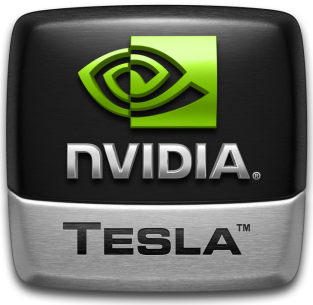
Scheduling Jobs onto NVIDIA Tesla GPU Computing Processors using PBS Professional

October 2010

www.pbsworks.com
www.nvidia.com/tesla



Abstract

With the advent of the Graphical Processing Unit (GPU) as a general-purpose computing unit, more and more customers are moving towards GPU-based clusters to run their scientific and engineering applications. This model allows users to use a CPU and GPU together in a hybrid computing model, where the sequential part of the application runs on the CPU and the computationally-intensive part runs on the GPU. By exploiting the massive parallelism in GPUs, customers can run applications almost forty percent faster, compared to the traditional CPU-based mode. This technical paper explains and compares two different approaches to configuring and scheduling jobs onto NVIDIA® Tesla™ GPUs using PBS Professional®, the industry-proven high-performance computing workload management solution.

Introduction to PBS Professional

PBS Professional is the professional version of the Portable Batch System (PBS), a flexible workload management solution, originally developed to manage aerospace computing resources at NASA. PBS Professional has since become the leader in supercomputer workload management and the de facto standard on Linux clusters.

Today, growing enterprises often support hundreds of users running thousands of jobs across many different types of machines in diverse geographical locations. In this distributed heterogeneous environment, it can be extremely difficult for administrators to collect detailed, accurate usage data, or to set system-wide resource priorities. As a result, many computing resources are under-utilized, while others are over-utilized. At the same time, users are confronted with an ever-expanding array of operating systems and platforms. Each year, scientists, engineers, designers and analysts must waste countless hours learning the nuances of different computing environments, rather than focusing on their primary goals. PBS Professional addresses these problems for computing-intensive industries such as science, engineering, finance and entertainment. Now you can use the power of PBS Professional to better control your computing resources. This allows you to unlock the potential in the valuable assets you already have, while at the same time reducing the load on system administrators and operators, freeing them to focus on other activities. PBS Professional can help you effectively

manage growth by tracking real usage levels across your systems and allowing you to target future purchases to your needs.

PBS Professional consists of two major component types: commands and daemons / services. A brief description of each is given here to help you understand how the pieces fit together.

Commands – PBS Professional supplies both a graphical interface and a set of POSIX 1003.2d-conforming command-line programs. These are used to submit, monitor, modify and delete jobs. These client commands can be installed on any system type supported by PBS and do not require local presence of any of the other components of PBS.

There are three command classifications: commands available to any authorized user, commands requiring special PBS privilege, and administrator commands. Administrator commands require root access or the equivalent to use.

Server – The *Server* daemon/service, *pbs_server*, is the central focus of PBS Professional. All commands and other daemons/services communicate with the Server via an Internet Protocol (IP) network. The Server's main function is to provide the basic batch services such as receiving/creating a batch job, modifying the job, and passing the job to the execution node. One Server manages the machines and jobs in a PBS complex; a secondary Server may be configured to handle failover.

Execution Node Manager (MOM) – The *MOM* is the daemon/service which actually places the job into execution. The *pbs_mom* daemon is informally called MOM as it is the mother of all processes for jobs. (MOM is a reverse-engineered acronym that stands for Machine Oriented Mini-server). MOM places a job into execution when it receives a copy of the job from the Server. MOM creates a new session for each job and gathers information about the resource usage of that job. MOM also has the responsibility for communicating with all MOMs assigned to the job and returning the job's output to the user when directed to do so by the Server. One MOM runs on each execution host.

Scheduler – The job *Scheduler* daemon/service, *pbs_sched*, implements the site's scheduling policy, controlling when each job is run and on which resources. The Scheduler may communicate with the various MOMs to query the state of system resources and with the Server for availability of jobs to execute.

Introduction to GPU computing

General-purpose computing on graphical processing units is a technique for using a GPU, which typically handles computation only for computer graphics, to perform computations that were traditionally handled by the CPU. It is made possible by adding programmable stages and higher-precision arithmetic to rendering pipelines, allowing software developers to use stream processing on non-graphics data.

This technique converts the massive floating-point computational power of a modern graphics accelerator's shader pipeline into general-purpose computing power. In applications requiring massive vector operations, this can yield several orders of magnitude higher performance than can be achieved with a conventional CPU. NVIDIA, the inventor of GPUs, has led this new approach with an array of applications.

NVIDIA has teamed up with national laboratories as well as commercial software leaders to create GPU-based applications to accelerate results. One area of science that benefits greatly from GPU acceleration is molecular dynamics simulation. NVIDIA partnered with Sandia National Laboratories, a US Department of Energy laboratory, to accelerate LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator). An application widely used by research scientists, LAMMPS running on NVIDIA's Tesla GPUs outperforms the same code running on CPUs by up to 40x. To learn more about GPU-acceleration on life-science codes, visit www.nvidia.com/bio_workbench.

Commercial application users around the world can also enjoy the tremendous acceleration of GPU-acceleration on MATLAB, a technical computing environment with users in a wide range of vertical markets such as aerospace/defense, automotive, biotech/pharmaceutical, electronics/semiconductors, and financial services. Users of MATLAB can now perform computationally intensive tasks faster than ever before by

Tesla GPUs. For examples of how GPU-acceleration is impacting the speed of results for MATLAB users, www.accelereyes.com/successstories.

In the last several years, NVIDIA has been releasing GPUs built on the CUDA™ parallel computing architecture, supporting an API extension to several programming languages. This enables specified functions from a normal program to run on the GPU's stream processors, and allows programs to take advantage of a GPU's ability to operate on large matrices in parallel, while still making use of the CPU when appropriate. With CUDA, CPU-based applications can directly access resources of a GPU for more general purpose computing, without the limitations of using a graphics API.

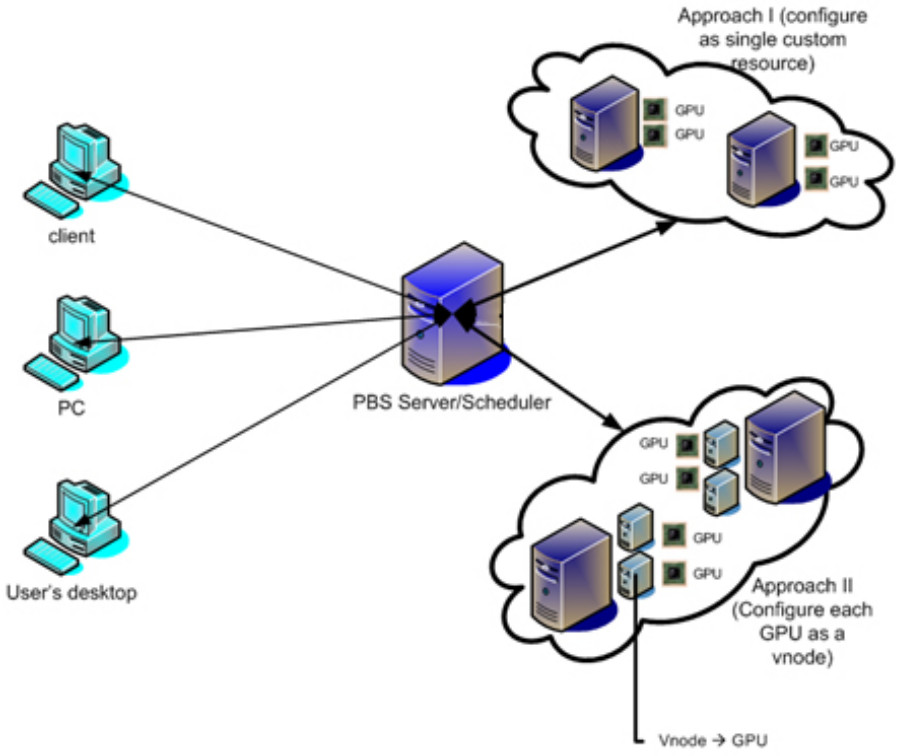


Figure1: Block diagram showing PBS Server scheduling job onto GPUs

Configuring GPUs for use in PBS Professional

PBS supports both basic and advanced GPU scheduling. Basic scheduling consists of prioritizing jobs based on site policies, controlling access to nodes with GPUs, ensuring that GPUs are not over-subscribed, and tracking use of GPUs in accounting logs. Configuring PBS to perform basic scheduling of GPUs is relatively simple, and only requires defining and configuring a single custom resource to represent the number of GPUs on each node. Approach 1 describes basic GPU scheduling.

Although basic GPU scheduling will meet the needs of 95% of customers, PBS also supports more advanced GPU scheduling: the ability for a job to separately allocate (request and/or identify) each individual GPU on a node. This capability is useful for sharing a single node among multiple jobs, where each job requires its own GPUs. In this case, both PBS and the applications themselves must support individually allocating a node's GPUs. This more advanced scheduling requires defining a "PBS vnode" for each GPU. Approach 2 describes advanced GPU scheduling.

This section covers both approaches on how to configure GPUs in PBS Professional, and also compares the pros and cons of each approach. Note that in order to schedule jobs onto GPUs, the administrator must manually configure GPUs as resources. Note also that PBS Professional allocates GPUs, but doesn't bind jobs to any particular GPU; the application itself (or the CUDA library) is responsible for the actual binding.

Approach 1 – Basic GPU Scheduling

Configure all GPUs as a single custom resource

The administrator adds only one custom resource in this approach. PBS Professional treats all GPUs with equal priority (similar to the built-in *ncpus* PBS resource) and doesn't bind jobs to any particular GPU.

Steps to configure using single custom resource approach:

Assuming there are two execution hosts, *nodeA* and *nodeB,* present in the cluster, and each execution host has 4 GPU devices, the administrator configures a single consumable custom resource "*ngpus"* to represent all GPU devices on the execution hosts.

The administrator then configures the *ngpus* custom resource in the following way:

1. Stop the PBS Professional server and scheduler. On the server's host, type:
   /etc/init.d/pbs stop

2. Edit *$PBS_HOME/server_priv/resourcedef* to add the following line:
   ngpus type=long flag=nh

3. Edit *$PBS_HOME/sched_priv/sched_config* to add *ngpus* to the list of scheduling resources:
   resources: "ncpus, mem, arch, host, vnode, ***ngpus"***
4. Restart the PBS Professional server and scheduler. On the server's host, type:
   /etc/init.d/pbs start

5. Add the number of GPU devices available (in our example, this number is 4) to each execution host in the cluster via *qmgr:*
   qmgr –c "set node nodeA resources_available.ngpus=4"
   qmgr –c "set node nodeB resources_available.ngpus=4"

6. Submit a one-node job (e.g., "my_gpu_job") on 2 GPUs in the following manner:
   *qsub –lselect=1:ncpus=1:****ngpus=2*** *my_gpu_job*

The administrator can find the number of GPUs available/assigned on the execution hosts via the '*pbsnodes'* command.

Approach 2 – Advanced GPU Scheduling

Configure GPUs as vnodes

In this approach, the administrator configures each GPU device in its own vnode, including the GPU device number.

Steps to configure GPUs as virtual nodes with device number:

1. Stop the PBS Professional server and scheduler. On the server's host, type:
   `/etc/init.d/pbs stop`

2. Edit *$PBS_HOME/server_priv/resourcedef* to add two new custom resources, *ngpus* and *gpu_id*:
   `ngpus type=long flag=nh`
   `gpu_id type=string flag=h`

3. Edit *$PBS_HOME/sched_priv/sched_config* to add *ngpus* and *gpu_id* to the list of scheduling resources:
   resources: "ncpus, mem, arch, host, vnode, ***ngpus, gpu_id"***

4. Restart PBS Professional services. On the server's host, type:
   `/etc/init.d/pbs start`

5. Create a vnode configuration for each execution host where GPUs are present. In this example, we create a script named "nodeA-vnodes" for "nodeA" which has 4 CPUs, 2 GPUs, and 16 GB of memory:

```
$configversion 2
nodeA: resources_available.ncpus = 0
nodeA: resources_available.mem = 0
nodeA[0]: resources_available.ncpus = 2
nodeA[0] : resources_available.mem = 8gb
nodeA[0] : resources_available.ngpus = 1
nodeA[0] : resources_available.gpu_id = gpu0
nodeA[0] : sharing = default_exclusive
nodeA[1] : resources_available.ncpus = 2
nodeA[1] : resources_available.mem = 8gb
nodeA[1] : resources_available.ngpus = 1
nodeA[1] : resources_available.gpu_id = gpu1
nodeA[1]: sharing = default_exclusive
```

6. Add vnode configuration information to PBS Professional in the following manner (for each node with GPUs):
   $PBS_EXEC/sbin/pbs_mom –s insert nodeA-vnodes nodeA-vnodes

7. Signal each *pbs_mom* to reread the configuration files:
   kill –HUP <pbs_mom PID>

8. Submit job (e.g., “my_gpu_job”) requesting one node with one GPU:
   qsub –lselect=1:ncpus=1:ngpus=1 my_gpu_job

The PBS Scheduler looks for a vnode with an available GPU and assigns that vnode to the job. Since there is a 1-1 correspondence between GPUs and vnodes, the job can now ask PBS which vnode it was given, and determine the *gpu_id* of that vnode. Finally, the application can use the appropriate CUDA call to bind the process to the allocated GPU.

Alternatively, one can submit a job requesting a particular *gpu_id* as well. The following requests 4 nodes, each with GPU id 0:

qsub –lselect=4:ncpus=1:gpu_id=gpu0 my_gpu_job

This provides a high degree of control for users and administrators, allowing them to run their applications based on the devices for which their applications are programmed.

Comparison of Basic and Advanced GPU Scheduling

	Approach 1	Approach 2
Configuration & Administration	Easy and centralized. Must make configuration changes only in PBS Server and Scheduler.	More complicated and distributed. Must make configuration changes on all nodes with GPUs.
Granularity	All GPUs are given equal priority. The user does not specify GPU on which to run job.	Users can select GPUs by specifying their device numbers.
Number of custom resources configured	Only one	Two, plus virtual nodes on all MOMs based on number of GPUs
Uses	Good choice when jobs do not share nodes (i.e., only one GPUs job at a time is run on any given node, exclusively).	Good choice when sharing nodes by multiple jobs at the same time is required, or individual access to GPUs (by device number) is required.

Conclusion

The two approaches suggested in this paper provide the techniques necessary to configure and schedule jobs onto NVIDIA Tesla GPUs using PBS Professional. This document provides detailed information on how to make efficient use of hundreds of computing cores in Tesla GPUs with PBS Professional. In closing it should be noted that users can check the number of GPUs assigned and requested for the job in the PBS accounting log file, or via PBS Professional commands such as "*pbsnodes*". It should also be noted that none of the approaches specified in this paper tries to pin tasks to particular GPU devices. As with normal CPU-based jobs, assignment of tasks to cores is handled by the operating system services (or CUDA library).