EXTENDING THE ARC INFORMATION PROVIDER TO REPORT INFORMATION ON GPU RESOURCES

max.isacson@physics.uu.se

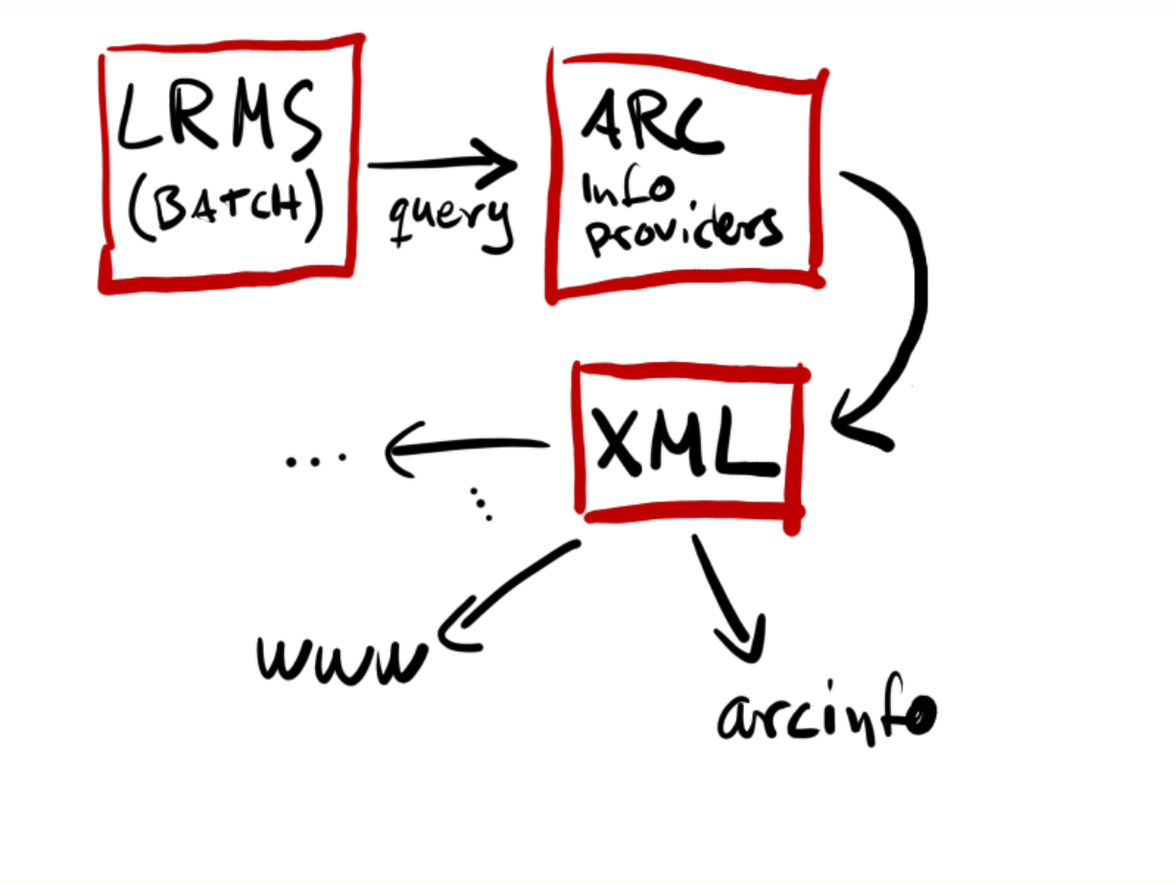
- GPU computing is very common
- Mainly used in Deep Learning
- Not used much in HEP
- Promising for tracking/reconstruction/simulation
- ARC already compatible through Runime Environments
- But GPU resources on clusters are not reported automatically

sinfo

```
$ sinfo -aho "%G"
(null)
gpu:k80ce:4,mps:no_consume:1,gpuexcl:no_consume:1
gpu:k80ce:8,mps:no_consume:1,gpuexcl:no_consume:1
gpu:v100:2,mps:no_consume:1,gpuexcl:no_consume:1
hbm:16G
hbm:0
```

- kebnekaise @ Umeå
- SLURM backend
- 80 NVIDIA Tesla K80 cards
- 20 NVIDIA Tesla V100 cards

Information flow



- SLURM provides sinfo command
- Parsed in ARC using the CEinfo.pl script
- Resulting XML used in e.g. arcinfo
- Modifications done in ARC6 RC1

SLURMmod.pm

```
our(/*...*/, @sinfo_gresinfo);

#...

sub cluster_info() {
# ...
  $lrms_cluster->{gres} = [@sinfo_gresinfo];
# ...
}

sub slurm_get_data {) {
# ...
  @sinfo_gresinfo = slurm_read_gresinfo();
}

# ...

sub slurm_read_gresinfo($){
  my @sinfo_gresinfo;
  my $gresinfo;
  open (SCPIPE,"$path/sinfo -a -h -o \"gresinfo=%G\"|");
  while(<SCPIPE>){
  my $string = $_;
    if ($string !~ m/\(null\)/) {
    $gresinfo = get_variable("gresinfo",$string);
      push(@sinfo_gresinfo, $gresinfo);
      }
  }
  close(SCPIPE);

  return @sinfo_gresinfo;
}
```

- Provides the SLURM interface
- Execute sinfo and parse the output
- Store the output in the $lrms_cluster table
- Can be passed around to other modules as needed

LRMSInfo.pm

```
my $lrms_info_schema = {
  'cluster' => {
  # ...
    'gres' => ''
  },
  # ...
};
```

ARC1ClusterInfo.pm

```
sub collect($) {
  # ...
  my $getComputingService = sub {
    # ...
    my $getComputingManager = sub {
      # ...
      $cmgr->{GeneralResources}{Resource} = $cluster_info->{gres};
      # ...
    };
    # ...
  };
  # ...
}
```

- Also add this to the Computing Manager table $cmgr

GLUE2xmlPrinter.pm

```
sub ComputingManager {
  Element(@_, 'ComputingManager', 'Manager', sub {
    # ...
    $self->begin('GeneralResources');
    $self->GeneralResources($data->{GeneralResources});
    $self->end('GeneralResources');
    # ...
  });
}

# ...

sub GeneralResources {
  my ($self, $data) = @_;
  $self->properties($data, 'Resource');
}
```

- Print the $cmgr table into XML
- Create a new GeneralResource section to the XML tree
- Print each entry as a Resource field

CEinfo.pl XML output

```
<inforoot>
  <domains>
    <admindomain>
      <!-- ... -->
      <services>
        <computingservice>
          <!-- ... -->
          <computingmanager>
            <!-- ... -->
            <generalresources>
              <resource>gpu:k80ce:4,mps:no_consume:1,gpuexcl:no_consume:1</resource>
              <resource>gpu:k80ce:8,mps:no_consume:1,gpuexcl:no_consume:1</resource>
              <resource>gpu:v100:2,mps:no_consume:1,gpuexcl:no_consume:1</resource>
              <resource>hbm:16G</resource>
              <resource>hbm:0</resource>
            </generalresources>
            <!-- ... -->
          </computingmanager>
        <!-- ... -->
        </computingservice>
      </services>
    </admindomain>
  </domains>
</inforoot>
```

GLUE2.cpp

```
namespace Arc {
  /* ... */
  void GLUE2::ParseExecutionTargets(XMLNode glue2tree,
      std::list<ComputingServiceType>& targets) {
    /* ... */
    for (; GLUEService; ++GLUEService) {
      /* ... */
      for (XMLNode xComputingManager = GLUEService["ComputingManager"];
          (bool)xComputingManager; ++xComputingManager) {
        /* ... */
        if (xComputingManager["GeneralResources"]) {
          for (XMLNode n = xComputingManager["GeneralResources"]["Resource"];
              n; ++n) {
            ComputingManager->GeneralResources.push_back((std::string)n);
          }
        }
        /* ... */
      }
      /* ... */
    }
    /* ... */
  }
  /* ... */
}
```

- XML tree parsed by the GLUE2 class
- Simply add the right information to the ComputingManager

ExecutionTarget.h

```
namespace Arc {
/* ... */
  class ComputingManagerAttributes {
  public:
    /* ... */
    std::list<std::string> GeneralResources;
    /* ... */
  };
  /* ... */
}
```

ExecutionTarget.cpp

```
namespace Arc {
  /* ... */
  std::ostream operator<<(std::ostream& out,
      const ComputingManagerAttributes cm) {
    /* ... */
    if (!cm.GeneralResources.empty()) {
      out << IString("General resources:") << std::endl;
      for (std::list<std::string>::const_iterator it =
          cm.GeneralResources.begin();
          it != cm.GeneralResources.end(); ++it) {
        out << "   " << *it << std::endl;
      }
    }
    /* ... */
  }
  /* ... */
}
```

- Make arcinfo aware of the new information
- Add a new string list to ExecutionTarget
- Modify the stream operator

arcinfo

```
$ arcinfo
Computing service:
  # ...
  Batch System Information:
    # ...
    General resources:
      gpu:k80ce:4,mps:no_consume:1,gpuexcl:no_consume:1
      gpu:k80ce:8,mps:no_consume:1,gpuexcl:no_consume:1
      gpu:v100:2,mps:no_consume:1,gpuexcl:no_consume:1
      hbm:16G
      hbm:0

    # ...
  # ...
```

- Possible to query GPU resources with arcinfo
- Could push this information elsewhere, e.g. the Grid Monitor
- Could also be used for job brokarage

Minimal runtime environment

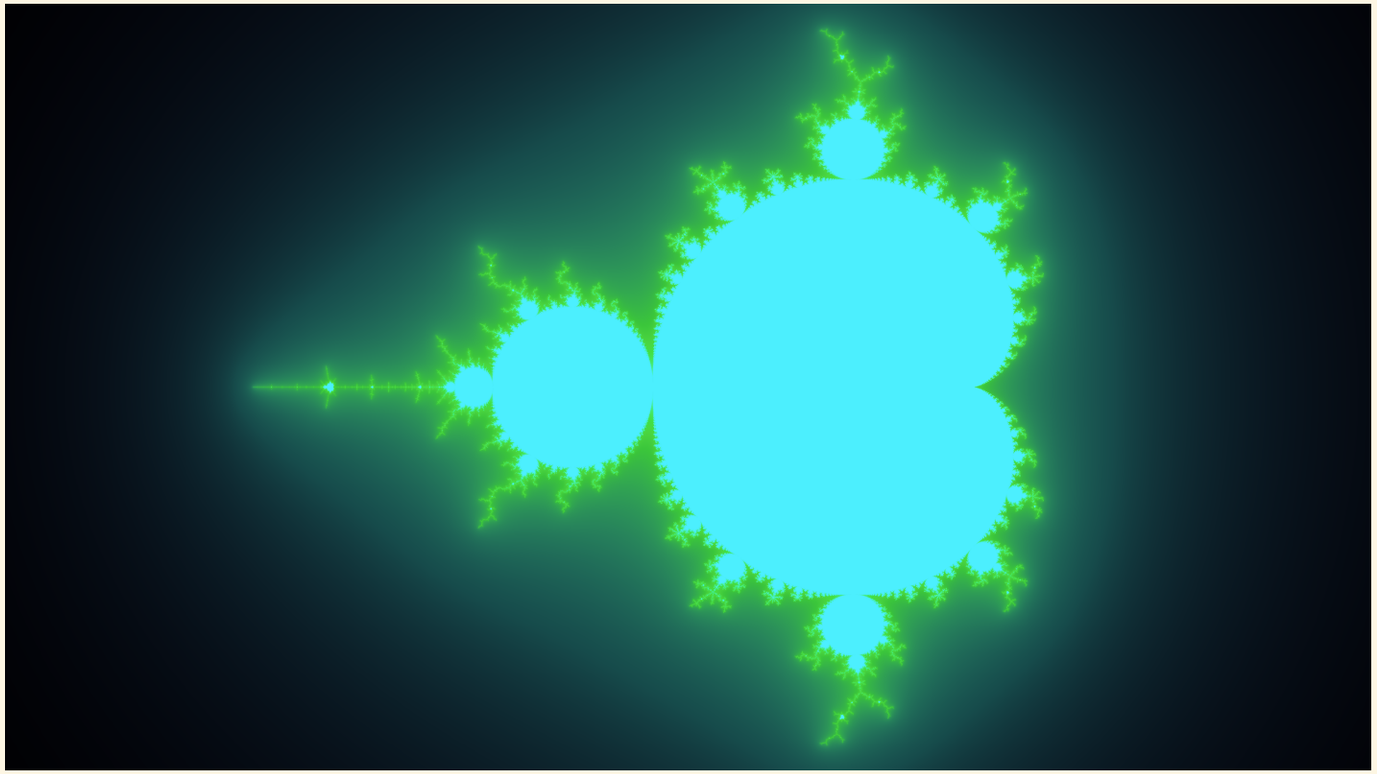
```
#!/bin/bash

case "$1" in
    0) # called during creation of batch script on frontend
        export joboption_rsl_project=SNIC20XX-Y-ZZ
        export joboption_nodeproperty_0="--gres=gpu:k80:1"
        ;;
    1) # called before execution of the main executable on the computing node
        module load GCC
        module load CUDA
        ;;
    2) # called after execution of the main executable on the computing node
        ;;
    *) # error
        return 1
        ;;
esac
```

Example: Track fit xrsl-job

```
&
(jobName="GPUtracking")
(executable="cuda_fitter")
(arguments="muon.txt")
(runTimeEnvironment="ENV/KGPU")
(inputFiles=("cuda_fitter" "") ("muon.txt" ""))
(outputFiles=("/" ""))
(wallTime="30")
(stdout="std.out")
(stderr="std.err")
```

Output

```
input file: muon.txt
1101 events are read from muon.txt
Total 3034 tracks for fitting
Total test time: 700.719971 msecs.
GPU time : 5.756000 msecs
CPU time : 284.036011 msecs
GPU tracks=3000
CPU tracks=3000
GPU time/track = 0.001919
CPU time/track = 0.094679
```



- 4K Mandelbrot fractal
- Takes about 120 ms to render on a single K80 card (about 5K GPU cores)
- Most of it (~118 ms) is writing the bitmap to disk