ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2010, Vol. 39, No. 2

EFFICIENT VISUALIZATION BY USING PARAVIEW SOFTWARE ON BALTICGRID

Arnas Kačeniauskas, Ruslan Pacevič, Andrej Bugajev

Laboratory of Parallel Computing, Vilnius Gediminas Technical University
Saulėtekio Str. 11, Vilnius, LT-10223, Lithuania
e-mail: arnas.kaceniauskas@vgtu.lt

Tomas Katkevičius

Computing Centre, Vilnius Gediminas Technical University
Saulėtekio Str. 11, Vilnius, LT-10223, Lithuania

Abstract. The paper describes efficient visualization performed by using ParaView software on BalticGrid. Development efforts, software implementation details and grid deployment issues are presented. The benchmark based on visualization of poly-dispersed particle systems validates efficiency of the deployed software. Efficiency tests are performed on multi-core architecture. Data reading, glyph generation, CPU and GPU rendering is investigated and discussed.

1. Introduction

Visualization is a powerful tool for analysing data and presenting results, across a wide range of disciplines [6]. Computers are used to create visual images from the data while the human mind is used to make inferences from this imagery in order to better understand the data. Scientific visualization is becoming increasingly important in analyzing and interpreting numerical and experimental datasets. However, large data sets and the complex visualization process require large development efforts and impressive computing resources. Distributed visualization allocates different parts of the machine processing to different computers in order to improve performance. Grid computing [17] represents one of the most promising advancements for modern computational science and visualization.

In the dawn of the visualization era, creating visualizations meant programming using a low-level graphics library. A new approach was brought forward by the Application Visualization System (AVS) [26]. AVS (later called AVS Express) is a modular visualization environment (MVE) providing an application development environment for visualization using a visual network editor. OpenDX [19] is another MVE based on the flexible and universal dataflow model.

A lot of visualization systems and environments are developed by using object-oriented approach of Visualization Toolkit (VTK) [22]. VTK is an open source, object-oriented software system providing a toolkit for 3D computer graphics, image processing and visualization. It consists of a C++ class library, together with several interface layers including Tcl/Tk, Java and Python which can be used to access the classes and build applications. More than 850 separate classes, including several hundred data processing filters, are available in the toolkit. VTK is based on the dataflow model supporting reference counting, which allows data to be shared instead of duplicated. VTK supports portable multithreading for shared-memory implementation and portable distributed parallel processing based on MPI [18].

An open-source, turnkey application ParaView [23], designed for large data visualization using distributed parallel processing, is built on the top of VTK. ParaView Meshless was applied to efficient visualization of SPH (Smooth Particle Hydrodynamics) data [3]. The first ParaView deployment on grid was performed in TeraGrid [24] built by Globus middleware [5].

Visualization systems have become an essential part of the emerging fabric of grid services. However, this leads to very complicated environments handling complex simulation and visualization on remote heterogeneous architectures [17]. The goal of the ICENI [9] project was the provision of high-level abstractions for scientific computing which will allow users to construct and define their own applications through a graphical composition tool. ICENI was being implemented in Java and JINI. The visualization server process the data and provides the output to a renderer (current demos are based on VTK [22]), which can

then send the graphical output back to the visualization client. This can either be done using standard OpenGL remote rendering, or using Chromium [8].

RealityGrid [21] was a project which aims to examine how scientists in the condensed matter, materials and biological sciences communities can make more effective use of the distributed computing and visualization resources provided by the grid. RealityGrid is making use of visualization as part of distributed applications in which the simulation in one place communicates with the visualization in another and a steering client in a third. Because of difficulties experienced in integrating existing MVEs into larger distributed applications, RealityGrid has selected VTK [22] as a lower-level environment, along with enabling technologies such as Chromium [8].

Most of grid environments for visualization [1, 16] are based on the Globus middleware [5] and its toolkit for service development. Visualization software can be highly integrated with working environment. BalticGrid [2] is built on gLite middleware [4]. Only part of Globus functionality can be accessed in the considered grid environment, therefore, most of available visualization software cannot be applied.

The paper presents the first ParaView implementation on grid built by gLite middleware. ParaView software has been adapted for the nature of pilot BalticGrid applications developing special purpose parallel reader for partitioned unstructured datasets stored in HDF5 format. GPU rendering on multi-core architectures has been employed and investigated in real grid environment. The performance analysis reveals how efficiently visualization can be performed on gLite based grid.

2. ParaView architecture

ParaView [23] is an open-source application designed to visualize large datasets. ParaView supports hardware-accelerated parallel rendering and achieves interactive rendering performance via level-of-detail (LOD) techniques. ParaView runs on distributed and shared memory parallel machines as well as single processor PC and has been successfully tested on Windows, Mac OS X, Linux and various Unix workstations, clusters and supercomputers. Under the hood, ParaView uses the VTK [22] as the data processing and rendering engine and has a user interface written using Qt [20].

ParaView is designed as layered architecture. The foundation is VTK, which provides data representations, visualization algorithms, and a mechanism to connect these representations and algorithms together to form a working program. The second layer is the parallel extensions to the VTK. This layer extended VTK to support data streaming and parallel computing. These extensions are currently part of the toolkit. The third layer is ParaView itself.

ParaView is designed as a three-tier client-server architecture. Data Server is the unit responsible for data reading, filtering, and writing. All of the pipeline objects seen in the pipeline browser are contained in the data server. Render Server is the unit responsible for rendering. Client is the unit responsible for establishing visualization. Employing GUI the client controls the object creation, execution, and destruction. These logical units need not be physically separated. In the most common client-server mode, the *pvserver* program is executed on a parallel machine. ParaView client application connects to the server. The *pvserver* program has both the data server and render server embedded in it. The client and server are connected via a socket, which is assumed to be a relatively slow mode of communication, so data transfer over this socket is minimized.

ParaView provides highly configurable GUI based on Qt for the interactive exploration of large datasets (Figure 1). GUI layout is highly configurable, so that it is easy to change the look of the window. The toolbars can be moved around and even hidden from view. Any VTK or user developed filter can be added to ParaView if the user provides a simple XML description for its user interface for its property sheet. ParaView's user interface can be modified and extended both statically, with XML configuration files and dynamically at a run time. One of the most convenient ways to automate ParaView is to use the Python scripting that is built into ParaView. Sometimes it is convenient to automate post-processing and visualization with a Python script that completely bypasses the GUI and any need for user intervention. ParaView comes with a program called *pvbatch*. It can run in parallel without having to establish a slow client/server connection.

ParaView supports large data visualization via techniques that include data streaming, LOD rendering and parallel computing. ParaView supports parallelism using either shared memory processes via threads or distributed memory via MPI. ParaView uses a parallel rendering library called IceT. It uses a sort-last algorithm for parallel rendering. This parallel algorithm allows each process to independently render its partition of the geometry and then composes the partial images together to form the final image. The wonderful thing about sort-last parallel rendering is that its efficiency is completely insensitive to the amount of data being rendered. This makes it a very scalable algorithm and well suited to large data.

3. Software implementation in grid

By default, the ParaView client connects to the server, but sometimes security policies require the ParaView server to be behind a firewall or some other network limiting technology. Configuration like gLite Computing Element denies incoming connection requests and adds challenges to configuring the server to connect with a client.

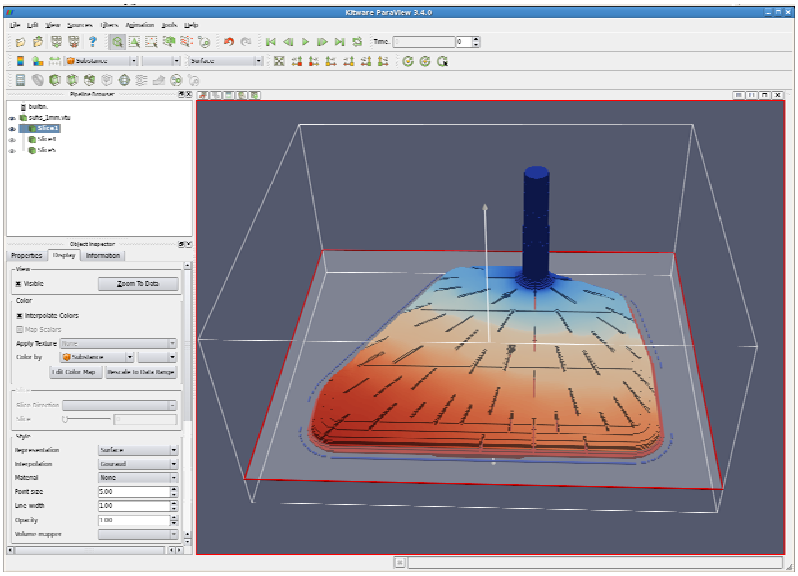


Figure 1. ParaView GUI containing interactive visualization of the industrial oil filter 6HP26

ParaView was implemented in gLite grid by using special client-server communication mode named reverse connection (Figure 2). If the server is behind the firewall, you can reverse the connection direction: the server will connect back to the client. The server is instructed to perform a reverse connection by simply adding the `-rc` flag to its command line:

> mpirun -np 4 pvserver -rc --client-host=host.lt --use-offscreen-rendering

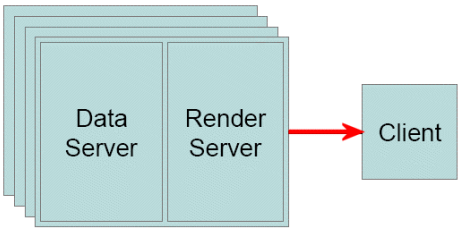


Figure 2. Reverse connection mode of client-server communication employed in grid

Provided command runs ParaView server employing offscreen CPU rendering on 4 nodes and establishes a reverse socket connection between the first node of the server and the client running on *host.lt*. This socket by default is on port 11111.

Two modes of user interaction have been implemented in BalticGrid:

- Interactive mode based on GUI,
- Batch mode employing Python scripting.

User can start interactive visualization session employing full power of GUI and highly interactive widgets. Interactive GUI can run on grid by using client-server communication mode named reverse connection. ParaView program *pvserver*, containing data server and render server, runs on Working Nodes and establishes a reverse socket connection between the first node of the server (rank=0 MPI process) and the client running on a local PC. Alternative batch mode is very attractive for experienced grid users that generate long animations. GUI is disabled and the slow socket communication is replaced by usual grid protocols and utilities for file transfer.

4. ParaView deployment on grid

ParaView 3.6.1 has been deployed in BalticGrid-II [2] in order to provide for numerous users enhanced application services aimed for visualization of large datasets produced in grid. Deployment has been performed by using BalticGrid-II SGM (Software Grid Manager) system. All software packages are installed in the predefined location, which is specified by content of special variable. After successful installation, the SGM marks the site in global grid information system as capable of running this particular application. By use of this flag, called "a tag" the ordinary users may indicate which sites they want to use.

ParaView has been deployed and tested on several sites that are marked by ParaView tags: VO-balticgrid-A-GRAPHICAL-PARAVIEW-3.6.1-OSMESA (CPU rendering) and VO-balticgrid-A-GRAPHICAL-PARAVIEW-3.6.1-DRI (GPU rendering). Required environment can be read from the source file *env.sh*. Installed software as well as the file *env.sh* resides in the predefined directory. ParaView dependencies like Mesa and OpenMPI have been installed by using separate SGM scripts.

The most of grid sites are targeted for CPU rendering, because of GPU installation issues and of the absence of professional high performance graphics hardware. Moreover, it is quite difficult to use GPU rendering and to gain satisfactory performance on heterogeneous multi-core architectures. gLite JDL abi-

lities are not enough flexible for running parallel MPI jobs on multi-core nodes. In general, GPUs can be implicitly shared by pointing multiple processes to the same display on the same host. One problem is that many GPUs will not render correctly two windows on top of each other. The two windows share memory space and clobber each others memory. The *pvserver* assumes that each process has equal access to local rendering. This means that there is no special mechanism to coordinate the rendering between pairs of processes. Thus, GPU rendering can not be directly applied on multi-core architectures of grid clusters.

In order to avoid the described problem, the *--use-offscreen-rendering* flag can be employed. This will create each rendering context in its own offscreen buffer and guarantees that the memory will not overrun that of another rendering context:

> mpiexec pvserver -rc --client-host=host.lt -display :0.0 --use-offscreen-rendering

Provided command executes *pvserver* on the number of cores, which is defined in JDL file. One node has several cores, but only one GPU is available per node. Thus, user launches one process per CPU and lets multiple processes send rendering requests to the same GPU. This option will maximize the speed of filter processing, but will throttle down the rendering speed as GPU processors and buses must be shared. However, the rendering speed can be throttled quite a bit before making a serious impact on visualization performance, even when running interactively.

Special purpose parallel reader is developed for unstructured datasets stored in predefined HDF5 [7] format. It is adapted for the nature of pilot applications that decompose the solution domain into sub-domains, each being assigned to a processor, and ensure load balancing. Each process writes its own result file, which can be read and visualised independently. Developed reader is based on the idea that each process independently reads its file, independently performs visualization and sends resulting image to the rank=0 MPI process. Parallel rendering library composites the partial images together and forms the final image, which is sent to a client. Information on partitioned data files is stored in XML file, which is read by main process running on the rank=0 MPI node. Each process receives the name of data file and other parameters from the main process. Then it works independently until partial image is sent to the main process. Efficiency of performed visualization is completely insensitive to the amount of data being rendered. This makes it a very scalable algorithm well suited to large data.

5. Description of benchmark problem

ParaView is very useful for BalticGrid users performing large distributed computations of actual industrial problems like oil filters, sediment transport, dam break, rail guns, nano-powders, compacting, mixing and hopper discharge [10, 11, 13-15, 25]. For example, porous media flow has been investigated in oil filters. Up to 9 millions finite volumes have been employed for modelling of the real industrial filter 6HP26 (Figure 1). Particle systems have been chosen as a pilot application for visualization, because of large number of particles that are employed modelling and visualizing actual industrial applications [14-15]. Particle systems have no permanent connections or usual grid that can be applied for visualization purposes. Discrete element computations are based on particle positions, forces acting between particles and Newton's laws.

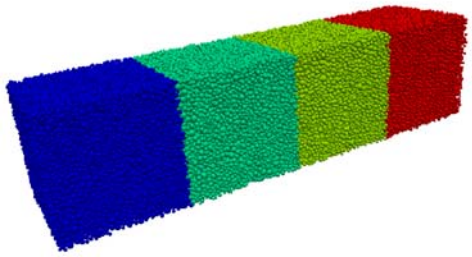


Figure 3. 100037 poly-dispersed particles visualized on 4 processors

Visualization of the tri-axial compaction problem [15] of poly-dispersed particle systems (Figure 3) is considered for performance analysis of ParaView. The three-dimensional computational domain imitates a representative macroscopic volume element containing particles and presents a box in the form of rectangular parallelepiped. Numerical solution of tri-axial compaction helps to evaluate unknown material properties. This problem is very actual and widely solved in the area of material sciences.

The considered benchmark is based on the glyph generation, because particles and computed forces are often represented by glyphs that can be coloured by investigated scalar values or oriented by the examined vectors. The examined poly-dispersed system contains 100037 heterogeneous particles. Meaningful data are composed from the positions of particles and their radius, therefore, the real size of the visualized dataset is quite small (3.13 MB). Numerical results include a lot of values of primary and derived variables that are written in HDF5 [7] files supplemented by XML metadata. Moreover, results of ten iterations have been written to result files. Thus, the size of complete HDF5 file is equal to 235.16 MB. The total size of partitioned result files is up to 236.59MB, which is close to the size of the single file. Particles are represented by generated spherical glyphs (Figure 3). The size of the object, which encapsulates data of generated glyphs, is equal to 326 MB. Rendered polygon mesh consists of 9603552 cells and 5001850 vertices.

The most importand thing is that the size of rendered polygon mesh (glyphs) is significantly larger than the initial data size (Figure 4). Usually, the second order difference is observed, for example 3.13 MB

and 326 MB. It makes the described benchmark very specific and inconvenient for some visualization tools.

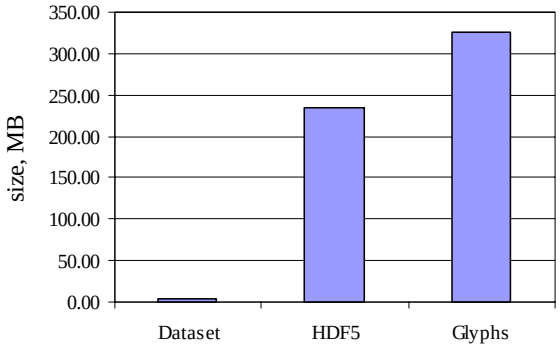


Figure 4. Size of considered dataset

6. Performance tests and discussions

Designing visualization software to run on special purpose hardware always results in high cost supercomputers [6]. These high-end resources are expensive and often based in secure location with limited access privileges. Average consumers, with their actual spending power, are driving developed software and required technology. Visualization must find ways to leverage these requirements, otherwise it will be left aside of the community life. Therefore, heterogeneous grid architecture based on ordinary PC clusters and comodity hardware should be the main target of the research. Efficient visualization performed by using ordinary personal computers reveals challenge, but is highly appreciated in academic communities. The desktop delivered visualisation and grid computing should become solutions to provide sufficient performance visualizing a relatively large dataset using relatively cheap PC clusters connected by a usual network.

The benchmark was performed on BalticGrid testbed ce2.grid.vgtu.lt collected from ordinary personal computers and equipped by high performance GPUs. This cluster was considered for benchmark, because it supported direct GPU rendering and was targeted for visualization purposes. The cluster based on the multicore architecture consists of 14 HP Compaq dc7900 personal computers (nodes) including Pentium(R) Dual-Core CPU E5300 (128KB L1 cache and bus frequency equal to 2.60GHz), 4GB DDR2 RAM 800 MHz, Seagate 500GB HDD (timing buffered disk reads 129.59 MB/sec). Each node is equipped by high performance GPU (Nvidia GeForce 9600GT 512MB 256bit). Nodes are connected to 1Gbps Ethernet LAN by 3Com Baseline Switch 2928-SFP Plus (24 auto sensing 10/100/1000Mbps Base-TX ports).

Performance of ParaView software was evaluated by the measurements of parallel speed-up S_p:

$$S_p = \frac{t_1}{t_p}$$

where t_1 is the program execution time for a single processor; t_p is the wall clock time for a given job to execute on p processors. The benchmark tests were repeated up to ten times and the averaged values were provided in Table 1. Only tests performed in the usual gLite grid conditions, when two processes run on one dual-core node and employ the same GPU, are presented in Table 1. Other cases were also examined in order to perform quantitative comparison.

Table 1. The benchmark results obtained on the grid testbed

	1	2	4	6	8
Reader, s	0.01939	0.01182	0.00744	0.00586	0.00505
Sphere, s	0.00044	0.00047	0.00047	0.00046	0.00046
Glyphs, s	12.90	5.71	2.92	1.85	1.48
Geometry, s	2.20	1.12	0.57	0.38	0.28
MPI(Client), s	0.00025	0.00026	0.00021	0.00025	0.00021
MPI(Server), s	0.00013	0.00014	0.00015	0.00016	0.00016
GPU render, s	5.72	4.69	2.40	1.63	1.24
CPU render, s	20.07	11.46	5.84	3.96	3.03

The main attention was focused on the performance of the developed HDF5 reader, rendering of the resulting polygon mesh and speed-up attained. The first row shows the number of processes. The second row shows the time consumed by the developed HDF5 file reader. The third row presents the run time of *vtkSphereSource* module, which generates spherical particles. This time is negligible comparing to time consumed by other visualization filters. The fourth row provides the performance results of the *vtkPVGlyphFilter*. The fifth row presents the time spent on generating outlines by the *vtkPVGeometryFilter*. The next rows provide data on communication performed by MPI. The sixth row presents time spent by *vtkMPIMoveData* module on the client, while the seventh row presents time consumed by *vtkMPIMoveData* on the rank=0 MPI node of the server. The zero process gathers all results from other processes and sends resulting image to the client. Communication times of other processes are significantly shorter. In general, parallel communication was fast enough and consumed negligible amount of time. The eight row and the ninth row show GPU and CPU rendering time, respectively. Work performed by *vtkPolyDataMapper*, *vtkOrderedCompositeDistributor*, *vtkPVUpdateSuppressor* is not investigated, because it takes about 10^{-4}s or even 10^{-5}s.

The data transfer between the remote parts of the distributed visualization environment was also considered. HDF5 files were transferred from the Storage Element to Working Nodes by using LFC means. This process lasted from 8.6 s to 17.08s. The consumed time linearly depended on the number of partitioned files and their total size, which varied from 235.16MB to 236.59MB. In average, job submission to the grid testbed took about 15.7s. It is worth to note that job submission and data transfer in grid environment takes quite significant amount of time. The performance analysis of gLite Resource Broker and LFC does not

fall to the scope of current research, therefore, investigation was not performed.

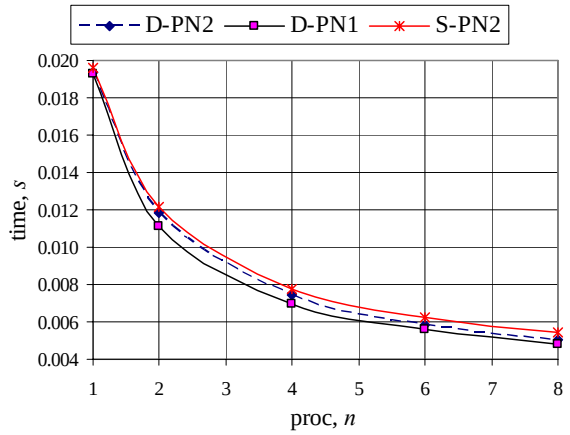
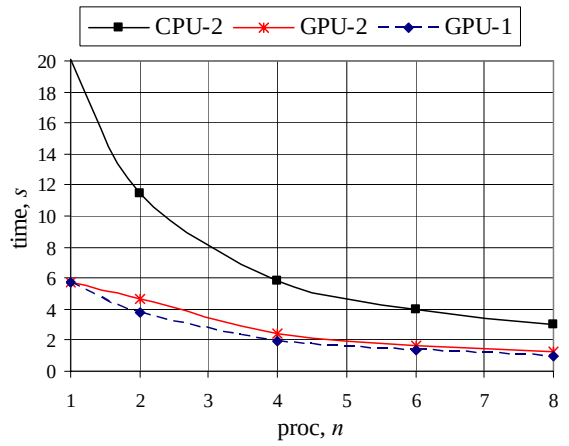


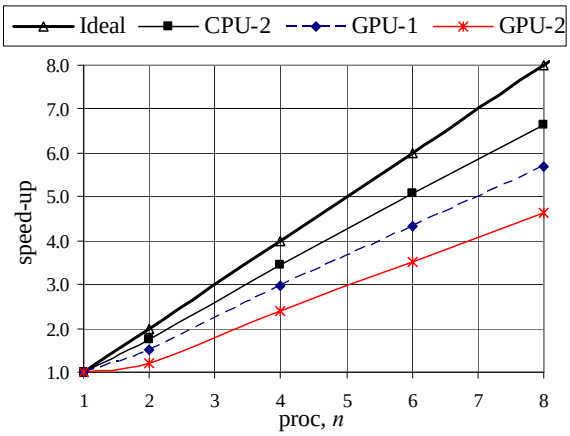
Figure 5. Execution time of the developed parallel reader for partitioned HDF5 files

Figure 5 shows the execution time of parallel HDF5 reader for different number of processors. Shared home systems are considered as well as distributed home clusters. Before averaging, tests were repeated more than one hundred times. The curves D-PN2 and S-PN2 represent data reading on distributed home and shared home, respectively. In both cases, two processes run on one double-core node and use the same hard disk drive. The curve D-PN1 represents data reading from distributed home by one process per node. The best performance was measured in this case, that was expected. However, the difference measured is not significant. It can be explained by rather small data size and complex inherited structure of all ParaView readers. In general, the developed reader demonstrated good parallel performance in real grid environments based on distributed or shared home systems (curves D-PN2 and S-PN2).

Figure 6 illustrates rendering performance employing CPU as well as GPU. Figure 6a shows rendering time, while Figure 6b illustrates speed-up S_p measured. The curves GPU-1 and GPU-2 represent GPU rendering performed by one process per node and two processes per node, respectively. The curve CPU-2 represents CPU rendering performed by two processes running on one dual-core node. The special curve Ideal shows ideal speed-up. Rendering time of two processes sharing one GPU (the curve GPU-2) is not significantly different from that of two processes employing separate GPUs (the curve GPU-1). However, measured speed-up S_p of GPU-2 is significantly lower, which leads to bad scaling on larger number of processors. In general, GPU rendering significantly outperforms the CPU rendering for relatively small number of processors. However, speed-up S_p of CPU rendering is higher, which gives advantage employing large number of processors.



a)



b)

Figure 6. Rendering performance: (a) rendering time; (b) parallel speed-up

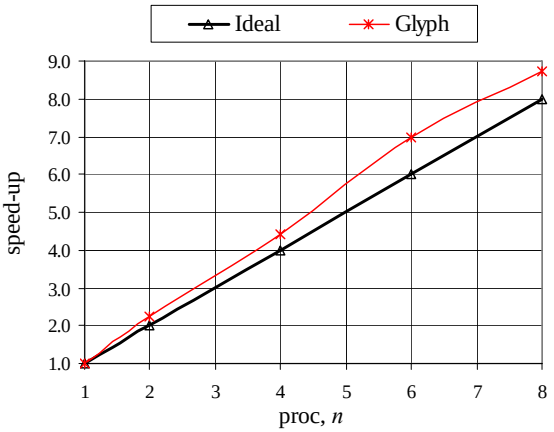


Figure 7. Parallel speed-up of *vtkPVGlyphFilter*

Figure 7 demonstrates the parallel speed-up of glyph generation. *vtkPVGlyphFilter* requires long execution time (Table 1), but it does not need any inter-processor communication. Thus, even super-linear speed-up caused by advantageous cashing can be obtained (Figure 7). However, it was observed that other filter *vtkPVGeometryFilter* runs efficiently for

problems without complex topologies. In other cases, any significant speed-up can not be achieved increasing the number of processors [12].

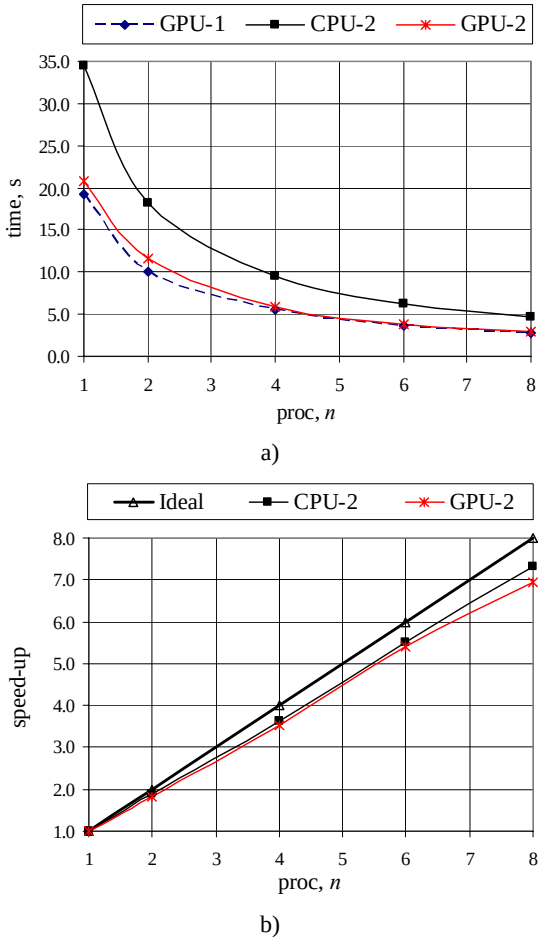


Figure 8. Total performance of visualization: (a) execution time; (b) parallel speed-up

Figure 8 illustrates the parallel performance of the whole visualization process employing GPU rendering or CPU rendering. Figure 8a shows execution time, while Figure 8b provides speed-up measured. The total execution time of visualization employing GPU is shorter than that of using CPU rendering. However, parallel speed-up of visualization employing GPU rendering is lower, therefore, the total time of visualization becomes significantly closer when 8 processes are used. It can be concluded that Figure 8b proves high speed-up of visualization performed on grid testbed based on multi-core architecture.

6. Conclusions

In this paper, deployment and performance of visualization software ParaView on BalticGrid infrastructure was investigated and described. Reverse connection enabled implementation of fully interactive user communication mode in grid built by gLite midleware. ParaView software was sucsesfully deployed on BalticGrid testbed collected from ordinary dual-core computers equiped by GPUs. The benchmark based on visualization of poly-dispersed particle systems illustrated high efficiency of the deployed software. The developed reader for partitioned unstructured HDF5 files demonstrated good parallel performance in real grid environment. GPU rendering on multi-core architectures significantly reduced visualization time. However, measured speed-up of GPU rendering was not high enough, which limited employing large number of processors. Performed speed-up analysis revealed that deployed ParaView software is well designed for distributed visualization of considered datasets.

Acknowledgement

The work described in this paper is supported by the European Union through the FP7- INFRA-2007-1.2.3: e-Science Grid infrastructures, contract No 223807, project “Baltic Grid Second Phase (BalticGrid-II)”.

References

[1] **A.A. Ahmed, M.S.A. Latiff, K.A. Bakar, Z.A. Rajion**. Visualization Pipeline for Medical Datasets on Grid Computing Environment. *Proc. of* 5*th Int. Conf. on Computational Science and Applications, IEEE Computer Society Press*, 2007, 567–575.

[2] **BalticGrid:** *http://www.balticgrid.eu/*.

[3] **J. Biddiscombe, D. Graham, P. Maruzewski.** Visualization and Analysis of SPH Data. *ERCOFTAC Bulletin, SPH special edition*, 2008, *Vol*.76, 9–12.

[4] **gLite:** *http://glite.web.cern.ch/glite/*.

[5] **GLOBUS:** *http://www.globus.org/*.

[6] **C.D. Hansen, C.R. Johnson.** The Visualization Handbook. *Elsevier*, 2005.

[7] **HDF5:** *http://hdf.ncsa.uiuc.edu/products/hdf5/*.

[8] **G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P.D. Kirchner, J.T. Klosowski**. Chromium: A Stream Processing Framework for Interactive Rendering on Clusters. *ACM Transactions on Graphics*, 2002, *Vol*.21(3), 693–702.

[9] **ICENI:** *http://www.lesc.ic.ac.uk/iceni/index.html*.

[10] **A. Kačeniauskas.** Parallel GMRES Solution of Convective Transport Problems on Distributed Memory Computers. *Information Technology and Control*, 2004, *Vol*.30(1), 69–73.

[11] **A. Kačeniauskas**. Development of Efficient Interface Sharpening Procedure for Viscous Incompressible Flows. *Informatica*, 2008, *Vol*.19(4), 487–504.

[12] **A. Kačeniauskas.** Report on Performance Analysis of Sequential and Parallel Visualization in Grid Environment. *Deliverable DJRA*1.1 *of FP7 BalticGrid*-II *project, August* 2008.

[13] **A. Kačeniauskas, R. Kutas**. Implementation of stress dependent boundary conditions in FEM code for coupled problems. *Information Technology and Control*, 2008, *Vol*. 37(1), 69–74.

[14] **A. Kačeniauskas, R. Kačianauskas, A. Maknickas, D. Markauskas**. Computation and Visualization of Poly-Dispersed Particle Systems on gLite Grid. *Civil-Comp Proceedings*, *Vol*.90, ISSN 1759-3433, *Proc. of* 1*st International Conference on Parallel, Distributed and Grid Computing for Engineering* (*Eds. B.H.V. Topping and P. Iványi*), ISBN 978-1-905088-28-7, *Civil-Comp Press, Stirlingshire, United Kingdom*, 2009, 1–18.

[15] **R. Kačianauskas, A. Maknickas, A. Kačeniauskas, D. Markauskas, R. Balevičius**. Parallel Discrete Element Simulation of Poly-Dispersed Granular Material. *Advances in Engineering Software*, 2010, *Vol*.41(1), 52–63.

[16] **D. Kranzlmuller, G. Kurka, P. Heinzlreiter, J. Volkert.** Optimizations in the Grid Visualization Kernel. *Proc. of the Workshop on Parallel and Distributed Computing in Image Processing, Video Processing and Multimedia, IPDPS* 2002, *Ft. Lauderdale, Florida*, 2002.

[17] **M. Li, M. Baker.** The Grid: Core Technologies. *Wiley*, 2005.

[18] **MPI:** *http://www-unix.mcs.anl.gov/mpi/*.

[19] **OpenDX:** *http://www.opendx.org/*.

[20] **Qt:** *http://trolltech.com/products/qt/*.

[21] **RealityGrid**: *http://www.realitygrid.org/*.

[22] **W. Schroeder, K. Martin, B. Lorensen.** Visualization Toolkit: *An Object-Oriented Approach to* 3*D Graphics,* 4*th Edition. Kitware. Inc.*, 2006.

[23] **A. Squillacote.** ParaView Guide, Version 3. *Kitware, Inc.*, 2008.

[24] **TeraGrid:** *http://www.teragrid.org/userinfo/data/vis/pv_overview.php*.

[25] **L. Tumonis, M. Shneider, R. Kačianauskas, A. Kačeniauskas**. Structural Mechanics of Railguns in the Case of Discrete Supports. *IEEE Transactions on Magnetics*, 2009, *Vol*.45(1), 474–479.

[26] **C. Upson, T.A. Faulhaber, D. Kamins, D.H. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, A. Van Dam.** The Application Visualization System: a Computational Environment for Scientific Visualization. *IEEE Computer Graphics and Applications*, 1989, *Vol*.9(4), 30–42.

Received December 2009.

DOI: 10.5755/j01.itc.39.2.12297