann, "IBRAC: Image-Based Rendering Acceleration and Compression," Computer Graphics Forum, Sep. 2000.

[22] E. W. Bethel, B. Tierney, J. Leigh, D. Gunter, and S. Lau, "Using high-speed WANs and network data caches to enable remote and distributed visualization," Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM, Nov. 2000.

[23] T. Richardson, ""The RFB Protocol," realvnc.com, 2010. [Online]. Available: http://www.realvnc.com/docs/rfbproto.pdf. [Accessed: 06-Sep.-2012].

[24] LibVNCServer/LibVNCClient [Online], Available: http://libvncserver.sourceforge.net, [Accessed: 20 Feb. 2013].

[25] D. Rantzau, K. Frank, U. Lang, D. Rainer, and U. Woessner, "COVISE in the CUBE: An Environment for Analyzing Large and Complex Simulation Data," 2nd Workshop on Immersive Projection Technology, 1998.

[26] A. Wierse, U. Lang, and R. Rühle, "A system architecture for data-oriented visualization," Database Issues for Data Visualization, vol. 871, no. 13, pp. 148–159, 1994.

[27] Vistle GitHub repository [Online], Available https://github.com/vistle/vistle, [Accessed: 01 Mar. 2014].

[28] K. Moreland, W. Kendall, T. Peterka, and J. Huang, "An image compositing solution at scale," presented at the High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for, 2011, pp. 1–10.

[29] Snappy – a fast compressor/decompressor [Online], Available: https://code.google.com/p/snappy/, [Accessed: 23 Feb. 2013].

[30] F. Niebling, A. Kopecki, and M. U. Aumüller, "Integrated Simulation Workflows in Computer Aided Engineering on HPC Resources", International Conference on Parallel Computing, 2011, Ghent.

[31] S. Woop, L. Feng, I. Wald, and C. Benthin, "Embree ray tracing kernels for CPUs and the Xeon Phi architecture.", SIGGRAPH Talks, p. 44, 2013.