rCUDA is currently available for Linux

rCUDA is available for the x86, ARM and Power processor architectures. Furthermore, rCUDA clients and rCUDA servers for different processor architectures can interact among them. This availability allows for different applications to run in different nodes with the illusion of dealing with local GPUs.

Seamless use of GPUs installed in remote computers

The rCUDA middleware virtualizes GPUs located in different nodes of the cluster, creating a pool of GPUs that can be shared among them. Consequently, rCUDA enables convenient access to them, thus allowing transparently programming and applications. Furthermore, application source code does not need to be modified in the way rCUDA provides applications with the illusion of dealing with multiple GPUs.

High energy savings by sharing the GPUs installed in the single node

rCUDA is binary compatible with CUDA. Just install rCUDA in your cluster and keep using your applications as usual!

How rCUDA works

The rCUDA middleware leverages the client-server architecture. The rCUDA client is a normal library which is installed in the cluster node executing the CUDA applications. The rCUDA client accesses rCUDA’s databases in the node. The server side of rCUDA acts as computer managing (GDI) the GPUs. In order to use rCUDA, applications do not need to be modified. They just include the rCUDA libraries instead of the CUDA ones.

The rCUDA client provides applications with the illusion of dealing with local GPUs. When a CUDA application runs on a remote node, the rCUDA client forwards the request to the server, acting as the actual GPU. The rCUDA server receives these requests and forwards them in the actual GPUs.

rCUDA is multiplatform

Make your GPUs even better with rCUDA!

Save acquisition and energy costs

rCUDA is binary compatible with CUDA. Just install rCUDA in your cluster and keep using your applications as usual!

How rCUDA works

The rCUDA middleware leverages the client-server architecture. The rCUDA client is a normal library which is installed in the cluster node executing the CUDA applications. The rCUDA client accesses rCUDA’s databases in the node. The server side of rCUDA acts as computer managing (GDI) the GPUs. In order to use rCUDA, applications do not need to be modified. They just include the rCUDA libraries instead of the CUDA ones.

The rCUDA client provides applications with the illusion of dealing with local GPUs. When a CUDA application runs on a remote node, the rCUDA client forwards the request to the server, acting as the actual GPU. The rCUDA server receives these requests and forwards them in the actual GPUs.

rCUDA is multiplatform

Make your GPUs even better with rCUDA!

Save acquisition and energy costs

rCUDA is binary compatible with CUDA. Just install rCUDA in your cluster and keep using your applications as usual!

How rCUDA works

The rCUDA middleware leverages the client-server architecture. The rCUDA client is a normal library which is installed in the cluster node executing the CUDA applications. The rCUDA client accesses rCUDA’s databases in the node. The server side of rCUDA acts as computer managing (GDI) the GPUs. In order to use rCUDA, applications do not need to be modified. They just include the rCUDA libraries instead of the CUDA ones.

The rCUDA client provides applications with the illusion of dealing with local GPUs. When a CUDA application runs on a remote node, the rCUDA client forwards the request to the server, acting as the actual GPU. The rCUDA server receives these requests and forwards them in the actual GPUs.
**rCUDA for Cloud**

Using GPUs in cloud computing has become increasingly mainstream. CUDA applications running inside a virtual machine (VM) can be accelerated by using the GPUs in the host. However, assigning GPUs to VMs presents several challenges, and using the PCI passthrough mechanism effectively shares the GPUs among the VMs.

**More GPUs with rCUDA**

rCUDA abstracts GPUs from the nodes where they are isolated. This means that GPUs can be used by more nodes than the number of non-VM applications to use all the GPUs of the cluster for a single node.

**rCUDA and Slurm**

Slurm is also able to deal with the virtual GPUs provided by rCUDA. The virtual GPUs are isolated from the Slurm queues: only one Slurm job can use all the requested amount of GPUs. rCUDA for Slurm can assign the requested amount of GPUs on the node, without the need for a separate mechanism based on the sharing of the GPU from the node. rCUDA allows to share remote GPUs among several VMs, thus creating a pool of GPUs that can be concurrently shared by all the VMs.

**The need for integrating rCUDA and Slurm**

Combining rCUDA with Slurm provides large benefits in overall cluster throughput as well as important reductions in energy consumption.

**Cheaper cluster upgrades with rCUDA**

Using rCUDA for providing GPUs to VMs allows for an increased number of VMs per node. This can be achieved by attaching to them remote CUDA GPUs, thus making use of several remote GPUs.

**GPU migration with rCUDA**

Using GPU migration for load balancing can take place in several scenarios: either GPUs can be migrated from one node to another, or from one node to a cluster node. This allows an important reduction in energy consumption, as GPUs that are not used can be migrated to other nodes, thus reducing energy consumption.

---

**How can a VM access CUDA GPUs?**

rCUDA is able to virtualize the hardware resources by means of the PCI passthrough technique. This technique allows assigning a PCI device to a VM as an exception way. Thus, a GPU assigned to a VM cannot be assigned to other VMs until the device is detached from the former VM. This is an important feature of rCUDA, as it allows sharing of a server with four GPUs. Each GPU is assigned to one VM.

**How does rCUDA avoid PCI passthrough?**

rCUDA is designed following a different mechanism. Therefore, rCUDA clients can be placed inside VMs whereas the CUDA environment is placed in the node where the GPU resides. Using the PCI passthrough mechanism is not required. As a consequence, GPUs can be concurrently shared by at least two VMs.

**Access all the GPUs in the cluster from a single node**

Thanks to rCUDA, the amount of GPUs that can be assigned to a single non-MPI application is not limited by the number of GPUs attached to the physical machine running the application. The node where the application is executed is not limited by the number of GPUs attached to the physical machine running the application. In this way, the node can be reduced to the amount of GPUs installed in the single node.

**Which performance can be expected?**

Four VMs were created in a node with four GPUs. Each of the VMs was assigned one GPU and executed the same CUDA-based application for four results. Results are shown in the graph below. Each result consists of 1000000 iterations. Moreover, rCUDA was used in order to share the four GPUs among the VMs.

**TensorFlow gets accelerated by using many GPUs**

Using rCUDA, TensorFlow gets accelerated by using many GPUs. TensorFlow is able to process data in parallel, making use of several remote GPUs.

---

**Using GPU migration for server consolidation**

When several applications share a given GPU in the cluster, rCUDA allows to migrate jobs and can balance load by migrating some of the applications to other GPUs. When several applications share a given GPU in the cluster, rCUDA allows to migrate jobs and can balance load by migrating some of the applications to other GPUs. This allows important reductions in energy consumption.

---

**Using GPU migration for load balancing**

SLURM can only assign that job nodes owning the requested amount of GPUs, neglecting the benefits of using several remote GPUs, as all the non-VM applications have to share the four GPUs installed in the single node. Thus, several applications sharing the same GPU may be similar to that depicted in the image below. The figure on the right shows the migration time of an application when using rCUDA. Each arrow shows how much time it takes to migrate an application. The figure on the right shows how much time it takes to migrate an application. The figure on the right shows how much time it takes to migrate an application.

---

**SLURM**

SLURM is a job scheduler for Linux clusters. SLURM makes use of several remote GPUs.

---

**Made-to-Measure**

The term "Made-to-Measure" refers to a type of product development where the product is designed and manufactured to meet the specific needs and requirements of a customer, as opposed to being a standard or mass-produced item. In the context of this document, it likely refers to a custom or customized solution, possibly in the realm of cloud computing or high-performance computing, where the solution is tailored to the specific needs of the customer.

---

**Introduction**

Considering the performance of the different CUDA implementations, it can be seen that the use of remote GPUs can significantly improve the performance of the applications. Moreover, the use of remote GPUs can reduce the energy consumption, as GPUs that are not used can be migrated to other nodes.

---

**Execution Time (s)**

The chart below shows the execution time of an application with and without GPU migration. The chart on the right shows the execution time of an application with and without GPU migration. The chart on the right shows the execution time of an application with and without GPU migration.