GPUs with HTCondor

Throughput Computing 2023

Jason Patton

Center for High Throughput Computing, UW-Madison

GPU Basics

How to enable GPUs on EPs

1. Add metaknob `use feature: GPUs`
   - EP runs `condor_gpu_discovery` and adds all detected GPUs as custom “GPU” resources
2. Add `GPUs` to each `SLOT_TYPE` *if needed*

EP config example for single partitionable slot

```
use feature: PartitionableSlot
use feature: GPUs
```

How to enable GPUs on EPs

1. Add metaknob `use feature: GPUs`
   - EP runs `condor_gpu_discovery` and adds all detected GPUs as custom "GPU" resources
2. Add `GPUs` to each `SLOT_TYPE` *if needed*

EP config example for 4 GPUs and 4 static slots

```
use feature: GPUs

SLOT_TYPE_1 = GPUs=1,CPUs=25%,Memory=25%
NUM_SLOTS_TYPE_1 = 4
```

How to request GPUs

- `request_gpus = 1`
- (Can request more than one)
- Still need to list other resource requests
- No consideration of GPU capability, memory, etc. *on its own*

Submit file example

```
universe = container
container_image = pytorch-runtime.sif
executable = ml_training.py

request_gpus = 1
request_cpus = 1
request_memory = 32GB
request_disk = 4GB

log = ml_training.log

queue 1
```

Jobs with particular GPU requirements

- Starting with HTCondor 10, use `require_gpus`
- Common targets are **capability** and **memory**:

Submit file example

```
universe = container
container_image = pytorch-runtime.sif
executable = ml_training.py

request_gpus = 1
request_cpus = 1
request_memory = 32GB
request_disk = 4GB

require_gpus = (Capability >= 8.0) && (GlobalMemoryMb >= 16000)

log = ml_training.log

queue 1
```

The GPU job environment

- How does my job know which GPU to use?
  - HTCondor sets env var CUDA_VISIBLE_DEVICES=GPU-<uuid>
  - Your software *must* know how to use it!

Interactive job example

```
[jcpatton@submit ~]$ condor_submit -i 'request_gpus = 1' ...
Welcome to slot1_1@gpu0001.wisc.edu! ...

[jcpatton@gpu0001 ~]$ echo CUDA_VISIBLE_DEVICES=$CUDA_VISIBLE_DEVICES
CUDA_VISIBLE_DEVICES=GPU-36175dcc

[jcpatton@gpu0001 ~]$ nvidia-smi -L
GPU 0: NVIDIA A100-SXM4-80GB (UUID: GPU-1c850794-610c-fc2d-fd1c-454e76fe48c6)
GPU 1: NVIDIA A100-SXM4-80GB (UUID: GPU-36175dcc-eaea-d913-07d3-a542040dd7b9)
GPU 2: NVIDIA A100-SXM4-80GB (UUID: GPU-bd87f4bc-3691-5929-c0ae-2f65eaec5e75)
GPU 3: NVIDIA A100-SXM4-80GB (UUID: GPU-c88dc69f-5e3f-eef2-d2fb-cfb501937ead)
```

How to determine GPU usage

- GPU monitoring is automatically enabled (since 8.8.5) with the `use feature: GPUs` metaknob
- Two measurements of GPU usage:
  1. **Average usage**: Fraction of time that the GPU was being used during job execution
  2. **Peak memory usage**: Peak GPU memory usage, in MB
- GPU usage is recorded in user job logs, job ads, and in slot ads

How to determine GPU usage

Job example

```
$ tail -n 20 test.16957654.log
005 (16957654.000.000) 2023-07-07 06:38:25 Job terminated.
       (1) Normal termination (return value 0)
       <...extra output snipped...>
       Partitionable Resources :    Usage  Request Allocated Assigned
          Cpus                 :     0.00        1         1
          Disk (KB)            :    31     1048576   6063041
          Gpus (Average)       :     0.94        1         1 "GPU-bd87f4bc"
          GpusMemory (MB)      : 30573
          Memory (MB)          :  1677        2048      2048

       Job terminated of its own accord at 2023-07-07T11:38:23Z with exit-code 0.
...

$ condor_history 16957654 -af:h GPUsAverageUsage GPUsMemoryUsage
GPUsAverageUsage    GPUsMemoryUsage
0.9399650729167964         30573.0
```

How many GPUs are available?

“show only machines’ p-slots”

“show only machines w/ GPUs”

“show hostname and number of GPUs”

```
$ condor_status -compact -const 'TotalGpus > 0' -af:h Machine TotalGpus
Machine                        TotalGpus
gpu2000.chtc.wisc.edu          4
gpu2001.chtc.wisc.edu          4
gpu2003.chtc.wisc.edu          8
gpu2004.chtc.wisc.edu          8
gpu2005.chtc.wisc.edu          8
gpu2007.chtc.wisc.edu          8
gpu2008.chtc.wisc.edu          8
gpu2009.chtc.wisc.edu          8
gpu2010.chtc.wisc.edu          8
gpu2011.chtc.wisc.edu          8
```

Slots Example

Advanced GPU Topics

Before we start…

- More details in TJ Knoeller's HTCondor Week 2022 talk
- Timestamped links to YouTube → 🎥

🎥 Heterogenous GPU devices

- Handled properly in HTCondor 10 using nested ClassAds
- Results in a list of **`AvailableGPUs`** and a nested ClassAd per GPU in the slot ad (**`GPUs_GPU_<uuid>`**) containing specific properties:

(P-)Slot ClassAd Example

```
AvailableGPUs = { GPUs_GPU_c4a646d7,GPUs_GPU_6a96bd13 }
GPUs_GPU_6a96bd13 = [ DevicePciBusId = "0000:AF:00.0"; Id = "GPU-6a96bd13"; ECCEnabled =
false; DriverVersion = 12.1; DeviceName = "NVIDIA TITAN RTX"; DeviceUuid = "6a96bd13-
70bc-6494-6d62-1b77a9a7f29f"; MaxSupportedVersion = 12010;
GlobalMemoryMb = 24212; Capability = 7.5 ]
GPUs_GPU_c4a646d7 = [ DevicePciBusId = "0000:3B:00.0"; Id = "GPU-c4a646d7"; ECCEnabled =
true; DriverVersion = 12.1; DeviceName = "Tesla V100-PCIE-16GB"; DeviceUuid = "c4a646d7-
aa14-1dd1-f1b0-57288cda864d"; MaxSupportedVersion = 12010;
GlobalMemoryMb = 16151; Capability = 7.0 ]
```

🎥 Jobs with particular GPU requirements

- Starting with HTCondor 10, use `require_gpus`
- Common targets are **capability** and **memory**:

Submit file example

```
universe = container
container_image = pytorch-runtime.sif
executable = ml_training.py

request_gpus = 1
request_cpus = 1
request_memory = 32GB
request_disk = 4GB

require_gpus = (Capability >= 8.0) && (GlobalMemoryMb >= 16000)

log = ml_training.log

queue 1
```

🎥 Splitting GPUs into MIGs

- Splitting a GPU into multi-instance GPU (MIG) devices results in a **heterogeneous GPUs** situation
- The parent GPU device is omitted in slot ClassAds
- Only full UUIDs are used for the MIG devices
- Only one MIG device can be used per job (NVIDIA-imposed limitation)

Marking GPUs as offline

- Observation: Some GPUs are notoriously flaky
- List UUIDs in `OFFLINE_MACHINE_RESOURCE_GPUS` to “turn off” GPUs in the config *without killing jobs*
- Then `condor_reconfig` (no restart needed!)

Shell example

```
# condor_status -af:h DetectedGpus AvailableGpus
DetectedGpus                  AvailableGpus
GPU-c4a646d7, GPU-6a96bd13    { GPUs_GPU_c4a646d7,GPUs_GPU_6a96bd13 }
# echo 'OFFLINE_MACHINE_RESOURCE_GPUS = GPU-c4a646d7' > /etc/condor/config.d/99-offline-gpus
# condor_reconfig
Sent "Reconfig" command to local master
# condor_status -af:h DetectedGpus AvailableGpus
DetectedGpus                  AvailableGpus
GPU-c4a646d7, GPU-6a96bd13    { GPUs_GPU_6a96bd13 }
```

Prioritizing GPU jobs on EPs

- Option one - Split EP into GPU and non-GPU slots
  - Example using two partitionable slots
    1. Contains all GPU resources *and only runs GPU jobs*
    2. Contains remaining resources

EP config example

```
SLOT_TYPE_1 = GPUs=100%,CPUs=25%,Memory=50%
SLOT_TYPE_1_PARTITIONABLE = TRUE
SLOT_TYPE_1_START = $(START) && (TARGET.RequestGpus > 0)
NUM_SLOTS_TYPE_1 = 1

SLOT_TYPE_2 = CPUs=75%,Memory=50%
SLOT_TYPE_2_PARTITIONABLE = TRUE
NUM_SLOTS_TYPE_2 = 1
```

Prioritizing GPU jobs on EPs

- Option two - Set up backfill slots
  - Old way - Use “Bologna batch”
  - New, improved way - Use first-class backfill partitionable slots
- Idea: GPU jobs *may* preempt backfill (non-GPU) jobs.
  - *Maybe* allow oversubscription on some resources! (CPUs?)
- See Todd Tannenbaum’s “What’s new in HTCondor” talk

Oversubscribing GPUs

- Observation: GPUs seem to handle oversubscribing well if GPU memory isn't exhausted.
- Current option - Assign the same GPU to multiple slots
- Add the `-divide <n>` option to `GPU_DISCOVERY_EXTRA`
  - Duplicates `DetectedGpus` n times before assigning GPUs to slots.
- Caveats: No limit on GPU memory usage, no security
- Example: Allow two slots (jobs) per GPU:

EP config example

```
GPU_DISCOVERY_EXTRA = $(GPU_DISCOVERY_EXTRA) -divide 2
```

Oversubscribing GPUs

- A new *user* option soon - use job sets!
- Idea: A user should be able to (and should best know how to) fill up their own leased GPU(s) with jobs.
- See Todd Tannenbaum’s “What’s new in HTCondor” talk

Thank you!

PATh PARTNERSHIP to ADVANCE THROUGHPUT COMPUTING

This work is supported by NSF under Cooperative Agreement OAC-2030508 as part of the PATh Project. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

Any Questions?