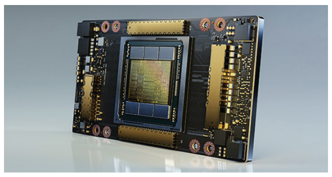
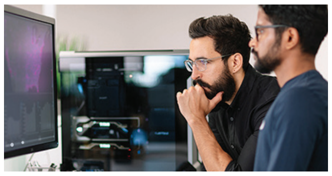
ALTAIR® GRID ENGINE® SUPPORT FOR NVIDIA® DGX™ SYSTEMS

Developed to meet the needs of demanding high-performance computing (HPC), artificial intelligence (AI), and analytics workloads, NVIDIA® DGX™ systems are built on the revolutionary NVIDIA A100 and V100 Tensor Core GPU platforms. By using Altair® Grid Engine® to manage GPU workloads on DGX systems, organizations can boost performance, use resources more efficiently and improve overall productivity.

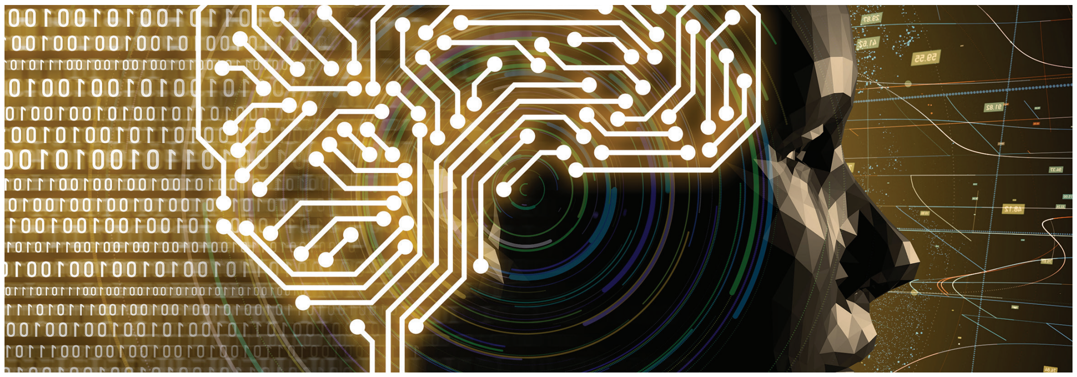
1. About NVIDIA DGX Systems

NVIDIA DGX systems are a family of products from NVIDIA purpose-built for deep learning applications. The NVIDIA DGX family includes the NVIDIA DGX Station™, NVIDIA DGX-1™ and DGX-2™ rackmount servers, and the newer NVIDIA DGX™ A100 system.


NVIDIA A100 GPUs deliver unprecedented acceleration


Purpose-built to meet the needs of demanding HPC, AI, and analytics workloads

In AI data centers, managing distributed GPU-powered machine learning frameworks is a central challenge. Data scientists run diverse workloads ranging from data preparation to model training to model validation to inference. Workloads need to run quickly, use resources efficiently, and be deployed considering factors such as CPU and GPU architecture, memory, cache, bus topologies, and NVIDIA interconnect and network switch topologies.



The latest generation of NVIDIA DGX A100 systems integrate 8 NVIDIA A100 Tensor Core GPUs with an NVIDIA NVLink™ powered NVSwitch™ fabric. When configured with 80GB A100 GPUs, these systems deliver over 3X the performance of an NVIDIA DGX-2 system based on the standard deep learning recommendation model (DLRM) for PyTorch benchmark. NVIDIA DGX A100 systems are 6U servers that can be deployed individually or as part of the NVIDIA DGX POD or DGX SuperPOD reference architecture. Each DGX SuperPod cluster has 140 x DGX A100 systems for a total of 1,120 GPUs. DGX multi-system configurations employ NVIDIA InfiniBand switched fabric with 8 x 200 Gb/s connections per DGX A100 server.

2. Support for NVIDIA DGX Systems in Altair Grid Engine

Altair Grid Engine is an enterprise-class workload scheduling and management solution used across many industries for applications that include machine learning, deep learning, and HPC. Altair Grid Engine provides rich support for scheduling GPU-aware applications and containers. It also features a direct integration with NVIDIA Data Center GPU Manager (DCGM), making it an ideal workload manager for NVIDIA DGX environments. Regardless of whether you are using a single DGX system or are deploying a cluster comprised of thousands of GPUs, Altair Grid Engine provides rich workload and resource management capabilities for your NVIDIA DGX environment.

3. Introduction to Altair Grid Engine

Altair Grid Engine is a leading solution for workload and distributed resource management. It runs across clusters, clouds, and supercomputers, improving workload throughput, system efficiency, and overall productivity. With advanced support for containers and GPUs, Altair Grid Engine is well suited to GPU-aware applications running on NVIDIA DGX systems.

Solve bigger, more complex data science problems faster

Architecture and Components – Altair Grid Engine supports a client-side command-line interface (CLI) and a set of system daemons/services that run across nodes in an Altair Grid Engine cluster. Altair Grid Engine runs on Linux®, Microsoft Windows®, and a variety of UNIX® operating environments. A brief description of Altair Grid Engine's various components is provided below to help readers understand how the pieces fit together.

Command-line Interface – Altair Grid Engine provides a comprehensive set of programs that can be run from the command line to support various user, administrative, and operational tasks. Altair Grid Engine complies with the POSIX 1003.2d standard, so basic user commands and syntax will be familiar to Altair® PBS Professional® users. The Altair Grid Engine architecture is significantly different from PBS Professional, and the administrative commands are also different.

Master Host – In an Altair Grid Engine cluster, the qmaster is the central control and information point. The qmaster process is called sge_qmaster for historical purposes. The qmaster keeps track of all status information in the cluster including load indices across all execution hosts, running and queued jobs, users and their roles, and various policies and configuration details. The qmaster is also responsible for scheduling, matching resources with job requirements, planning resource assignment schedules, and handling resource reservations. The qmaster is multi-threaded to maximize responsiveness and scheduling performance. To ensure that the qmaster can be recovered at any time, or fail over to another host, all activity is logged to a shared filesystem or to a Berkeley DB database depending on the cluster administrator's preference.

Shadow Master Hosts – The qmaster is a potential central point of failure, so Altair Grid Engine environments are often deployed with one or more shadow master hosts to ensure continuous operation if the master host fails. Shadow master hosts are optional. Customers may choose to run one or multiple shadow hosts depending on their desired level of redundancy. A lightweight sge_shadowd process runs on the execution host responsible for monitoring sge_qmaster on the master host and starting an instance of sge_qmaster on a shadow host if the master host fails. If multiple shadow daemons are active in the cluster, they negotiate with one another to ensure that only a single instance of the sge_qmaster is started.

Execution Host – In an Altair Grid Engine environment, execution hosts are where various workloads are executed. A daemon called sge_execd runs on each execution host. It tracks and reports host resource utilization (CPU load, memory usage, etc.) and the state of jobs on each node.

Workload Types – When a job is launched, sge_execd spawns an sge_shepherd process for each job responsible for starting, monitoring, and managing various jobs. Altair Grid Engine supports a variety of different workload types. Among the workloads supported are:

- Batch jobs
- Parallel jobs
- Interactive jobs
- Array jobs
- Checkpoint jobs
- Immediate jobs
- Containerized jobs
- GPU jobs

4. Obtaining Altair Grid Engine

Visit the Altair website to obtain a copy of Altair Grid Engine or request a free software trial. Please note that there are open-source versions of Grid Engine available. However, open-source Grid Engine does not include the GPU-aware scheduling features required by NVIDIA DGX systems.

5. Before You Start

Altair Grid Engine is typically installed on a cluster comprised of one or more NVIDIA DGX systems connected by a TCP/IP network. It should be installed by a Linux system administrator familiar with the NVIDIA DGX server environment. You will need to perform some prerequisite steps before installing Altair Grid Engine. These include selecting a master host, ensuring that the TCP/IP network is properly configured, and setting up a shared NFS file system accessible by all NVIDIA DGX hosts. Consult the Altair Grid Engine Installation Guide for details.

Altair Grid Engine will work the Ubuntu® LTS operating system pre-installed on NVIDIA DGX systems. It is also supported on most other Linux operating systems including Red Hat® Enterprise Linux®, SUSE® Linux® Enterprise Server, CentOS®, and Oracle® Linux®. Details of supported operating systems can be found in the Altair Grid Engine Release Notes. These instructions assume that you are running Altair Grid Engine 8.6.5 or later. The steps described in sections 6 to 8 of this guide are essential for configuring GPU and DCGM support in Altair Grid Engine on DGX systems. Setting up Docker as described in section 9 and CPU-GPU affinity as described in section 11 are optional.

The NVIDIA DGX A100 system includes 8 GPUs, 6 NVIDIA NVSwitches, 10 Mellanox network interfaces, dual 64-core AMD CPUs, 2TB system memory, and 30TB Gen4 NVMe SSD

6. Enabling Cgroups

Cgroups (control groups) is a feature available in modern Linux distributions that limits the resources that a process or set of processes can access. Altair Grid Engine supports multiple features related to cgroups. The qconf command is used by Altair Grid Engine administrators to display or modify the configuration of the cluster. Configuration changes can either be made globally such that they apply to all cluster hosts, or individually. Individual host settings will override the global settings.

On NVIDIA DGX systems cgroups needs to be enabled by setting the cgroup_path and the devices parameters so that Altair Grid Engine will manage access to all GPUs on the host. This will block all devices from /dev/nvidia0 to /dev/nvidia254 and make them unavailable to users and processes running outside of Altair Grid Engine.

To run qconf you will need to be logged in as the Altair Grid Engine administrator account specified when you installed it. You can run qconf from any cluster host. When you run qconf, you will be placed in the vi editor (or the editor pointed to by $EDITOR). You will need to add or modify the cgroup_params line as shown below. After making changes, you will need to save any changes in vi to return to the Linux shell.

```
$ qconf -mconf <hostname>
  cgroup_params  cgroup_path=/sys/fs/cgroup \
                 devices=/dev/nvidia[0-254]
```

A host will not have 255 GPUs. The syntax above in cgroup_params ensures that Altair Grid Engine will manage all the GPUs.

7. Enabling DGCM Support

Altair Grid Engine version 8.6.0 and later are integrated with NVIDIA's Data Center GPU Manager (DCGM). DCGM provides detailed information about GPU resources. With this integration, Altair Grid Engine has full visibility into GPUs on each host including GPU type and version; available memory; operating temperature; and socket, core, and thread affinity. This information helps Altair Grid Engine schedule GPU-aware applications more efficiently to optimize both performance and resource use. As explained earlier, Altair Grid Engine runs a daemon on each cluster host called execd. DCGM support is enabled by setting the execd parameter UGE_DCGM_PORT to the port DCGM uses to communicate on each host, 5555 by default. You can run the qconf command as the Altair Grid Engine admin to set the UGE_DCGM_PORT for each cluster host where DCGM is installed.

```
$ qconf -mconf <hostname>
  ..
  execd_params    UGE_DCGM_PORT=5555
```

If all hosts in your cluster have GPUs and DCGM installed, you can run the command below once to apply this setting globally to all cluster hosts.

```
$ qconf -mconf global
  ..
  execd_params    UGE_DCGM_PORT=5555
```

NVIDIA DGX A100 systems can be deployed individually or as part of the NVIDIA DGX SuperPOD Solution for Enterprise

8. Setting up GPU Resources

Altair Grid Engine uses a resource map construct called an RSMAP complex to manage access to consumable resources on a host — those with a finite supply, such as memory, free space on a file system, floating software licenses, or GPUs. Before RSMAPs can be used, a consumable GPU resource of type RSMAP must be added in Altair Grid Engine as shown. As before, the qconf command should be run as the cluster administrator and vi used to enter or edit parameter settings. Details about what these settings mean are provided in the sge_complex man page.

```
$ qconf -ace gpu

  name         gpu
  shortcut     gpu
  type         RSMAP
  relop        <=
  requestable YES
  consumable  YES
  default      0
  urgency      0
  aapre        NO
  affinity     0.000000
```

Each id of the RSMAP complex can be configured to represent a GPU device on a cluster host as shown below. The device parameter is needed to facilitate blocking using cgroups. The cuda_id provides mapping between the device as it appears in DCGM and the physical device visible to Altair Grid Engine. The same cuda_id is also used by nvidia_smi.

The qconf -me command is used to modify the configuration of each execution host. For an NVIDIA DGX-2 system with 16 GPUs, a configuration will look like this:

```
$ qconf -me dgx2-1

  hostname      dgx2-1
 ..
  complex_values  \
  gpu=16(gpu0[device=/dev/nvidia0,cuda_id=0] \
         gpu1[device=/dev/nvidia1,cuda_id=1] \
         gpu2[device=/dev/nvidia2,cuda_id=2] \
         gpu3[device=/dev/nvidia3,cuda_id=3] \
         gpu4[device=/dev/nvidia4,cuda_id=4] \
         gpu5[device=/dev/nvidia5,cuda_id=5] \
         ...
         gpu15[device=/dev/nvidia15,cuda_id=15])
```

Each GPU can optionally be represented by more than one complex/id. This is discussed in section 13 dealing with GPU sharing/oversubscription.

9. Installing and Configuring Docker

To submit Docker jobs requiring access to GPUs, the NVIDIA Container Toolkit (a.k.a. nvidia-docker2) must be installed on each cluster host. Docker and NVIDIA runtime are installed on NVIDIA DGX systems by default, but if you use a different operating system, you will need to install these components yourself. You will need a Docker image with NVIDIA GPU support containing your application. You can use the NVIDIA-supplied CUDA images or other images derived from these base images. There are a variety of CUDA images available in the NGC Catalog.

Altair Grid Engine has built-in support for Docker and the NVIDIA Container Toolkit, allowing you to manage containerized GPU workloads just as you would manage any Grid Engine job. When submitting a containerized GPU job to Altair Grid Engine, you should specify the --gpus flag. This will result in the NVIDIA runtime being selected. Running a containerized TensorFlow training model using Altair Grid Engine to select a specific GPU will look something like this:

```
$ qsub -l docker,\
          docker_images="*tensorflow:18.03-py2*",\
          gpu=1(V100)[affinity=true]\
          --gpus all -b y\
      -S /bin/sh <command-inside-container>
```

The -l switch specifies resource requirements. In this example we require a host with the Docker runtime installed and we express a preference that a particular docker image be available (Altair Grid Engine will download the container automatically if it is not available on the host). We also require a host with a single available V100 GPU.

Managing distributed GPU-powered machine learning frameworks is a central challenge in AI data centers

10. Submitting Jobs

In this section, we provide some simple examples of running different kinds of jobs under Altair Grid Engine on an NVIDIA DGX cluster. The qsub command in Altair Grid Engine is used to submit jobs to the cluster. Details are available in the Altair Grid Engine User's Guide.

Submitting Regular Jobs – Non-GPU jobs run on the NVIDIA DGX cluster just as they would on any other Altair Grid Engine cluster. Jobs that attempt to access a GPU at runtime without specifically requesting a GPU will be blocked by cgroups. In this example, we submit an array of five non-GPU tasks numbered 2,4,6,8,10 to a priority queue:

```
$ qsub -t 2-10:2 -l q=priority array.sh
```

Submitting Jobs Requiring GPUs – Jobs can request access to a GPU using the -l switch on the qsub command line. Resource requests can include the resource name, the amount of resource, the name or ID of the resource, and directives related to affinity.

The general syntax is:

```
-l <cplx_name>="<amount>(<id>)[<affinity>]"
```

To submit a job that requires access to one GPU:

```
$ qsub -l gpu=1 job.sh
```

To request access to a specific type of GPU:

```
$ qsub -l gpu="1(P100)" job.sh
```

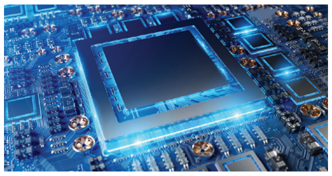
A GPU named "P100" must be configured in complex_values in the RSMAP for this example to work. When a user submits a job requesting a GPU as above, Altair Grid Engine responds with a job_id. The user can then use the qstat command with the returned job_id as shown to get status information about the job including information about the assigned GPU.

```
$ qstat -j <job_id>
  exec_host_list   1:dgx03:1
  granted_req.     1:gpu=1
  granted devices  1:dgx03: /dev/nvidia0
  resource map     1:gpu=dgx03=(V100)
```

Submitting Docker Jobs Requiring GPUs – Docker jobs requiring GPU resources must request a Docker host with the required docker image and specify the number of GPUs required for the containerized workload. In the example below we require two GPUs and a host where the Docker resource is set to true (meaning Docker is installed), and we would prefer a host that already has the required cuda:9 docker image loaded to avoid needing to pull the image again from a repository. Altair Grid Engine has visibility to a list of all docker images available on each host, so a wildcard is used to match a specific image against the list.

```
$ qsub -l gpu=2,docker=1,docker_images="*cuda:9*"\
  --gpus all -xd "-env    NVIDIA_VISIBLE_DEVICES=${gpu(0)},${gpu(1)}"\
   job.sh
```

The --env NVIDIA_VISIBLE_DEVICES argument is specified on the Altair Grid Engine qsub command line to pass the environment variable NVIDIA_VISIBLE_DEVICES into the container. The placeholder values ${gpu(0)} and ${gpu(1)} are replaced by the actual GPU devices scheduled by Altair Grid Engine. The default for NVIDIA_VISIBLE_DEVICES is "all" so passing only the GPUs selected by Grid Engine ensures that there are no conflicts and workloads cannot interfere with one another.


Altair Grid Engine efficiently manages the most demanding GPU workloads on NVIDIA DGX systems

11. Requesting CPU-GPU Affinity

With GPU-aware applications, part of the workload runs on one or more CPU cores and part of it runs on a GPU. It is important that processor cores be close to a selected GPU and share a common switch or PCIe bus to maximize performance. When CPU cores are close to a GPU, they're described as having affinity, an important consideration for Altair Grid Engine scheduling decisions. Requesting CPU-GPU affinity is only possible when support for DCGM is enabled as outlined in section 7.

Part of the information that DCGM provides to Altair Grid Engine for each GPU is information about processor affinity. The affinity value in the example below shows the CPU sockets, cores, and threads on the host that have affinity to each GPU by displaying them in upper case. This example shows a dual-CPU host where each processor (socket) has 8 cores each with 2 threads. If a GPU workload is placed on the first GPU (cuda0), the best performance will be obtained by scheduling the CPU component of the job on the first 4 cores on the first socket.

```
host.cuda.0.affinity=SCTTCTTCTTCTTcttcttcttcttScttcttcttcttcttcttcttctt,
host.cuda.0.gpu_temp=36,
host.cuda.0.mem_free=16280.000000M,
host.cuda.0.mem_total=16280.000000M,
host.cuda.0.mem_used=0.000000M,
host.cuda.0.name=Tesla V100-PCIE-16GB,
host.cuda.0.power_usage=28.527000,
host.cuda.0.verstr=390.46,
host.cuda.1.affinity=ScttcttcttcttcttcttcttcttSCTTCTTCTTCTTcttcttcttctt,
host.cuda.1.gpu_temp=40
..
```

This is complicated behind the scenes, but DCGM and Altair Grid Engine hide this complexity from users. Jobs requiring GPUs can request that they are scheduled on a CPU-GPU combination with good affinity by using the optional affinity parameter as shown:

```
$ qsub -l gpu=1[affinity=true] gpu_job.sh
```

In Altair Grid Engine versions prior to 8.6.5, the affinity parameter can be 0/false (the default) or 1/true. In Altair Grid Engine 8.6.5 and later, affinity can also be set to "2" meaning that affinity is "nice to have" but not essential. If a job is submitted with affinity=2, Altair Grid Engine will attempt to schedule CPU cores and GPU devices with good affinity but will schedule the job anyway if the affinity requirement cannot be satisfied.

12. NVIDIA DGX A100 Systems and MIG

Customers using NVIDIA DGX A100 systems will be familiar with Multi-Instance GPU (MIG), a major feature of NVIDIA A100 GPUs that allow them to be securely partitioned into up to seven separate GPU instances for CUDA applications. MIG is beneficial for workloads that do not fully saturate a GPU's compute capacity. Rather than running a small GPU application and underutilizing the GPU, MIG allows multiple applications to share the same GPU while providing isolation to ensure that one client application cannot impact other clients. Details about MIG are provided in the NVIDIA Multi-Instance GPU User Guide.

Recent versions of Docker now support a flag “--gpus” that allows users running containerized workloads to select GPUs and MIGs with the same colon syntax described above. This functionality is supported in Docker API versions 1.40 and later. This is explained in the Docker Documentation describing Runtime options with Memory, CPUs, and GPUs. Altair Grid Engine 8.6.16 and later versions have partial support for MIG. Altair is continuing to enhance Altair Grid Engine to simplify the use of MIG. In Altair Grid Engine version 8.6.16 and later, the following syntax can be used to submit a job that starts a container with access to MIG 0 on GPU 0. This represents an initial step in fully supporting MIG.

NVIDIA DGX systems were designed to meet the demanding needs of HPC, AI, and analytics workloads

```
$ qsub -l docker=1,docker_images="*nvidia*" -xd "--gpus device=0:0" ...
```

Customers can make use of RSMAPs and the new --gpu flag to take advantage of MIG in Altair Grid Engine environments even while formal support for MIGs is evolving. Customers wishing to take advantage of MIG can contact Altair to obtain assistance.

13. GPU Sharing

In addition to NVIDIA A100 MIG functionality described above, GPUs can be “oversubscribed” even on DGX-1 and DGX-2 servers without support for MIG (i.e., one physical GPU can be mapped to two or more RSMAP ids/complexes). This way more than one job can run on a GPU. For example, a host has two NVIDIA V100s and two NVIDIA P100s installed. The V100 has twice the performance of the P100, so a cluster administrator may wish to indicate that each V100 can support up to two jobs at the same time. To achieve this, we can describe a resource complex for our four physical devices with six virtual entries, where each V100 GPU appears twice, each able to support up to two jobs.

```
qconf -me <host>
...
complex_values   gpu=6\
   (V100_0[device=/dev/nvidia0,cuda_id=0]\
    V100_0[device=/dev/nvidia0,cuda_id=0]\
    V100_1[device=/dev/nvidia1,cuda_id=1]\
    V100_1[device=/dev/nvidia1,cuda_id=1]\
    P100[device=/dev/nvidia2,cuda_id=2]\
    P100[device=/dev/nvidia3,cuda_id=3])
```

If we want to use half of a V100 GPU for a job, and we do not care which physical GPU is assigned, we can request a single V100 GPU as shown:

```
$ qsub -l gpu=1(V100_*) job.sh
```

If a job needs exclusive access to a physical V100 GPU, it needs to request both “slots” on the shared GPU. In this case, use the wildcard selector V100_* to obtain the correct GPU model, and use the XOR operator to make sure that the GPU slots allocated are on the same physical GPU. For example, to request two slots on a V100 and make sure that both slots are on the same physical device (2 x V100_0 slots or 2 x V100_1 slots) use the following syntax:

```
$ qsub -l gpu=2(V100_*^) job.sh
```

14. Learning More

For additional information about using Altair Grid Engine with NVIDIA DGX systems, please consult the Altair Grid Engine documentation. You will require the following documents:

- Altair Grid Engine Installation Guide
- Altair Grid Engine Administrator’s Guide
- Altair Grid Engine User’s Guide

For more information or technical support, please visiit **altair.com/contact-us**.