GPGPU Working Group
Background, Progress, Future Work

John Walsh

Background

- Initial survey work of GPGPU Task Force (Sept 2012)
  - Seperate User and Resource Centre (RC) surveys
  - Surveys had good response rates
    - Indicated increased GPGPU usage at RCs
    - Indicated future User Application use/development
- Formed after EGI Technical Forum September 2013
  - Simple goal of working toward full grid support (i.e.)
    - Discovery and Enumeration (GLUE + Info Providers)
    - JDL and Job Submission Support
    - Resource Accounting
  - Team drawn from User/MW Dev and Ops communities

Working Group

- Group composed of Users/RC admins/GPGPU experts
  - 20+ members registered
  - New contributors/RCs and input welcome
- Co-ordination and interaction:
  - Mailing list
    - vt-gpgpu (at) mailman.egi.eu
  - Wiki
    - https://wiki.egi.eu/wiki/GPGPU-WG
    - Knowledge-base
      - https://wiki.egi.eu/wiki/GPGPU-WG:GPGPU_Working_Group_KnowlegeBase
  - Phone Conferences
    - 3 Phone Conferences since December 2013

Integration Problem

- Problem statement is simple
  - However, full GPGPU integration is non-trivial!
    - Most popular (Torque/MAUI) LRMS doesn't support GPGPUs
    - Requires specification from diverse stakeholders
      - GLUE + Job Description
    - Transition of services to GLUE2
      - GLUE2 has many advantages
    - Time – we all have other duties!

Initial Naive Approach

- Focus on Nvidia/CUDA
- RC assign GPGPU nodes to a single Queue
  - Queue named with suffix “gpu”
- Sites advertises GLUE1 CUDA SoftwareEnvironment
- Node assigns 1 CPU per GPGPU
  - Job Slots ↔ GPGPU slots
  - SMPGranularity = #Physical GPGPU per Node
  - No implicit requirement for LRMS to handle GPGPU allocation

SWOT

- Naive approach
  - Limited use-cases can be dealt with.
  - Potential resource wastage (less real CPU slots available)
  - A priori info to setup local WN environment
- However,
  - It is a start (better than no start)
  - No M/W changes are required

Simple JDL

```
[
Executable = "myHello.sh";
StdOutput = "std.out";
StdError = "std.err";
InputSandbox =  {"myHello.sh"};
OutputSandbox = { "std.out","std.err"} ;
VirtualOrganisation = "gputestvo";
Requirements =  ( RegExp(".*gpu", other.GlueCEUniqueID)
&& Member("CUDA",
other.GlueHostApplicationSoftwareRunTimeEnvironment));
]
```

Simple JDL (2 GPGPUs)

```
[
# Should be equivalent to PBS -l nodes=1:ppn=2.
Type="Job";
JobType="Normal";
CPUnumber=2;
# myScript.sh must take responsibility for executing the GPGPU application on the allocated core/GPGPU pairs!
Executable = "myScript.sh";
StdOutput = "std.out";
StdError = "std.err";
InputSandbox =  {"myScript.sh"};
Requirements =  ( RegExp(".*gpu*", other.GlueCEUniqueID) && Member("CUDA", other.GlueHostApplicationSoftwareRunTimeEnvironment));
]
```

Advanced (Naive) Approach

- GPGPU cluster is advertised as a SubCluster
  - Can advertise GPU properties as if CPU
    - GPU Clock Speed
    - Vendor
    - ….
- Allows more advanced JDL Resource Requirements
  - E.G. Only consider “Nvidia” CPU Vendor
  - Not tested in production

JDL Limitations

- Quickly ran into a JDL “brick wall”
  - Use case: Moldyngrid application
    - Job required N-to-M CPU/GPU ratio
    - Required 8 GPGPUs and many CPUS
    - Required MPI
    - WMS list-match failed
      - However, ample number of real CPU resources were actually available
- Need to develop another approach!

Towards Advanced Support

Objectives (abstract)

- See if we can specify actual GPGPU resource requirements in JDL
- Correctly handle LRMS support for requested/specified GPGPUs
- Enable GPGPU Enumeration, Publication and discovery
- Support WMS or direct submission

Requesting Resources (JDL)

- Consider JDL specification **GPN_PN=N**
  - This is #GPGPUs Per Node
  - Independent of CPU request
  - Independent of batch system
- How do we turn this into a LRMS req?
  - Will review batch system GPGPU support
  - Will look JDL parameter passing in CREAM

Batch System Integration (review)

- Support for Nvidia/CUDA in batch systems
  - Torque 4 Nvidia management library support
  - SLURM supports Nvidia out of the box
  - LSF support GPGPUs/Intel Xeon Phi
  - SGE support(?)
- ATI support?
- Some support for Intel Xeon Phi

Torque/MAUI Experience

- Torque 2.5.7/MAUI most widely used batch system EGI
- Investigated latest torque 4.X and torque 2.5.7
  - Typical (torque-only) gpgpu request:
    - e.g. qsub test.sh -l nodes=1:ppn=2:gpus=1
  - However, MAUI silently drops the gpus=1 request
- Have tested patched MAUI
  - Patch enables “Generic Resources”
  - Developed by Jonathan Michalon (Uni Strasbourg)

MAUI GRES

- Running in testbed environment
  - Many months (seems stable)
  - RPM/SRPM available
    - Contact me (john.walsh at scss.tcd.ie)
  - Not just for GPGPUs
  - GPGPU gres is requested in torque
    - qsub -W "x=GRES:gpu@1"
  - Note: does not handle GPGPU allocation to physical dev. However, have written a unprivileged daemon that can help!

Magrathea

- CESNET have also developed a modification to torque to handle GPGPU access controls and process protection mechanisms:
  - http://www.metacentrum.cz/en/devel/torque/index.html

SLURM (I)

- SLURM support for Nvidia GPGPUs
- Works out of the box (slurm.conf configuration)
  - Auto setting CUDA_VISIBLE_DEVICES
  - Useful when >1 GPGPU on Physical Node
- SLURM batch GPGPU request (example)
  - Add batch system parameter “--gres=gpu:1”

SLURM (II)

- SLURM/CREAM-CE released April 2014
- Have installed in Testbed,
- First impression:
  - Quick & seems to perform well
- Only issues
  - handling Maxtime=infinite problem (GIP fail)
    - Still need to submit GGUS ticket
  - Sporadic direct job submission failures
    - Did not see this with WMS submission

Key Observation

- In batch system:
  - GPGPUs treated as class of Generic Res.
  - GRES not allocated in isolation of CPU(s)
- Will see later the influence on experimental GLUE2 publishing/discovery prototype

CUDA_VISIBLE_DEVICES

- Nvidia GPGPU devices usually configured with permissive access priviliges
- Concurrent user jobs could try to access same devices
  - Some COMPUTE_MODE settings:
    - Exclusive
    - Shared (thread/process modes)
  - CUDA_VISIBLE_DEVICES=X,Y,Z
    - Can restrict the users view of available devices
  - Attempted to set as readonly environment variable
    - Easy to bypass (e.g. launch child shell)
  - However, many HPC centres seem to allocate on a WholeNode basis

Conclusion (LRMS)

- Know what to pass into the LRMS
- Will now look at how how to request GPGPUs in JDL, and how the LRMS will process this request

JDL and LRMS

Towards CREAM support

- Reviewed batch system GPGPU support
  - Batch treats them as 'Generic' or 'Consumable' Resources
- Will now consider a simple JDL GPGPU request
  - Want to specify allocation of *'X'* GPGPUs per node
  - On JDL we add “GPN_PN='X' via environment declaration (see example on next slide)
  - Environment for this purpose is no longer needed, and will be removed

Example JDL

```
[
Executable = "myHello.sh";
StdOutput = "std.out";
StdError = "std.err";
InputSandbox =  {"myHello.sh"};
OutputSandbox = { "std.out","std.err"} ;
VirtualOrganisation = "gputestvo";
Requirements =  ( RegExp(".*cream-pbs-gputestq",
other.GlueCEUniqueID) && Member("CUDA",
other.GlueHostApplicationSoftwareRunTimeEnvironment));
Environment = {"GPU_PN=1"};
]
```

From JDL to CREAM to LRMS

- Job Submitted from UI
  - Directly to CE,
  - Via WMS
- JDL directives can are either:
  - interpreted by WMS/CREAM,
  - or simply passed though (ignored) to CREAM
- Either way, the full JDL is stored in an internal CREAM database on CE

CREAM Job Hooks

- CREAM supports many ways to update the job and it's environment
  - Pre-submission to LRMS
  - During runtime on WN
- Use pre-submission hook to extract the JDL GPU_PN directive
  - Requires CREAM JOBID, and proxy
  - Requires glite-ce-cream-cli to recover JDL
    - Other method to get this from database directly?
  - Parse the JDL (developed ClassAD application). If GPN_PN defined and valid, then inject an LRMS GPGPU request

Results

- Method tested with:
  - CREAM Torque/MAUI
  - CREAM SLURM
- Tested direct submission to CREAM CE
- Tested submission via WMS

Service Discovery
(GLUE and Info Providers)

Glue Schema

- GLUE2 Entities
  - ExecutionEnvironment
  - ApplicationEnvironment
  - ApplicationHandles
  - App Benchmarks
- Entity attribute
  - Mandatory values
  - Also, non-mandatory values
- Entities can be extended
- Entities can publish “OtherInfo”

ExecutionEnvironment

- CPU environment
- Homogeneous set of resources
- Seems to be a natural fit to describe GPGPU environment

ApplicationEnvironment

- GLUE2 update of SoftwareEnvironment
- Currently generated from GLUE1
  - Generated data is basic
- But:
  - Can associate application benchmark
  - Can advertise Job Slots (Free) etc
    - Consider #Concurrent Software Licences
  - Advertise many OtherInfo key/value pairs

Execution vs Application Environment

- Discussion of ExecutionEnvironment at EGI-CF 2013 (Manchester)
  - Publish GPGPUs as CPUs
  - Disadvantages
    - CPU env not captured.
    - (More) Difficult to determine GRES allocation to jobs in queue and to VOs
- ApplicationEnvironment “Job/Slot” attributes can easily map to any GRES
  - Solve the problem for GPGPUs and we have a solution for other Res types thrown in for free
  - Simple Structure

ApplicationEnvironment Info Provider

- Extended glite-ce-glue2-applicationenvironment-dynamic
- Each app (and versions) can now publish
  - static data
  - dynamic data
- Capacity and utilization published Free/Max Slots and Jobs
- Arbitrary Data published as OtherInfo

CUDA ApplicationEnvironment

dn: GLUE2ApplicationEnvironmentID=CUDA_wn140.grid.cs.tcd.ie,GLUE2ResourceID=wn
140.grid.cs.tcd.ie,GLUE2ServiceID=wn140.grid.cs.tcd.ie_ComputingElement,GLUE2
….
objectClass: GLUE2ApplicationEnvironment
GLUE2ApplicationEnvironmentID: CUDA_wn140.grid.cs.tcd.ie
GLUE2ApplicationEnvironmentComputingManagerForeignKey: wn140.grid.cs.tcd.ie_Co
mputingElement_Manager
GLUE2ApplicationEnvironmentMaxSlots: 2
GLUE2ApplicationEnvironmentAppName: CUDA
GLUE2ApplicationEnvironmentFreeJobs: 2
GLUE2ApplicationEnvironmentMaxJobs: 2
…..
GLUE2EntityOtherInfo: ApplicationArea=/usr/local/cuda
GLUE2EntityOtherInfo: GPUCUDAComputeCapability=2.1
GLUE2EntityOtherInfo: GPUMainMemorySize=1024
GLUE2EntityOtherInfo: GPUMP=4
GLUE2EntityOtherInfo: GPUCoresPerMP=48
GLUE2EntityOtherInfo: GPUCores=192
GLUE2EntityOtherInfo: GPUClockSpeed=1660
GLUE2EntityOtherInfo: GPUECCSupport=false
GLUE2EntityOtherInfo: GPUVendor=Nvidia
GLUE2EntityOtherInfo: GPUModel=GTS_450
GLUE2EntityOtherInfo: GPUPerNode=2

CUDA/GPGPU AppEnv

- Dynamic GPGPU capacity and utilisation data determined from LRMS
  - Available for SLURM and Torque/MAUI
- Static Data represents GPGPU properties or Software properties
- Implicit relation between Application and GPGPU GRES

Resource Matching

Finding Available Resources

- Would ideally like to define GPGPU resource requirements in JDL file
- Our Schema is GLUE2, but WMS has limited GLUE2 support
  - WMS ISM will not consider our OtherInfo
- How do we find suitable resources?
  - WMS uses classad for matching
  - Can we do the same?

LDAP to ClassAD

- Filter ApplicationEnvironment (From TL BDII)
  - Look for CUDA environment with new attributes
  - For each match
    - Extract interesting values
    - Convert to a Condor Machine ClassAd

Example Machine ClassAd

```
MyType =    "Machine";
Machine    =    "wn140.grid.cs.tcd.ie";
GPUMainMemorySize=1024;
GPUMP=4;
GPUCoresPerMP=48;
GPUCores=192;
GPUClockSpeed=1660;
GPUECCSupport=false;
GPUVendor="Nvidia";
GPUModel="GTS_450";
GPUPerNode=2;
Requirements = true;
Rank = 1;
```

Specifying GPGPU requirements

- GPGPU JDL likes expression match against the machine ClassAds?

```
Requirements = GPUVendor=="Nvidia" && (GPUMP >= 4);
Rank = 1;
condor_test_match -machine-ads machine_props.ad -job-ads job.ad
demand
Fraction of machines matched: 1.000 (1 of 1)

---
Requirements = GPUVendor=="ATI" && (GPUMP >= 4);
Rank = 1;

Fraction of machines matched: 0.000 (0 of 1)
```

Caveats

- A GPGPU JDL used to match resources
  - Currently uses an EMI-Condor tool on UI
- Build up a list of suitable CEs from result
- This list is then used in real job JDL
  - i.e. Discovery and Submission are separate tasks
- ISM and BDII data may be different (propogation times)

Summary

- Prototype (CUDA) GPGPU support with:
  - Enumeration/Publication
  - Discovery
  - LRMS handling (Torque/MAUI & SLURM)
  - JDL spec & requirements
- Resource selection and submission are distinct tasks
- Approach works for other generic resources

Future Work

- Clearly more work needed
  - OpenCL is much more complex.
  - Expand on parameters passed to LRMS?
- Is this the right approach?
- Move GPGPU Schema to ExecutionEnvironment
  - Requires improved info providers
  - But better resource info (granularity)
- Integration with M/W stack?
  - Packaging/testing?
- Need more input from RCs and M/W experts

Questions?