PROCEEDINGS OF SCIENCE

Supporting grid-enabled GPU workloads using rCUDA and StratusLab

John Walsh*
Trinity College Dublin
E-mail: john.walsh@scss.tcd.ie

Brian Coghlan
Trinity College Dublin
E-mail: coghlan@cs.tcd.ie

David O'Callaghan
Trinity College Dublin
E-mail: david.ocallaghan@cs.tcd.ie

Recent advances in hardware and software virtualisation capabilities have made it possible to customise hardware and software environments for a huge variety of applications. Grid infrastructures have capitalised on many of these advances, for example, through the use of grid-enabled virtual machines which provide well-known and trusted user services and environments. In particular, the StratusLab cloud distribution has greatly facilitated the creation of hybrid cloud/grid infrastructures. At the same time, we have seen a rapid increase in the utilisation of General Purpose Graphical Processing Units (GPGPUs) to handle massively data parallel workloads.

There are significant technical difficulties in integrating GPGPUs as first-class grid-resources. Furthermore, the use of full GPGPU hardware pass-through to virtual machines, which could be used to overcome some of these challenges, has only had limited success. An alternative network-based GPGPU virtualisation method has been shown to be more successful.

We review these difficulties, and propose how both StratusLab and network-based GPGPU virtualisation, such as rCUDA and Mosix VCL, may be used to ameliorate some of these issues.

EGI Community Forum 2012 / EMI Second Technical Conference
26-30 March, 2012
Munich, Germany

*Speaker.

PoS(EGICF12-EMITC2)010

1. Introduction

The increasing capabilities of general purpose graphics processing units (GPGPUs) over the past few years has resulted in a huge increase in their exploitation by all the major scientific disciplines. With three of the top five clusters in the current November 2011 Top500[1] supercomputers list using NVIDIA GPGPUs, we would expect the number of GPGPU deployments at grid resource centres to grow significantly over the next few years. However, there are two major problems in supporting grid access to such resources. Firstly, there is currently no standardised way for resource centres to advertise/publish availability of these resources. Secondly, there are deficiencies in current batch scheduling systems which make it difficult to guarantee exclusive access to those resources.

GPGPU application development is dominated by two API frameworks: OpenCL[2] and CUDA[3]. OpenCL supports heterogeneous compute environments such as AMD and Intel CPUs, AMD, Intel and NVIDIA GPGPUs, and Cell Broadband Engine co-processors. CUDA applications currently run on NVIDIA GPGPU devices only. However, the GNU Ocelot project[4] is attempting to enable CUDA application development for non-NVIDIA processors. CUDA applications offer a performance advantage over OpenCL[5].

We present the results of investigations into access to many GPGPUs distributed over a grid-enabled cluster. We propose a possible virtual machine(VM) architecture that employs multiple distinct virtualisation technologies: GPGPU virtualisation using Mosix VCL[6] or rCUDA[7], and machine virtualisation using the StratusLab[8] "Infrastructure as a Service" (IaaS) distribution. The proposed solution is, however, intended to be vendor-neutral, and indeed initial experiments with our GPGPU VM were carried out using the Xen Cloud Platform[9]. In addition, we consider that in so far as the current hardware technology allows, the architecture can be further extended to achieve greater levels of user configuration, control and security. A potential benefit in using this model is to serve as an alternative method to easily exploit many locally distributed GPGPU resources without the need for additional application frameworks such as MPI[10].

Using a cloud approach for provisioning GPGPU worker nodes on the European Grid Infrastructure (EGI)[11] has already been investigated[12]. Our investigation, however, looks at some of the challenges that we have encountered with both the currently available hardware and software solutions. It should therefore also serve as a partial review of the current state of the art. This is a "work in progress".

2. GPGPU Grid Integration Challenges

Although it is fairly straightforward to assemble grid-enabled worker nodes with GPGPUs, integrating them as first-class resources, such as CPU and storage resources, into the grid infrastructure is a much more challenging problem. The fundamental issues are discussed in greater detail in the following sub-sections.

2.1 Batch System Integration

Some batch system schedulers, such as SLURM[16], (SUN) Grid Engine[17] and LSF[18], now support "Generic" or "GPGPU Resources". Maui[19] - the prominent job scheduler used by

the many of EGI resource centres - does not have official support for GPGPU resources. There is an experimental patch[20] that enables this feature, however we have not been able to confirm its robustness and functionality. MOAB[21], the advanced commercial offering of Maui, also support GPGPU resource scheduling. MOAB requires Torque versions 2.5.6 or higher, which supports a "gpu" attribute.

2.2 Hardware device level access security

In order to allow batch users to access the GPGPU device, the operating system access control is generally permissive and allows all users to read from/write to the device. In the case of typical grid compute resource centres with multi-core worker-nodes, each worker-node tends to expose as many job-slots to the batch system as there are CPU cores. A side-effect of this is that simultaneous user jobs may interfere with each other if they attempt to access the same GPGPU device concurrently. NVIDIA GPGPUs can offer some level of protection through the *EXCLUSIVE_THREAD* or *EXCLUSIVE_PROCESS* **Compute Mode** facility; when using this, one of the competing jobs will fail when trying to access the GPGPU device.

NVIDIA provides a second method to expose only a specified set of devices to a process by setting the *CUDA_VISIBLE_DEVICES* environment variable. Setting *CUDA_VISIBLE_DEVICES=""* in a process has the effect of hiding all NVIDIA GPGPU devices when accessed through the CUDA API. Indeed, SLURM uses this variable for both GPGPU allocation and to provide process level protection between multiple devices on the same host[22]. Note, however, that a user's job could unset this environment variable and therefore bypass any such device visibility restrictions imposed by *CUDA_VISIBLE_DEVICES*. We are not aware if there are similar mechanisms available for AMD or Intel GPGPUs.

The authors note that there may be some scope to exploit batch system prologue and epilog facilities to change device ownerships, but we have not investigated this possibility.

2.3 GPGPU Glue-schema support

The Grid Information System plays a pivotal role in collecting and publishing quasi-realtime information about the accumulated set of grid resources. The information is processed and republished according to a well defined standardised format known as the *Glue Schema*[23]. The Glue Schema defines a set of attributes and their semantics. Its intended use is to facilitate resource discovery, selection and monitoring. The current standard is version 2.0, however, version 1.3 is the predominant standard used by most production resource centres.

The most common usage of the Glue Schema relates to standard CPU and storage capabilities. The Glue Schema 2.0 standard defines an ExecutionEnvironment [24] attribute that could be used to publish GPGPU resources, but the authors have not yet investigated as to whether this is fully suitable for GPGPU "single instruction multiple data" (SIMD) models of execution.

It should be noted that the EUmedGrid community use the gLite *CERequirements* JDL expression and a CREAM CE BLAH customisation hook to explicitly assign grid GPGPU jobs to the appropriate GPGPU resources[25][12]. This solution does not address resource discovery and usage, and it depends on an a-priori knowledge of how the software environment tag is used.

2.4 Information System Providers

The grid information system is populated and updated regularly with data obtained through the use of information system provider "Plugins". Notwithstanding the lack of an appropriate GPGPU Glue Schema resource definition, the additional batch system support limitations covered in section 2.1 preclude the authors at present from developing appropriate GPGPU information system plugins.

3. GPGPU Virtualisation

We considered two complementary approaches to GPGPU virtualisation. The first uses GPGPU PCI pass-through to a VM. This aims at ensuring that the VM has complete control over the GPGPU, and therefore retains excellent and undiminished flops/second calculation rates. The second approach, networked GPGPU virtualisation, uses socket based communication. Here the VM accesses a non-local GPGPU through a network connection.

3.1 GPGPU Hardware Virtualisation

Full GPGPU hardware virtualisation is, at the time of writing, a highly specialised area and quite far from being a turnkey capability[13]. The current limiting factors include: 1) CPU support for direct I/O to PCI devices; 2) Motherboard support for direct I/O to PCI devices; 3) Kernel and hypervisor support for PCI pass-through; and, 4) GPGPU support for PCI pass-through.

Both items 1 and 2 are closely related, but can be solved with appropriate hardware supporting Intel VT-D or AMD IOMMU CPU extensions. Recent updates to the Linux kernel address PCI pass-through and direct I/O to any PCI devices passed into the control of its virtual machines. However, unlike most other PCI devices, there are still significant challenges in full GPGPU PCI-pass-through and very few GPGPU devices support this. To date we have not yet succeeded in creating a guest VM on our experimental test-bed using our existing GPGPU hardware.

3.2 Networked GPGPU Virtualisation

A more successful approach to the GPGPU virtualisation problem uses a client/server model with the GPGPU exposed through a network layer. This has been used by ShadowFax[14], rCUDA, gVirtuS[15], and Mosix VCL. The server nodes, i.e. those nodes with control over the physical GPGPU, can be relatively lightweight: They only need OS support to access the GPGPU and they do not need any additional software such as a grid middleware stack. The clients themselves do not require any physical GPGPU, but have access to collections of remote GPGPUs. Our experimental test-bed was used to investigate two solutions: rCUDA and Mosix VCL. The choices were partially driven by the need to accommodate both CUDA and OpenCL application use cases.

3.2.1 rCUDA

The rCUDA model allows the GPGPU client to transparently access "listed" GPGPU server nodes. Client application code must be recompiled and linked against the replacement rCUDA libraries. These libraries implement most of the CUDA 4.0 runtime API and handle the network communication with the rCUDA servers. Each server node must host one or more GPGPUs and

must launch a server daemon **rCUDAd**. This daemon uses a customisable TCP port number (with default). The client program will interrogate the environment variable *RCUDA* to determine which server host/port pairs to communicate with.

3.2.2 Mosix VCL

The Mosix VCL model takes a different approach to rCUDA, and works with the OpenCL 1.1 framework only. The "back-end" nodes host one or more OpenCL-enabled CPUs or GPGPUs. The "Hosting Node" front-end communicates with the back-end nodes using a "broker" daemon. The OpenCL applications are launched using the "vlcrun" script. This script accepts an optional user-defined command-line argument parameter that affects how the broker allocates remote CPUs/GPGPUs to the client application. The default policy is to allocate all resources. Unlike rCUDA, the OpenCL application does not require any re-compilation on the client.

3.2.3 Performance Considerations

The introduction of a network-layer will have a significant impact on the performance of GPGPU code. Using high-bandwidth, low-latency interconnects such as Infiniband will reduce some of the impact. Best perfomance by both rCUDA[26] and Mosix VCL[27] is achieved with long kernel operations coupled with infrequent buffer I/O. Mosix VCL supports an additional API, SuperCL, that improves the performance of the network-layer by encapsulating a programmable sequence of kernel or memory operations into a few asynchronous network packets.

4. GPGPU Provisioning with StratusLab

StratusLab is an open-source cloud distribution based on OpenNebula that allows both grid and non-grid resource centres to offer and to exploit IaaS clouds. It provides the client tools to fully manage the life-cycle of a virtual machine, and also provides a "market place" of pre-built VM images. Moreover, StratusLab facilitates **contextualisation** - the process of enabling customisations to the end-user's deployed virtual appliances. StratusLab lends itself to be a complete tool-chain to deploy the rCUDA/VCL front-end nodes.

4.1 Contextualisation Issues

As StratusLab only permits contextualisation of the VM guest and not the hypervisor host, we were unable to use it to manage the GPGPU server back-end. Furthermore, the front-end contextualisation is "static" as we do not currently have a mechanism for choosing GPGPUs dynamically from the pool of distributed resources. To simplify integration into our existing fabric, we provide a gLite worker-node front-end VM image.

4.2 Batch System Configuration

We have no mechanism to dynamically add the front-end node into the pool of batch system worker-nodes. We therefore use static worker-node entries. The batch system is also configured so that the front-end presents a single job slot. As only one job can run on the worker-node, this ensures exclusive access to the allocated GPGPUs.

5. A Fully Virtualised Architecture

We have considered some of the difficulties in hosting virtual machines with full GPGPU pass-through. However, assuming that such hurdles can be overcome with appropriate hardware, OS and hypervisor support, we propose that if such full virtualisation along the lines of that illustrated in Figure 1 were available, then we could greatly enhance the available compute infrastructure. Firstly, access to the GPGPU resources would be protected from competing concurrent processes. Secondly, a fully virtualised architecture would allow us to use the cloud platform exclusively to contextualise and deploy both front-end and back-end rCUDA/Mosix VCL nodes.

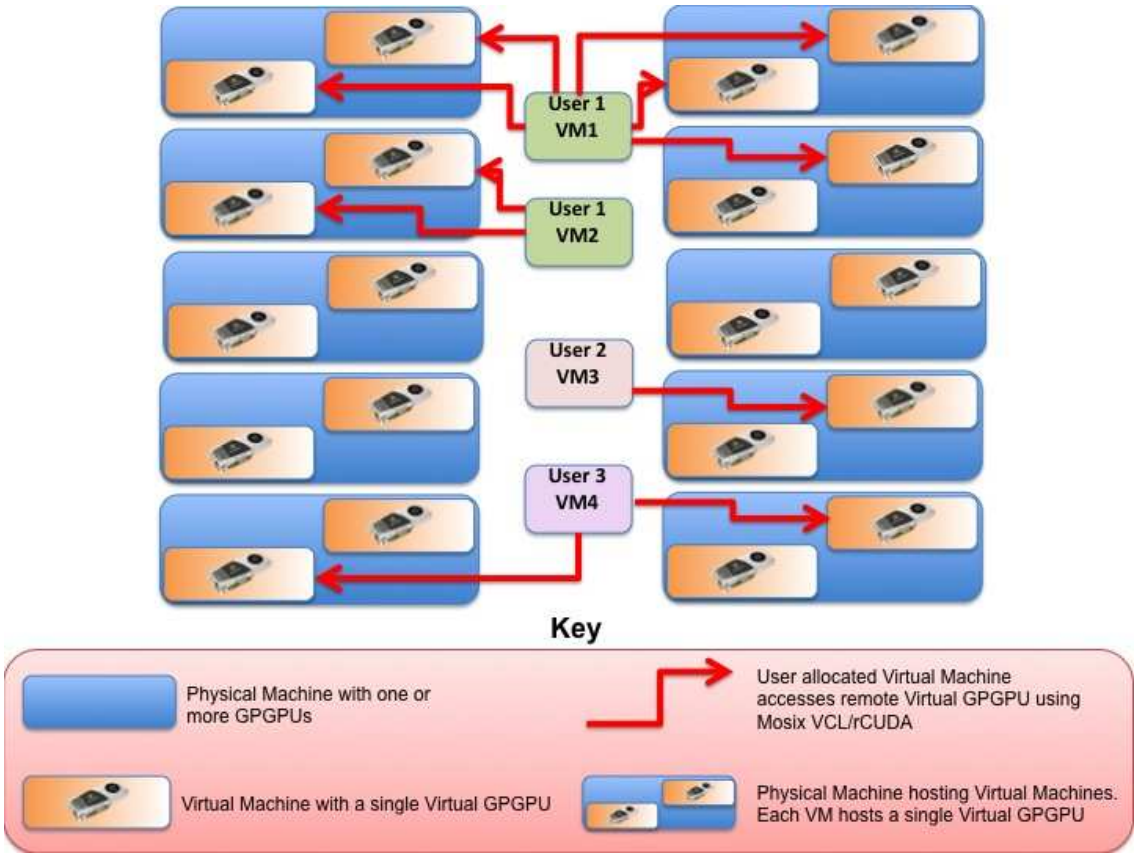

Figure 1: A fully virtualised architecture. Each user-allocated virtual machine has multiple unique virtual GPGPUs. These are accessed transparently through Mosix VCL or rCUDA.

6. Conclusions and Further Work

We have identified issues that hinder us from integrating GPGPUs as first-class grid resources. Some progress can be made by ensuring that the community agree on an appropriate GPGPU resource Glue Schema definition. Furthermore, we have identified batch system scheduler deficiencies and how the configuration of multi-core worker nodes in the batch system may cause GPGPU job integrity issues.

We have proposed how a multi-layered virtualisation may provide some job level protection, and how this may be used to provide an alternative execution environment to MPI when accessing many distributed GPGPUs.

We have discussed weaknesses in our cloud provisioning system, how at this stage our VM GPGPU contextualisation is static, and how we do not currently have a mechanism in which the GPGPU resources can be requested. Future work may include an extension of the OpenNebula platform to accommodate detection of GPGPU resources on the hypervisor nodes, and then to facilitate either PCI pass-through or network based GPGPU allocation methods.

References

[1] *Top 500 Supercomputers List, November 2011* http://www.top500.org/lists/2011/11

[2] M. Scarpino, *OpenCL in Action: How to accelerate graphics and computation* Manning Publications Co., ISBN 9781617290176, 2011

[3] D. Kirk, Wen-mei Hwu *Programming Massively Parallel Processors: A Hands-on Approach* Morgan Kaufmann Publishers, ISBN 9780123814722, 2010

[4] *GNU Ocelot Homepage* http://code.google.com/p/gpuocelot/

[5] Jianbin Fang and Ana Lucia Varbanescu and Henk Sips, A Comprehensive Performance Comparison of CUDA and OpenCL, The 40-th International Conference on Parallel Processing (ICPP'11), Taipei, Taiwan.

[6] *VCL Homepage* http://www.mosix.org/txt_vcl.html

[7] *rCUDA Homepage* http://www.rcuda.net/

[8] *StratusLab Project Homepage*, http://www.StratusLab.eu/index.php

[9] *Xen Cloud Platform Homepage* http://www.xen.org/products/cloudxen.html

[10] M. Snir, S. W. Otto, ; Huss-Lederman, S.; Walker, D. W., Dongarra, J. (1996), MPI: The complete reference , MIT Press , Cambridge, MA .

[11] *European Grid Infrastructure Homepage*, http://www.egi.eu

[12] Vella, F., Cefala, R.M., Costantini, A., Gervasi, O. and Tanci, C, GPU Computing in EGI Environment Using a Cloud Approach, International Conference on Computational Science and Its Applications (ICCSA) 2011, pp. 150-155

[13] Teo en Ming, *Xen VGA Passthrough to Windows 8 with Xen 4.2-unstable*, http://wiki.xen.org/wiki/Xen_VGA_Passthrough_to_Windows_8_with_Xen_4.2-unstable

[14] Merritt, Alexander M. and Gupta, Vishakha and Verma, Abhishek and Gavrilovska, Ada and Schwan, Karsten, Shadowfax: scaling in heterogeneous cluster systems via GPGPU assemblies, Proceedings of the 5th international workshop on Virtualization technologies in distributed computing (VTDC'11), pp. 3-10

[15] Giulio Giunta, Raffaele Montella, Giuseppe Agrillo and Giuseppe Coviello, A GPGPU Transparent Virtualization Component for High Performance Computing Clouds, Euro-Par 2010 - Parallel Processing, Lecture Notes in Computer Science, 2010, Volume 6271/2010, pp. 379-391

[16] *SLURM Batch Scheduler Documentation*, http://www.schedmd.com/slurmdocs/

[17] *Univa Grid Engine Version 8*,
http://www.univa.com/products/grid-engine/whats-new

[18] *IBM Platform Computing LSF Webpage*, http://www-03.ibm.com/systems/technicalcomputing/platformcomputing/products/lsf/index.html

[19] *Maui Cluster Scheduler Webpage*,
http://www.adaptivecomputing.com/resources/docs/maui/index.php

[20] *Maui Batch Scheduler System GPGPU code support* http://www.supercluster.org/pipermail/torqueusers/2012-February/014041.html

[21] *Moab Documentation Website*,
http://www.adaptivecomputing.com/resources/docs/mwm/7-0/Content/topics/nodeAdministration/generalnodeadmin.html

[22] *SLURM GPU support code* http://github.com/SchedMD/slurm/blob/master/src/plugins/gres/gpu/gres_gpu.c

[23] *OGF Glue Schema version 1.3*, http://forge.gridforum.org/sf/go/doc14185

[24] *OGF GLUE 2.0 ExecutionEnvironment Definition*,
http://glue20.web.cern.ch/glue20/#tableExecutionEnvironment

[25] *EUMed GPU Integration webpage*: http://wiki.eumedgrid.eu/twiki/bin/view/InfrastructureStatus/EumedSiteInstallationGPU

[26] J. Duato et al. *An efficient implementation of GPU virtualization in high performance clusters*, EURO-PAR 2009 WORKSHOPS, LNCS, vol. 6043. Springer-Verlag, 2010, pp. 385-394

[27] A. Barak and A. Shiloh, *The Virtual OpenCL (VCL) Cluster Platform*, Proc. Intel European Research & Innovation Conf., pp. 196, Leixlip, Oct. 2011.