Opportunities for container environments on Cray XC30 with GPU devices

Lucas Benedicic, Miguel Gila, Sadaf Alam, Thomas C. Schulthess

April 8, 2016

Abstract

Thanks to the significant popularity gained lately by Docker, the HPC community has recently started exploring container technology and potential benefits its use would bring to the users of supercomputing systems like the Cray XC series. In this paper, we explore feasibility of diverse, nontraditional data and computing oriented use cases with practically no overhead thus achieving native execution performance. Working in close collaboration with NERSC and an engineering team at Nvidia, CSCS is working on extending the Shifter framework in order to enable GPU access to containers at scale. We also briefly discuss the implications of using containers within a shared HPC system from the security point of view to provide service that does not compromise the stability of the system or the privacy of the use. Furthermore, we describe several valuable lessons learned through our analysis and share the challenges we encountered.

Keywords **Linux containers; Docker; GPU; GPGPU; HPC systems**

1 Introduction

In contrast with the now long known hypervisor-based virtualization technologies, containers provide a level of virtualization which allows running multiple isolated user-space instances on top of a common host kernel. Since a hypervisor emulates the hardware, both the "guest" operating system and the "host" operating system run different kernels. The communication of the "guest" system with the actual hardware is implemented through an abstraction layer provided by the hypervisor. Clearly, this software layer creates a performance overhead due to the mapping between the emulated and bare-metal hardwares. Containers, on the other hand, are light, flexible and easy to deploy. Their size is measured in megabytes, which is much less than hypervisors that require a much larger software stack and gigabytes of memory. This characteristic makes containers easily transferable across nodes within an HPC system (horizontal scaling), and deployable within one compute node and thereby increasing its density (vertical scaling).

As the role of graphics processing units (GPUs) is becoming increasingly important in providing power-efficient and massively-parallel computational power to the scientific community in general and HPC in particular. It is well known that even a single GPU-CPU framework provides advantages that multiple CPUs on their own do not offer due to the distinguished design of discrete GPUs.

Despite previous studies on GPU virtualization, the possibilities provided by different virtualization approaches in a strict HPC context still remain unclear. The lack of standardized designs and tools that would enable container access to GPU devices means this is still an active area of research. For this reason, it is important to understand the tradeoffs and the technical requirements that container-based technology imposes on GPU devices when deployed in a hybrid supercomputing system. One example of such a system is the Cray XC30 called Piz Daint, which is in production at the Swiss National Supercomputing Center (CSCS) in Lugano, Switzerland. The system features 28 cabinets with a total of 5,272 compute nodes, each of which is equipped with an 8-core 64-bit Intel SandyBridge CPU (Intel® Xeon® E5-2670), an Nvidia® Tesla® K20X with 6 gigabytes of GDDR5 memory, and 32 gigabytes of host memory.

Working in close collaboration with the National Energy Research Scientific Computing Center (NERSC) and an engineering team of the Nvidia CUDA division, CSCS is working on extending the Shifter framework [2] in order to enable scalable GPU access from containers. Container environment opened up opportunities to enable workload that were typically constrained by a specialized light weight operating system. It allows CSCS to consolidate workloads and workflows that currently require dedicated clusters and specialized systems. As an example, by closely collaborating with the Large Hadron Collider (LHC) experiments ATLAS, CMS and LHCb and their Swiss representatives in the Worldwide LHC Computing Grid (WLCG), CSCS is able to utilize the Shifter framework to enable complex, specific High Energy Physics (HEP) workflows on our Cray supercomputers.

The preliminary results of this work show an improvement in vertical scaling of the system and consolidation of complex workflows with minimal impact to users and performance. This is possible thanks to the deployment of multiple independent containers (processes) sharing the same GPU device. The increased density can significantly improve the overall performance of distributed, GPU-enabled applications by increasing GPU utilization and, at the same time, reducing their communication footprint. Additionally, it is also possible to tailor specific versions of the CUDA toolkit and scientific libraries to different applications without having to perform a complex configuration at the system level. This use case is even feasible for different applications sharing the same compute node. Using examples and results of a subset of LHC experiments workflows, we demonstrate that there is a minimal impact to user interface (job submission script) and utilization of resources as compared to a dedicated environment.

The layout of the paper is as follows: we begin with the motivation for this work, specifically extension of containers to include GPU resources and design challenges that are associated with incorporating one and more GPU devices.

This will be followed by implementation details for GPU and LHC workflows in the Cray environment. In section 4, we describe vertical scaling of the solution to accommodate GPU and node sharing for multiple containers. We conclude with future plans and opportunities to build on our efforts.

2 Motivation

The goal is to provide container's users the ability to access the compute resources of the GPUs available in the Piz Daint hybrid system. In particular, accessing the compute resources of Nvidia's GPU like the Tesla K20x installed on each of the compute nodes of Piz Daint is done through CUDA [1].

2.1 CUDA

CUDA is Nvidia's GPGPU solution that provides access to the GPU hardware through a C-like language (CUDA C), rather than the traditional approach of relying on the graphics programming interface. CUDA extends the C language by allowing the programmer to define CUDA kernels, i.e., special C functions, that are executed in parallel by several concurrent CUDA threads on the GPU CUDA exposes two programming interfaces: the driver API and the runtime API, this last one being built on top of the CUDA driver API. Providing access to the CUDA runtime API to container's users by extending Shifter's functionality is the main focus of this paper. Since CUDA is not a fully open standard, some internal details have not been officially documented. For this reason, an engineering team at Nvidia has been engaged to understand the details about the underlying hardware driver and its runtime libraries.

2.2 Complex workflows on Cray

Certain scientific workflows do not adhere to common HPC practices, consider for example the case of the WLCG, where the software is pre-built and pre-validated by each of the LHC experiments and centrally exposed to all computing facilities using a http-based, read-only filesystem. In this specific context, the software is pre-packaged for RHEL-compatible operating systems and running it on Cray Linux Environment is not immediate, as re-building all the software, taking into account the interdependencies of the various applications is a very complex task on its own and can, potentially, imply application re-validation by the experiments.

Shifter enabled our Cray XC supercomputers to overcome this limitation by being able to run unmodified ATLAS, CMS and LHCb production jobs.

2.3 Security aspects

From a security standpoint, Shifter suits much better than Docker the HPC environment. This is because, by default, Docker allows any user to become

root within the context of the container, and this may have implications if the container has access to shared filesystems. Consider the case in which an container image has malicious code embedded in it, since no validation and security checks are usually performed to the images prior to executing applications, this malicious code could have root access to any filesystem mounted on the compute node. On the other hand, Shifter runs applications in userland, with a very small part of it requiring root privileges (SUID) to create and destroy loop devices and mountpoints. Effectively, this limits the effects of malicious code to what the user could do outside the container.

3 Implementation

A number of relatively small contributions have been done to the Shifter codebase, having done most of the customizations to support these use cases at the configuration entry points provided by Shifter, to the images utilized, or to external points that any user can modify.

3.1 Design considerations

In formulating our design, we had several goals in mind:

- **Functionality** Whenever possible, Shifter's GPU support should be compatible with different programming models, e.g., CUDA or OpenCL, and should support the same level of compatibility as the kernel driver regarding the runtime versions, e.g., CUDA 6.5 or 7.0, and OpenCL 1.0 or 1.1. In regards of complex scientific workflows, there was a clear need for seamlessly supporting unmodified applications, regardless of the operating system or kernel version on the compute node.
- **Performance** Accessing the resources of the compute nodes from containers should not introduce any performance penalty, and should allow GPU-enabled applications to run as fast as the native case.
- **Transparency** Shifter's GPU support should be intuitive and useful in a variety of use cases. Additionally, it should be compatible with applications originally designed to run natively on the system.

3.2 CUDA

Clearly, the motivation comes from extending the benefits of containers to GPU devices and to seamlessly deploy GPU-based applications on multiple machines. But when accessing GPUs, the containers are no longer hardware nor platform agnostic, since they are using specialized hardware and their deployment requires, at least in the Cray XC case, the installation of the Nvidia driver in the host kernel.

Because of the way Nvidia's GPGPU solution for Linux is engineered, the container needs to have access to both of its main components: the kernel

driver and the CUDA runtime libraries. Since containers share the kernel and its drivers with the host system, only the CUDA user space libraries are left to be handled separately in the container. However, it is important that the version of these user space libraries matches the version of the Nvidia driver currently loaded into the host kernel.

One of the first working prototypes featured a complete installation of both the Nvidia driver and a supported CUDA toolkit inside a container. The prototype would then mount the character devices corresponding to the Nvidia GPUs (e.g., `/dev/nvidia0`) during startup of the container. However, this solution had a significant disadvantage: the version of the host driver had to exactly match the driver version installed in the container. Consequently, such container images could not be shared and had to be built locally on each host system.

The latest working solution for Shifter makes container images portable while still leveraging GPUs, since they are Nvidia-driver agnostic. The required character devices and driver files are mounted at deployment time, when starting the container on the target machine.

Before mounting the driver files, the prototype scans the target system looking for the driver libraries that match the currently loaded kernel driver. Two methods can be used to make these libraries available inside the container. The first one involves the use of environment variables `LD_LIBRARY_PATH`, `CUDA_ROOT`. The second one modifies the configuration of the `ldconfig` cache in order for the dynamic linking engine to find the libraries. This set of user-space libraries works as an interface, giving the CUDA applications access to the GPU device via the kernel driver itself. Since Shifter does not currently support mounting individual files, the "discovered" driver libraries are placed in a common directory before launching the container, this directory is then mounted inside the container, and its dynamic-linking configuration is altered in order for the runtime to access the "discovered" libraries.

By using this procedure, it is possible to successfully execute different CUDA and OpenCL applications achieving native performance.

3.3 Complex workflows on Cray

Given the complexity already inherent to the High Energy Physics workloads of WLCG, it was established that porting them to a Cray systems would require some sort of abstraction solution to let these applications run without modifications. Different approaches were evaluated and tested, KVM and Docker, but quickly showed that that Shifter was a better choice because of its relative simplicity in getting it to work, while providing minimal operational overhead.

For this particular case, a Docker image with all the required packages and tools needed for the HEP software to run is used, and deployed on our Cray systems without further modifications. The image itself is based on the official CentOS 6.7 image available on DockerHub with the basic RPMs that every site in WLCG needs to deploy. Shifter is built on a common filesystem that all

Docker	Shifter
Requires a daemon running on the compute nodes	Does not require any daemon on the compute nodes
Allows user to be root within their container and on shared filesystems	User cannot run anything as root
Is not Slurm-friendly	Well integrated with Slurm and other WLMs
Can isolate network by creating NAT or Bridge devices	Shows /dev, /sys and /proc to the container environment
Can run on multiple nodes with own tool (Swarm)	Can run on multiple nodes using WLM integration
Complex to build from scratch	Easy to build from scratch

Table 1: A quick comparison between Docker and Shifter features on an HPC environment

compute nodes mount and thanks to the SPANK Plugin provided, operating this environment is extremely straightforward.

The only modification required on the compute nodes, other than installing Shifter, is supporting CernVM File System (CVMFS), the read-only http-based filesystem containing the applications. Originally CVMFS was exposed to the compute nodes via DVS, but due to load issues, CVMFS had to be mounted natively using a preloaded cache on all compute nodes. Note that this is for performance issues, from a purely functional perspective, no other change is needed on the compute nodes to run this type of workload.

Then, to expose the compute resources to the WLCG community, a piece of software called Advanced Resource Connector Computing Element (ARC-CE, or simply ARC) is installed on an external node to Cray, connected to Slurm in a way that it is able to submit jobs. It has been slightly modified to make it generate a Slurm job submission file with the corresponding Shifter extension `#SBATCH --image` and to prepend `shifter` to the original binary executing the job contents. ARC-CE has its own information system and is responsible for exposing the resource to the WLCG community. This enables submission from the grid as well as a common interface for any WLCG user with our Cray system.

The generated Slurm job file by ARC is standard for any job generated, except for the Shifter extensions `#SBATCH --image`,`#SBATCH --imagevolume` and the `shifter` binary before to the original job execution command:

4 Vertical scaling

It was necessary to understand the requirements and advantages of deploying multiple CUDA processes over a single GPU device. The selected path for this use case is to use MPS [3], Native Slurm and ultimately Shifter.

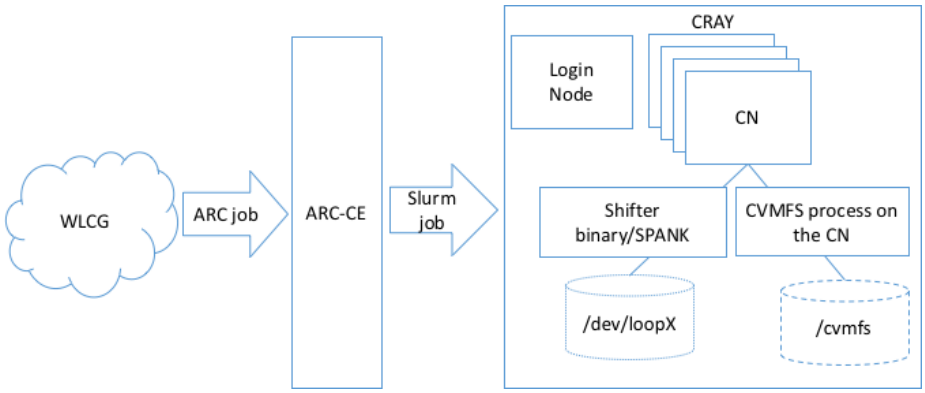

Figure 1: Enabling WLCG jobs on a Cray with Shifter

```
#!/bin/bash -l
# SLURM batch job script built by grid-manager
#SBATCH --no-requeue
#SBATCH -e /scratch/santis/wlcg/ARC1/session_dir/E34LDmigJ8nnHy6IepqQ1NrpABFKDmABFKDmS5GKDmABFKDmNvGe4m.comment
#SBATCH -o /scratch/santis/wlcg/ARC1/session_dir/E34LDmigJ8nnHy6IepqQ1NrpABFKDmABFKDmS5GKDmABFKDmNvGe4m.comment
#SBATCH -p wlcg
#SBATCH --nice=7
#SBATCH -J 'N053b7b4b_ba38_'
#SBATCH --image=docker:miguelgila/wlcg_wn:20160314
#SBATCH --imagevolume=/apps:/apps
#SBATCH --imagevolume=/scratch/santis:/scratch/santis
#SBATCH --imagevolume=/cvmfs/atlas.cern.ch:/cvmfs/atlas.cern.ch
#SBATCH --imagevolume=/cvmfs/grid.cern.ch:/cvmfs/grid.cern.ch
#SBATCH --imagevolume=/cvmfs/atlas-nightlies.cern.ch:/cvmfs/atlas-nightlies.cern.ch
#SBATCH --imagevolume=/cvmfs/atlas-condb.cern.ch:/cvmfs/atlas-condb.cern.ch
#SBATCH --imagevolume=/cvmfs/cms.cern.ch:/cvmfs/cms.cern.ch
#SBATCH --imagevolume=/cvmfs/lhcb.cern.ch:/cvmfs/lhcb.cern.ch
#SBATCH --imagevolume=/cvmfs/sft.cern.ch:/cvmfs/sft.cern.ch
#SBATCH --get-user-env=10L
#SBATCH -n 1
#SBATCH -t 2880:0
#SBATCH -t 2880:0
#SBATCH --mem-per-cpu=2000
[...]
shifter "./ARCpilot" "17.2.7-X86_64-SLC5-GCC43-OPT" [...]
```

Figure 2: Stripped down code of a generated Slurm job with Shifter enhancements

The performance plots provided towards the end of this section give a clear idea of the measured overhead and gains of this type of setup.

4.1 Sharing a GPU across Multiple Processes using MPS

The Nvidia Multi-Process Service (MPS) is a client-server based implementation of the CUDA API that enables multiple processes to utilize shared GPU devices in a concurrent manner using Hyper-Q [3]. Usually, a process creates

its own CUDA context to interact with the GPU. As different CUDA contexts are scheduled to the GPU in a time-sliced manner, work arriving from different processes cannot overlap generally resulting in the GPU being underutilized. The approach used by the MPS is to hold and to manage a single CUDA context through the MPS server. When multiple processes are started and connect to the server, i.e., the processes become the MPS clients, they interact with the GPU only via the MPS server. The MPS server takes care of funneling the work from all processes through the server's CUDA context, allowing workloads from different processes to overlap during their execution.

A diagram of multiple processes accessing a single GPU device directly, i.e., without an MPS instance, is shown in Figure 3. Figure 4 depicts the MPS server managing one CUDA context across several processes.

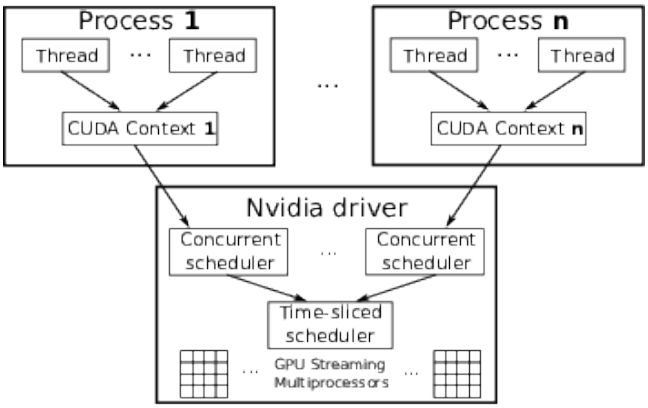

Figure 3: Workloads from different processes attached to their own CUDA contexts are scheduled to a single GPU device in a time-sliced manner.

4.2 Sharing nodes with different workflows

With native Slurm it is possible to share nodes with different workloads. In the context of HEP applications, ATLAS, CMS and LHCb jobs that require a reduced number of cores can be executed on the same node. This is common practice in the WLCG community, where cluster nodes are shared among the different experiments supported by each site, but relatively uncommon on Cray systems. The factor limiting the number of jobs that can be run in parallel on a node, other than memory and CPU, is Slurm's Generic Resource (GRES) Scheduling `craynetwork` parameter, which by default is set to 4. This is overridden and raised it up to 32, to better accommodate the size of the jobs arriving at the site (either 1-core jobs, or 8-core jobs). Then, Shifter requires a loop device available on the compute node for each job and so, the number of available loop devices is incremented to match the value of `craynetwork`.

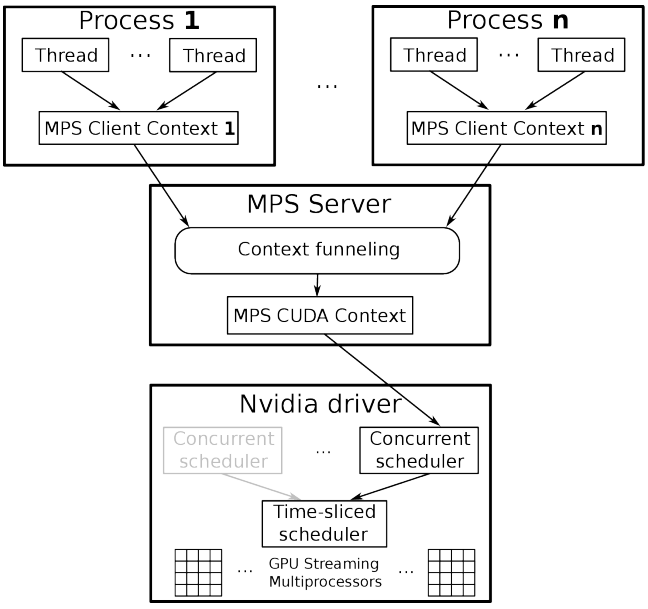

Figure 4: Client-server approach with a single CUDA context held and managed by the MPS server.

In this environment, using native SLURM and Shifter, it is trivial to share a node with different workflows and can run HEP jobs, as well as a number of other scientific applications on the same node and each will run within the boundaries of their own Shifter image.

To limit the impact of sharing resources within the same node with different scientific workloads, nodes that can be shared are separated on a specific partition and must be manually requested by users, or by ARC-CE in the case of WLCG. However, this dedicated partition overlaps with others and therefore users are not prevented from allocating nodes in exclusive mode up to the whole machine, if needed.

Running WLCG HEP jobs on our Cray systems with Shifter results on comparable performance, measured in job efficiency, as those on dedicated RHEL-compatible clusters.The following chart, extracted from ATLAS official dashboard, shows equivalent efficiency of two facilities as configured in the ATLAS environment for CSCS: `CSCS-LCG2-HPC_MCORE` (Cray TDS with Shifter) and `CSCS-LCG2_MCORE` (dedicated RHEL-compatible cluster).

5 Conclusions and Future Opportunities

In this paper, we demonstarted how Cray XC30 platform with GPU technologies can become more verstaile for applications and workflows that have so far been

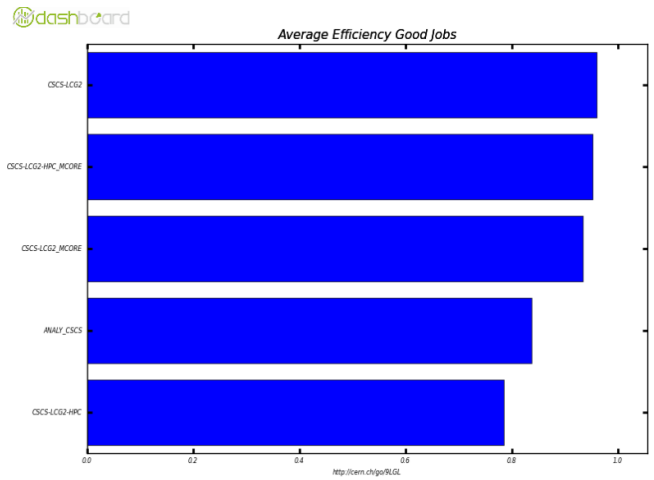

Figure 5: Comparable efficiency of jobs at CSCS facilities for ATLAS.

ruled out for the lightweight operating envirnment, which is is essential for high-end toghtly integrated systems. Furthermore, security concerns related to conatiner technologies are eliviated by Shifter. Two motivating examples, one with a GPU application and another with complex workflow has been presented in the paper. These examples show that not only minimal changes are required by users but also there is virtually no performance impact.

Our prelimary efforts yielded promising results and we are in process of exploring other opprtunities. These include but are not limited to:

- Tailored CUDA version: There are instances where for performance and functionality reasons multiple containers, i.e., CUDA processes, require different CUDA runtime versions. Possible use cases may target applications that require specific software stacks in order for the produced results to be certified, or even a case where a CUDA Toolkit upgrade would introduce a bug or regression in the application code. Conatiners offer an opprtunity for multiple versions to co-exist within a single Crau XC system.

- Extension to support GPU-to-GPU communication (GPUDirect): work in currently underway to allow GPUDirect inside multiple containers deployed over several compute nodes of the Cray XC30.

In short, with Shifter and Docker or container technologies users can package applications that require specific versions of software, and simply run them without further modifications. For instance, Python or Ruby applications can be tested on a local environment and then executed on a given Cray supercomputer without recompilation. This effectively broadens the type and kind

of application that can be run on Cray supercomputers and eases the access of these facilities to scientists. Scenarios in which users run their own services to support their own applications, all within their job allocations are also possible. This expands even further the possibilities that container solutions bring to Cray supercomputers and HPC facilities in general.

6 Acknowledgments

The authors would like to acknowledge contributions of the Shifter development team at NERSC and engineering team of the Nvidia CUDA division for their input and support.

References

[1] Shane Cook. *CUDA programming: a developer's guide to parallel computing with GPUs*. Newnes, 2012.

[2] Douglas M Jacobsen and Richard Shane Canon. Contain This, Unleashing Docker for HPC. In *Cray Users Group Conference (CUG'15)*, 2015.

[3] Nvidia Corporation. Multi-Process Service vR352. `http://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf`. Accessed: 2016-02-11.