Building a Virtual Cluster for 3D Graphics Applications

Alexander Bogdanov, Andrei Ivashchenko(✉), Alexey Belezeko, Vladimir Korkhov, Nataliia Kulabukhova, Dmitry Khmel, Sofya Suslova, Evgeniya Milova, and Konstantin Smirnov

Saint Petersburg State University, 7/9 Universitetskaya nab., Saint Petersburg 199034, Russia
aiivashchenko@cc.spbu.ru
http://spbu.ru/

Abstract. This paper discusses a possible approach to distributed visualization and rendering system infrastructure organization, based on Linux environment with the usage of virtualization technologies. Particular attention is paid to the minutiae, which may be encountered due to the environment setup and exploitation processes, and may affect system performance and usability. Some applications and development tools are studied, as they can provide a rapid onset of computing resources exploration.

Keywords: Computer graphics · Virtualization · Distributed rendering · Remote workstation

1 Introduction

Modern scientific studies not only actively exploit information technologies, but already rely on them. Due to the significant growth of domains involved in a single simulation in general, and detailed description of each involved process in particular, proportionally to the capabilities of the newest computational resources, the amount of data that should be processed and generated has greatly increased. Such complex data sets are very difficult to consider on their own, so various visualization techniques are applied to represent them in a more understandable form to the researcher. This task could become even more complicated if real-time data processing and exploration is expected. And if enormous calculations in most cases are only a matter of time, visualization of extra-large data volumes is still unresolved.

This paper discusses one of the ways to organize an environment for various tasks connected with real-time and batch rendering, based on usage of virtualization technologies. Also a set of tools and various applications that could help to scale out an existing application, or to develop a new one, is presented here.

O. Gervasi et al. (Eds.): ICCSA 2016, Part II, LNCS 9787, pp. 276–291, 2016.
DOI: 10.1007/978-3-319-42108-7_21

2 GPU Delivery to the Virtual Environment

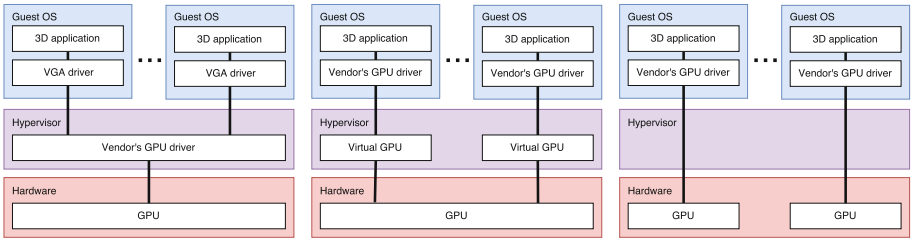
Of course, usage of virtualization approach for organizing computational environment will bring some overhead in resource consumption, but sometimes there is no other choice. This might be necessary if you are planning to share resources with other users, or this might be essential if you are getting the resources as a service this way. Since you have to use such a solution, you can try to extract some benefits out of there. For example, different virtual machine images and templates could be prepared for rapid bootstrapping of specific task-related environments. Some research, analysis and evaluation on organizing computing and networking resources as a virtual supercomputer tuned to particular application requirements have been addressed in [25,28,29]. Another case is also trivial: during working hours computing resources could be occupied by users' virtual machines providing remote access to the workspace and being used interactively, but at other times dormant computing capacities could be exploited to execute some heavy batch routines, like rendering with usage of ray tracing techniques, if we are talking about computer graphics related tasks.

Whereas the question about common virtual resources delivery is someway trivial due to amount of both open source and commercial virtualization solutions, the problem of supplying a guest environment with powerful graphics acceleration, especially for Linux-type operating systems, remains an open question.

The main idea of GPU virtualization is to be able to schedule GPU calls coming from a virtual machine the same way CPU calls are scheduled, and there are three main approaches presented in Fig. 1 which could be provided by virtualization platform to achieve the goal.

Virtual Shared Graphics Acceleration (vSGA) is probably the earliest technology introduced in the field of graphic resources sharing. This method requires an installation of vendor's graphics driver at the hypervisor layer, which should communicate with special VGA driver on virtual machine, as it shown in Fig. 1a. The weakest point of this solution is the presence of substantial constraints with regards to the graphics standards and platform support. Essentially, it means that vSGA could be used only with those operating systems that are supported by virtualization provider, and applications are able to run only with particular graphics API implemented for custom driver (e.g. VMWare implementation support is limited up to DirectX 9.0c and OpenGL 2.1) [36].

Virtual GPU (vGPU) is a more advanced technology available right now in that area. Right now there are only two virtualization platforms that are providing support for fully functioning GPU virtualization: XenServer by Citrix and vSphere by VMWare. VMWare ships it's products only on a commercial basis completely packed. XenServer itself is a free to use and open source virtualization server, but features of our interest are available only in the Enterprise Edition. Without a proper license GPU will be working only in a passthrough mode. Moreover, GPU virtualization should be backed by the manufacturer and physical card itself.



(a) vSGA technology (b) Virtual GPU (c) GPU Passthroug

Fig. 1. Ways of obtaining a hardware graphics acceleration on virtual machine

The main idea of vGPU is represented in Fig. 1b. The appropriate graphics driver installation here required on both hypervisor and guest operating systems. The hypervisor operates a virtual GPU manager software, which is responsible for providing a direct access to the virtualized part of hardware. The user's virtual machine should be running an original graphics driver, so mostly all features become available for the applications [14].

Still, there are some things that are working in a passthrough mode, but not available here due to technology limitations with virtual GPU or GPU sharing:

– OpenCL expression evaluation
– Physics computation with GPU acceleration
– Hardware accelerated tessellation

These restrictions are applied because graphics computing resources are not delegated straightly to the virtual GPU and could be reserved at task allocation stage. Instead, users have to use GPU computational resources in a time share manner.

If the available graphics hardware does not support virtual GPU technology, or if there is just no need to use one, generally, any virtualization solution combinations could be involved, since they support direct hardware passing to the guest virtual machine. The appropriate technology for that type of graphics acceleration delivery could be called as Virtual Dedicated Graphics Acceleration (vDGA), or just GPU passthrough [14]. The main positive aspect offered by this solution is that there is no need of installing any additional drivers or managing software on the host machine's operating system, because in that case the hypervisor is just connecting a native PCI bus device, as it shown in Fig. 1c. Since the original graphics driver could be installed to the guest operating system, the end user can be satisfied with the entire spectrum of functionality that has been incorporated by the graphics card manufacturer.

The approaches described above could be involved not only for the discrete graphics cards, but also for embedded system-on-chip solutions. Intel introduces its own Graphics Virtualization Technology standard (Intel GVT) for Intel HD Graphics coupled with processor [5]. The KVMGT and XenGT implementations are available for respective hypervisors. All three resource delivery

technologies described above are available and defined with suffixes, where Intel GVT-d stands for vDGA, -s relates to vSGA and -g is for vGPU.

3 Hardware Configuration and Infrastructure Organization

To perform the installation and deployment of the graphics cluster we used three powerful Dell PowerEdge R720 workstations with the following configuration:

- 2 Intel Xeon E5-2695 v2, 12 cores/24 threads @ 2.4 GHz
- 256 GB DDR3 RAM
- 2 NVIDIA GRID K2 GPU
- Two-way 10 Gbit Ethernet for intranet organization
- External 1 Gbit Ethernet interface

PowerEdge unit itself is a certified solution for virtual desktop infrastructure organization [13]. The NVIDIA GRID K2 graphics card equipped with the server is also positioned at the market as a specialized hardware for the mentioned purposes [15]. Basically, each of NVIDIA GRID K2 modules holds on board two dedicated Kepler GPU's. Each of them has:

- GK104 745 MHz main GPU chip
- 4 GB of 2.5 GHz 256-bit GDDR5 memory
- 1536 CUDA cores
- Up to 4 displays support with 2560x1600 dimension

If the capabilities of each chip were considered individually, it could be said that it comes very close to the desktop GeForce GTX 680 solution. There are two main differences that could be noted between these two cards: each chip of GRID GPU has twice as much memory, but the core and the memory work at a lower clock rate (745 MHz against 1058 MHz for the core, and 1250 MHz against 1502 MHz for the memory) [3,15]. As for the virtualization capabilities, the whole board can handle the following virtual GPU profiles:

- 2 GRID K280Q with 4 GB of memory (passthrough mode)
- 4 GRID K260Q with 2 GB of memory
- 8 GRID K240Q with 1 GB of memory
- 16 GRID K220Q with 512 MB of memory

The good news is that virtualization profiles could be mixed; there are no constraints on using the same, if one is already activated.

VMWare vSphere platform supplied with ESXi hypervisor was used to execute virtualization tasks, since the proper license with GPU virtualization support was available for that package. All three server machines were consolidated into a single unified resource pool that allowed us to redistribute computing facilities on the fly. One of the nodes was involved to create a 3D accelerated gateway to the cluster cycling with the K220Q profile. In general, the four basic

templates for future virtual machines have been created, where each one stands for its own type of vGPU profile. Each of these machines is expected to receive a proportional part to the amount of node's computational resources by default: 16^{th} part for K220Q profile, 8^{th} part for K240Q profile, and so on. Finally, eight computational nodes could be obtained with a passthrough mode, that enabled a whole functional stack with the following configuration:

- 6 CPU cores on a single socket 12
- 64 GB RAM
- GK104 based GPU with 1536 CUDA cores and 4 GB of memory

Such amount of RAM could be explained by a relatively small GPU memory capacity, and, therefore, the necessity of active read-back usage. The bigger amount of data should be placed on the scene, the larger fast memory area should be allocated. In a case of the need for slow storage volumes treatment, the user will experience a massive slowdown of application execution, and also a frame drop if it is a real-time application. Proper caching must be considered carefully while developing such applications.

While the platform is expected to be used for computational purposes, a high performance mode should be enabled for the hypervisor. Another tweak that should be applied is platform specific and implies power management delegation to operating system. These changes provide a correct way to tune the wattage by hypervisor on corresponding virtual machine request. Even those two small and simple attunements allow us to improve performance of synthetic single GPU benchmarks, like FurMark, up to 30 percent [17]. These improvements are true for both Citrix and VMWare virtualization platforms [4,18].

4 Preparation of Computing Environment

This section discusses the preparation stages that should be carried out on the computing and gateway nodes, which are represented with virtual machines in our case, since we are working with a virtual cluster. The initial steps include installing and configuring the video card driver, the X server and the desktop environment. Since we work with NVIDIA hardware, the description includes some context-related stages.

Before beginning the GPU driver installation process, it is necessary to ensure that the open-source implementation called nouveau is not installed, since it can bring some issues [6]. In most cases, this package comes by default with the operating system distribution if it ships with graphical subsystem preconfigured. A problem that can arise is that the operating system kernel can boot correctly only one load module responsible for the graphics hardware usage, and the second one will be just ignored. Additionally, nouveau could be put to the kernel's modules blacklist to ensure it will not be loaded.

Driver installation is trivial, and it could be useful to install some packages that are included into the CUDA Toolkit [2]. The performance analysis tools shipped with this bundle help to show the consumption of GPU resources by

the applications using hardware 3D acceleration. After the installation a proper X server configuration for the headless usage should be obtained. It could be done by nvidia-xconfig utility. Sometimes you would need to set the bus id of your GPU explicitly in Xorg configuration file, because there always will be two VGA controllers shown in lspci. One will stand for the real graphical unit, and another will show the virtual device provided by the hypervisor.

There is only one condition that should be noted while choosing a desktop environment: it should not require 3D acceleration because the remote session can not provide proper functioning in that mode due to several reasons, which are described below.

While preparing the computing environment it is necessary to pay extra attention to the several major issues, which have the greatest impact on system's performance and usage experience.

The first issue that should be solved arises due to the nature of the X Window System used in Linux. X server itself is network transparent and allows to execute render routines remotely, even over the SSH channel. But, actually, all graphics computations will be processed on client machine, so it is just makes no sense. In a case of the X proxy server usage, a dedicated X server is started for the remote session without hardware acceleration available. One would think that everything is fine, because X server supports off-screen rendering, but the rendered frames will not be displayed on the remote desktop. So, an appropriate solution should be found, which will allow to achieve a server-side hardware accelerated rendering combined with the remote graphical access.

With the X proxy session initiated the problem itself is expressed in the inability to load the GLX extension, which is responsible for X-delegated rendering command execution and graphics context initialization for the application. The first and the easiest solution that could be applied is to force the application to use an implementation of OpenGL specification called Mesa 3D, which could be used in software mode with GLX emulation enabled [12]. So, it means that all graphics operations will be performed by CPU. In a normal case Mesa could be used just as a regular OpenGL API implementation backed by GPU, but if it were possible there were no problems here. However, it does not solve the issue in full, because software rendering is not an option if we are looking forward to heavy graphics.

A more appropriate solution here would be the VirtualGL library that allows to display the framebuffer produced on another X server in the active session [23]. Functioning mechanism is presented in Fig. 2. The bootstrapping tool called vglrun injects the library on the application startup and passes the control of GL commands to it. After the frame rendering is done, it is received by the client part of VirtualGL service and appears in a local framebuffer, which could be displayed already. So, the X server responsible for the remote session is performing only 2D manipulations connected with desktop workflow.

For optimization reasons VirtualGL adds all resulting frames into processing queue and transmits them only if the client is able to accept them. To prevent the backlogging unprocessed frames are destroyed and replaced with new ones.

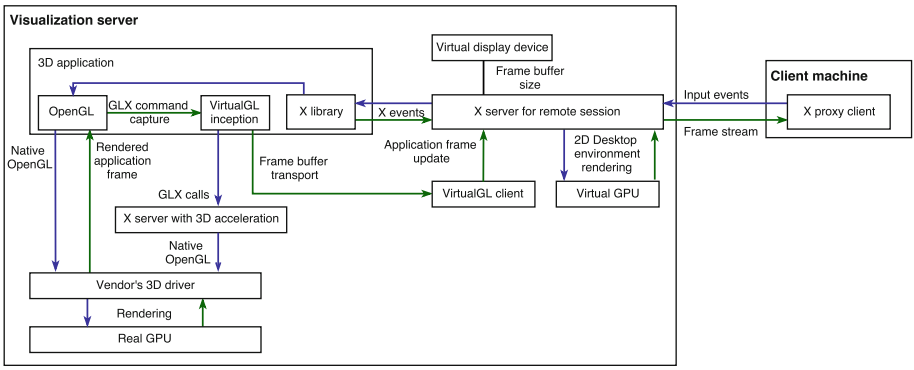


Fig. 2. VirtualGL rendering pipeline

This improves the responsiveness of interactive applications. But it is should be taken in mind that for benchmarking a frame spoiling must be switched off, as the maximum frame rate is expected as a result.

VirtualGL's toolset has several features that could be useful in such type of environment. An access to the 3D accelerated X server could be restricted to the particular user group on configuration setup. The client part of VirtualGL could be launched not only on the rendering machine, but also on the client side, so the rendered frame would be available in a local buffer immediately, avoiding an X proxy usage, but requires an extra network throughput.

It is worth noting that Windows operating system avoids all these problems, since it includes RemoteFX, which stands for a toolset that implements remote desktop access protocol specially prepared and actively exploited by vGPU [30].

The second problem follows the first and refers to the matter of proper technology choosing for virtual desktop delivery. This task could be mentioned as essential for Virtual Desktop Infrastructure (VDI) organization, because the resulting convenience of use depends exactly upon it. Since we are in need to operate with 3D accelerated applications, the chosen system should provide a sufficient frame rate for real-time interaction. There are, basically, variations of two most common solutions available, based on VNC and NX.

In addition, it is possible to refer to proprietary virtual desktop infrastructure solutions, provided by VMWare and Citrix. The latest versions of toolkit are also supporting Linux environment. However, both of them require license acquisition, therefore, they will not be discussed in depth in the article.

The core part of NX protocol and software was developed and published by NoMachine [21]. This technology is based on the usage of special NX proxy agent for the X server command sequence capturing and transmitting. That means that the X terminal should also run at the client side. Remote desktop application set, based on that solution, have some advantages, such as tolerance to the lack of good and stable bandwidth and secure tunneling. However, the usage of the X terminal on the client combined with the poor connection quality could lead to

the performance drop while running the application showing intensively changing frames, e.g. video playback, or OpenGL applications. Each time the thin client will require the corresponding X hook should be sent, thus the resulting amount of frames per second will depend on quantity of requests processed through the network. In some cases, where interactive demeanor or smooth image is not urgent, that will not be vital, but if there will be a need to capture the window frames, even on the workstation's side, the desirable frame rate will not be achieved.

VNC, which is standing for Virtual Network Computing, is another standard protocol with a huge amount of open-source implementations available [32]. VNC toolset is also consists of client and server parts, however the thin client is only responsible for input capturing and frame displaying. The main difference between VNC and NX system is the fact that all computing operations associated with obtaining a frame are happening at the server side. That is why this solution could be noted as more dependent of network connection quality due to the necessity of constant framebuffer steaming large amounts of image. In other hand VNC could show the true workstations' performance because it is not influenced by third party factors. The connection could be directed through the ssh tunnel, but in most cases it should be done manually.

Thus, it is clear that VNC is a more promising technology, when the remote access for interactive content is needed. In other cases NX could be used as a free and reliable alternative.

As for the particular implementation of VNC protocol, the TurboVNC had been chosen. This program optimizes the process of frame transmission and composition through the use of algorithms for analysis and comparison of the current and previous images, and eventually transmits only those parts of the image that differ [22]. As a result, the network load is reduced. Moreover, for the frame data compression that is going to be transmitted an accelerated a version of the libjpeg library, called jibjpeg-turbo, is used. By the usage of SIMD instruction set for image encoding the same result could be achieved from two to six times faster on assurances of developers [19]. Another nice and useful feature presented is a web-based desktop delivery support using Java applet technology. The only really significant weakness on which attention can be drawn is that TurboVNC can not be integrated with authentication services used in the infrastructure. The only two methods supported right now are password authentication and PAM-based authentication, which is recommended. The first option also has two possible ways of usage: user can set a permanent passphrase or generate a one-time password with vncpasswd utility. The password hash will be stored in .vnc folder at users home location along with the desktop environment initialization script.

It should be also understood that construction of any infrastructure should start from the list of those tasks, which are intended to be solved with the help of resulting system. Therefore, the last question should be addressed to methods of preparation and use of computing resources to tasks that are typical for the area of computer graphics. At least two main categories could be assigned here that

should be treated in a different manner: real-time applications, which require an urgent resource allocation in most cases, and batch tasks, like image or animation rendering, which could be queued and scheduled.

For more efficient use of computing resources, high-performance clusters are exploiting a resource management systems. These allow to handle the massive flow of jobs from different users. A manual task management comes as a problem of uncontrolled distribution and node interaction time, with presence of competition for computing resources showing up a negative impact on the entire system performance, brought up by parallel jobs. As a consequence, modern computing clusters should use a queuing system to operate with machine resources.

SLURM (Simple Linux Utility for Resource Management) is one of the systems featuring such a solution [31]. This software is shipped as free to use open-source project and its mentions could be found in the TOP500-ranked systems, including the most powerful today - Tianhe-2 cluster [16]. SLURM is able to run, manage, monitor and complete tasks on the nodes. It allows you to manage user access to computing power, according to the allocated and joint rights to perform the tasks. Also, it has a number of advantages with respect to the virtual graphics cluster.

In SLURM, as in other similar resource managers, tasks are getting carried out by scripts. The task's description could be divided into main body and so-called prologue and epilogue scripts. Prologue scripts are designed to prepare computing environments for a specific task, and an epilogue script could help to return the environment to the initial state. In a case of rendering tasks, prologue script can start up the X server on each computing node allocated by the user, and the epilogue should be responsible for killing X session on completion of the task.

Queues have a number of options, including the queue priority ordering, which makes sense in case there is a need to give a preference to the real-time rendering applications against batch tasks.

The group of researchers from Computer Engineering Department, Bogazici University have developed a plugin for SLURM profiler called AUCSCHED, designed for heterogeneous CPU-GPU clusters. It allows to reduce the usage of the computing resources up to 25 % compared with a bare SLURM system, because its current profiler's implementation could operate only with a node range, rather than computation device range [33]. AUCSCHED solves this problem, allowing to pack tasks on nodes more tightly and closely to each other, also allowing to change the number of used GPU (or other accelerating devices) at runtime [34]. Whereas the first feature is very useful for allocating a big amount of real-time tasks, both of them suit perfectly for the batch rendering jobs. As a result, AUCSCHED plugin increases efficiency while operating with several graphics cards.

5 Applications and Development Tools

Whereas all the necessary preparatory operations for computing infrastructure have been carried out, the environment evaluation should be performed and the

solution of the raised visualization objectives should be designed. Thus, in this section the most popular scalable applications and development tools will be considered to get an idea on how to start using obtained resources.

The first, and, probably, the most prominent system for distributed rendering that should be noted is the Chromium, which also could be mentioned as Cr [1]. Its basic principle lies in substitution of system's graphics libraries for selected running OpenGL applications. That means, that it is able to run the graphics applications in parallel, even if there is no source code access available. Onward, the Chromium application loader will use fake library to intercept into OpenGL instruction pipeline and redirect selected command streams to computational nodes that will perform the real rendering job. The resulting rendered frame could be presented not only on single monitor, but also on multiple physical display devices combined into grid, and this is shown as one of the Chromium features.

But in spite of all the positive Chromium's aspects, it has one essential flaw, which is revealed rather quickly when the amount of graphics data and OpenGL is getting larger. Since the Chromium is the streaming system and it does not require to distribute the target applications among all active rendering nodes, certain chunks of split graphics data and corresponding OpenGL command sequences should be transferred to the working servers. When communication costs become so large that the network is no longer able to deal with them, the application gets a slowdown instead of expected frame rate or quality improvement at final.

If the development of a new application or a porting of existing one is planned, the look at the Equalizer framework should be taken, which, probably, becomes one of the most powerful development toolset in the area of distributed rendering.

Equalizer is a system for development of distributed parallel OpenGL-based applications, which provides a rich API for management of 3D graphics object allocation, and also flexible graphics pipeline and node configuration [27].

The main feature of Equalizer, which can highlight it amid other similar frameworks and libraries, is the presence of load balancing mechanisms. Also, a well-designed architecture simplifies introduction of new functionality.

Apart of Chromium, the executed application should be available at the node that is going to be used. That solves the problem, which Chromium is affected by, due to the fact that the nodes need to transmit only particular parts of the data that is required to render the area or the object for which the node is responsible. Thus, a client node that is connected to the output device, may receive just a ready frame, or the set of parts thereof that simply need to be composed.

Once written with Equalizer library, an application acquires the ability to be used with various output devices, like display wall, immersive display, virtual reality glasses and a classic single display setup of course. This makes it easy to build systems that expose benefits of augmented reality and of course virtual reality, which broke through the world of computer graphics as a buzzword in the last years. The rendering workflow for the application done in the right way

could be managed just by making changes in the configuration file, that is getting passed to the both, server and client parts. The configuration itself can be made sufficiently flexible due to amount of parameters that could be involved. They include but not limited to algorithms of data sorting and distribution among nodes, data compression algorithms, the quality of resulted frames, displayable volumes of virtual space. The availability of all these things and taking them into mind while developing software itself and parameters needed allows to perform assigned tasks in a most efficient way.

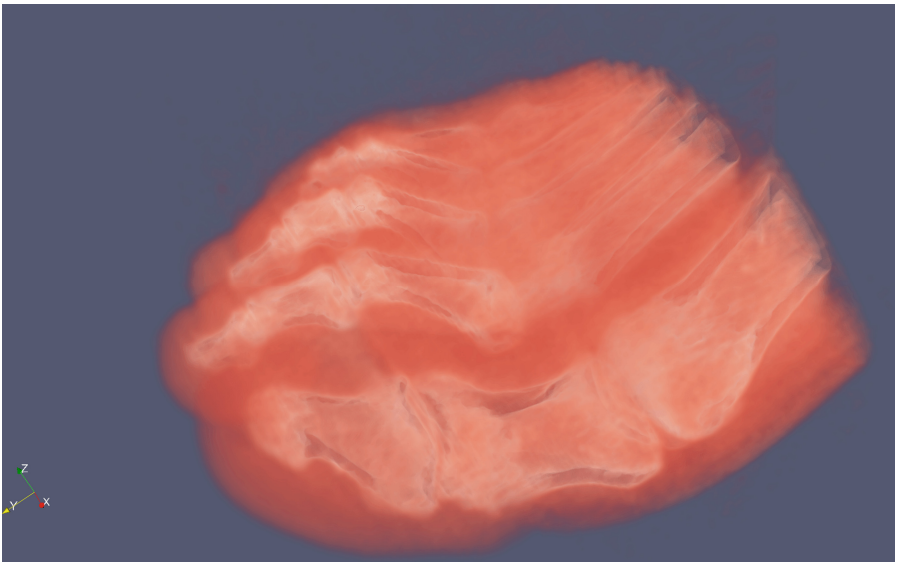


Fig. 3. Volume rendering of tomography results made with ParaView

Using Equalizer and VirtualGL in one sheaf is supported, but requires to make some specific changes to the configuration. First, the frame compression method should be explicitly set for VirtualGL at the application launch. Next, if several graphical units should be used at single node, each application's rendering pipe should be directed to the dedicated GPU manually, except the first one on each node, because it will be managed by VirtualGL itself [26].

In addition, Equalizer developers recommend to use their framework in combination with graphics engines that could provide a scene representation with the usage of scene graph data structure. In this case the whole collection of objects, which represent a virtual world, could be decomposed into a tree structure, thus making it easy to clip undesirable scene parts and distribute the load for remaining evenly. One of such libraries, which is commonly used for development of simulation systems and already has all necessary bindings available, is OpenSceneGraph [11].

Scientific visualization problem should be singled out as a separate topic of conversation, since it has its own peculiar properties. In most cases, the calculation results are represented as a discrete set of values, and the actual meaning is also depending on particular interpretation rules. The biggest difference lies in the way the display of volumetric objects. Here, the body is described not by an external polygonal mesh, but with the points set individually, which are filling the volume, so it comes for volume rendering. Visualization Toolkit (VTK in a short) could be mentioned as one of the most common solutions in this area [9]. The main value of the program is to have a programmable standardization of interfaces and data formats that not only allows, if necessary, easy to repeat current experiment, but also affects the amount of available solutions and extensions. or there could be found a specific exporter, like in case of OpenFOAM application [8].

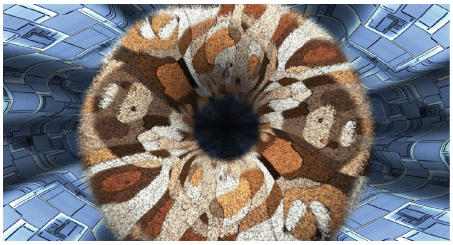
ParaView is standing out among the rest of VTK application, and it should be referred as the main subject for the following discussion [24]. That application allows to process, render and visualize data coming from many popular scientific packages and data sources. User is able to interact with the image in real-time, or render an animation, video and image files with the selected frame set. Initial data could be supplied with additional materials, like textures, polygonal models, shaders, etc., which could be involved into graphics scene compositing. Powerful API combined with Python scripting support allows to automate actions and tap advanced features. The ability to distribute rendering tasks is implemented with the usage of ICE-T library that automatically enables the feature of displaying the image on multiple screens simultaneously [7]. Figure 3 illustrates the ParaView rendering possibilities for volume data set representing a part of the foot [35].

Two more VTK-based visualization systems that could be noted here as the applications for distributed and parallel rendering are VisBox and VisIt. The first one could help to create a virtual or augmented reality showcase, while the second one will also provide an opportunity to process data in parallel.

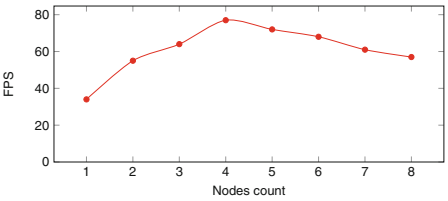
6 Evaluating Results

As mentioned above, the most rapid onset of system usage could be achieved with Chromium, since it provides a possibility to launch an existing software without making any changes on it. For the system performance evaluation we used some tools from GpuTest benchmarking suite that are able to show distributed rendering peculiarity [17]. All tests use the same configuration, which means FullHD resolution for the application running in a fullscreen mode and tight jpeg compression with frame spoiling enabled for the VirtualGL proxy.

The first benchmark case, called FurMark, appears as single object of a torus shape covered by fur, which is produced with a shader program, and filling the most of the display area, as shown in Fig. 4a. The object being always in transition causes fur effect to be recalculated all the time the movement is animated. This makes FurMark a great scenario to determine GPU calculating capabilities and also to scale it in a sort-first manner.



(a) FurMark scene



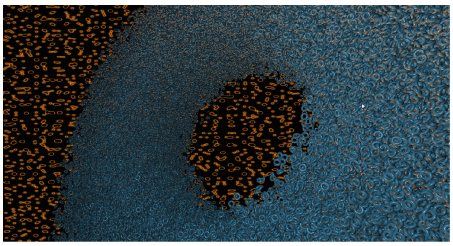
(b) FurMark benchmarking results

Fig. 4. Running FurMark with Chromium in sort-first mode

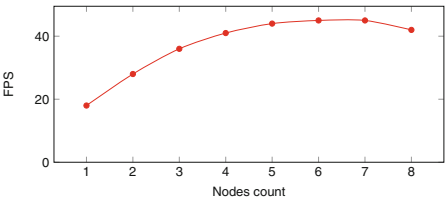
So, for this case the screen is split in a number of tiles equal to the amount of calculating nodes. The priority is given to horizontal division, except cases with an even number of nodes bigger than two: the screen will be also divided vertically here.

Results of evaluation are shown in Fig. 4b. As it was expected from a streaming system, performance was able to grow only until it has met the bandwidth limit, which happened after adding a fifth node. Dips on the odd node counts are indicating unequal distribution of workload. Nevertheless, the peak frame rate was twice as high as the initial one.

The second case uses the same torus figure but in a different manner. It fills exactly the same volume as in the previous test with toruses of a smaller size, as it shown in Fig. 5a. This test covers completely different area, its main aim is to show the transition effectiveness of a multiple objects. As for distributed rendering, it is a great opportunity to show a sort-last compositing technique, based on z-buffer relation. It means that workload will be split not by screen area ownership, but by the groups of dedicated objects. At the final stage the frame will be produced using only those objects that are not overlapped by another ones, however, it leads to some overheads.



(a) GiMark scene



(b) GiMark benchmarking results

Fig. 5. Running GiMark with Chromium in sort-last mode

As it could be seen in Fig. 5b, the workload is distributed much more evenly, considering that each computing system is getting almost the same amount of

data to process. Consequently, attaching additional nodes to the system, including the sixth one, brings positive impact on the result. Apparently, seventh processing unit has no influence at all, and extension up to eight nodes even leads to performance drop, which could be treated as a bandwidth limitation.

7 Conclusion

The technology stack and software environment described above reveal the possibility of Linux operating system usage as a platform for intensive computer graphics applications. The chosen solution allows not only to run applications with 3D acceleration remotely, but also to control a launch process with the usage of queue management system. The number of accelerated gateway nodes and computational nodes could be varied on demand, as well as their performance characteristics. A set of development tools and ready-to-use applications is given, which have the ability to work in parallel rendering mode in the following environment, allowing the most efficient use of available resources.

Further scope of work for virtual infrastructure improvement could be designated with effort of traditional X graphics server replacement to the alternative implementation, like Wayland or Mir, which are getting shipped as default at the latest Fedora and Ubuntu distributions respectively. Noted display management software have some advantages, such as simplified display server protocol and ability of direct framebuffer manipulation, thereby promising some experience and performance improvements [10,20].

Acknowledgements. Research was carried out using computational resources provided by Resource Center "Computer Center of SPbU" (http://cc.spbu.ru/) and supported by grants of Russian Foundation for Basic Research (projects no. 16-07-01111, 16-07-00886, 16-07-01113) and Saint Petersburg State University (project no. 0.37.155.2014).

References

1. Chromium Documentation. http://chromium.sourceforge.net/doc/index.html. Accessed 17 Jan 2016
2. CUDA Toolkit | NVIDIA Developer. https://developer.nvidia.com/cuda-toolkit. Accessed 15 Sept 2015
3. GeForce GTX 680 | Specifications | GeForce. http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-680/specifications. Accessed 03 Oct 2015
4. How to investigate and use Turbo mode, C-States, P-States in XenServer. http://xenserver.org/partners/developing-products-for-xenserver/19-dev-help/138-xs-dev-perf-turbo.html. Accessed 24 Oct 2015
5. Intel Graphics Virtualization Technology (Intel GVT). https://01.org/igvt-g. Accessed 18 Oct 2015
6. Part I. Installation and Configuration Instructions, Chap. 8. Common Problems. ftp://download.nvidia.com/XFree86/Linux-x86/256.44/README/commonproblems.html. Accessed 15 Sept 2015

7. Sandia National Laboratories: IceT. http://icet.sandia.gov/. Accessed 19 Jan 2016
8. The ParaView Post-processor. http://www.openfoam.org/features/paraview.php. Accessed 19 Jan 2016
9. VTK - The Visualization Toolkit. http://www.vtk.org/. Accessed 19 Jan 2016
10. Wayland FAQ. https://wayland.freedesktop.org/faq.html. Accessed 24 Mar 2016
11. Openscenegraph and equalizer. Technical report, Eyescale Software GmbH, Neuchâtel, Switzerland, April 2010
12. Mesa FAQ, 9 October 2012. http://www.mesa3d.org/faq.html. Accessed 21 Nov 2015
13. PowerEdge R720, R720xd Technical Guide, April 2012. http://partnerdirect.dell.com/sites/channel/Documents/PowerEdge-Rack-Server-R720-R720xd-Technical-Guide-April2012.pdf. Accessed 11 Sept 2015
14. Extending slurm with support for gpu ranges. Technical report, VMWare, Palo Alto, CA, United States (2013)
15. NVIDIA GRID K2 Graphics Board: Board Specification, January 2013. http://partnerdirect.dell.com/sites/channel/Documents/PowerEdge-Rack-Server-R720-R720xd-Technical-Guide-April2012.pdf. Accessed 03 Oct 2015
16. Simple Linux Utility for Resource Management, 24 November 2013. http://slurm.schedmd.com/. Accessed 29 Jan 2016
17. GpuTest - Cross-Platform GPU Stress Test, OpenGL Benchmark for Windows, Linux and OS X — Geeks3D.com. 4 March 2014. http://www.geeks3d.com/gputest/. Accessed 16 Nov 2015
18. VMware KB: Poor virtual machine application performance may be caused by processor power management settings. 28 August 2014. https://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1018206. Accessed 24 Sept 2015
19. libjpeg-turbo | About , Performance, 13 August 2015. http://www.libjpeg-turbo.org/About/Performance. Accessed 24 Sept 2015
20. Mir: Welcome to Mir, 24 March 2015. https://unity.ubuntu.com/mir/. Accessed 24 Mar 2016
21. NoMachine - A brief description of the NX protocol in version 4 or later, 21 December 2015. https://www.nomachine.com/AR11K00745. Accessed 24 Sept 2015
22. TurboVNC | About / A Brief Introduction to TurboVNC, 13 August 2015. http://www.turbovnc.org/About/Introduction. Accessed 24 Sept 2015
23. VirtualGL | Main / The VirtualGL Project, 13 August 2015. http://www.virtualgl.org/. Accessed 24 Sept 2015
24. Utkarsh Ayachit. The ParaView Guide: A Parallel Visualization Application. Kitware (2015)
25. Bogdanov, A., Degtyarev, A., Korkhov, V., Gaiduchok, V., Gankevich, I.: Virtual supercomputer as basis of scientific computing. In: Clary, T.S. (ed.) Horizons in Computer Science Research, pp. 159–198. Nova Science Publishers, New York (2015). ISBN: 978-1-63482-499-6
26. Eilemann, S.: VirtualGL Support. 22 December 2011. https://github.com/Eyescale/Equalizer/blob/master/doc/README.VirtualGL. Accessed 16 Nov 2015
27. Eilemann, S.: Equalizer Programming and User Guide. Eyescale Software GmbH, 26 July 2013
28. Gankevich, I., Gaiduchok, V., Gushchanskiy, D., Tipikin, Y., Korkhov, V., Degtyarev, A., Bogdanov, A., Zolotarev, V.: Virtual private supercomputer: Design and evaluation. In: 9th International Conference on Computer Science and Information Technologies, CSIT 2013, Revised Selected Papers, pp. 1–6 (2013)

29. Gankevich, I., Korkhov, V., Balyan, S., Gaiduchok, V., Gushchanskiy, D., Tipikin, Y., Degtyarev, A., Bogdanov, A.: Constructing virtual private supercomputer using virtualization and cloud technologies. In: Murgante, B., Misra, S., Rocha, A.M.A.C., Torre, C., Rocha, J.G., Falcão, M.I., Taniar, D., Apduhan, B.O., Gervasi, O. (eds.) ICCSA 2014, Part VI. LNCS, vol. 8584, pp. 341–354. Springer, Heidelberg (2014)
30. Isoka, D.: Understanding, Evaluating RemoteFX vGPU on Windows Server 2012 R2 — Remote Desktop Services Blog, 6 June 2014. https://blogs.msdn.microsoft.com/rds/2014/06/06/understanding-and-evaluating-remotefx-vgpu-on-windows-server-2012-r2/. Accessed 21 Nov 2015
31. Jette, M., Grondona, M.: Slurm: Simple linux utility for resource management. Technical report UCRL-MA-147996 REV 3, Lawrence Livermore National Laboratory, Livermore, CA, United States (2003)
32. Richardson, T., Stafford-Fraser, Q., Wood, K.R., Hopper, A.: Virtual network computing. IEEE Internet Comput. **2**(1), 33–38 (1998)
33. Soner, S., Özturan, C.: An auction based slurm scheduler for heterogeneous supercomputers and its comparative performance study. Technical report, Computer Engineering Department, Bogazici University, Istanbul, Turkey (2013)
34. Soner, S., Özturan, C., Karac, I.: Extending slurm with support for gpu ranges. Technical report, Computer Engineering Department, Bogazici University, Istanbul, Turkey (2013)
35. Tierny, J.: Visualization Tutorial Exercise - Visualization with ParaView. 26 July 2014. http://www-pequan.lip6.fr/~tierny/visualizationExerciseParaView.html. Accessed 28 Feb 2016
36. Jain, M.: Under the hood of GPU Sharing technologies: vSGA and vGPU (8 January 2014).https://www.citrix.com/blogs/2014/01/08/under-the-hood-of-gpu-sharing-technologies/. Accessed 18 Oct 2015