Measuring GPU Utilization on the Cray XK7

Jim Rogers
Director of Operations
National Center for Computational Sciences
Oak Ridge National Laboratory

May 9, 2013
Motivation: Transition from the Cray XT5 *Jaguar* to the XK7 *Titan*
Heterogeneous Compute Nodes: CPU+GPU

- **Implementation**
  - Phase 1 [Complete 02/2012]
    - Cabinet Upgrades
    - XT5->XK Compute Blades
    - AMD 6274 Opteron (16-core)
    - 32GB/node DDR3 DRAM upgrade
    - Gemini Interconnect
  - Phase 2 [Complete 05/2013]
    - Power System Upgrades
    - NVIDIA Kepler K20x

- **Schedule**
  - 01/2009. CD-0 (Mission Need)
  - 01/2010. CD-1 (Alternative Selection / Cost Range)
  - 02/2011. CD-3a (Performance Baseline / Permission to Issue Solicitation)
  - 11/2011. CD-2/3b (Permission to Issue Subcontract)
  - 12/2013. CD-4 (Project Closeout)
Motivation: OLCF’s Long Term Commitment to Hybrid Computing

• Titan is expected to remain in production through 2017

• Given a 60-month production life, it is even more important that we maximize the productivity of this hybrid architecture

• DOE Sponsor interest
  – How is OLCF effectively managing this resource for its user community?

• We need the complementary tools to understand GPU use:
  – How effectively are the users able to use the hybrid architecture?
  – How many delivered node-hours are GPU-enabled?
  – How can/should we adapt operational policies
    • allocation,
    • scheduling

• The OLCF-3 Titan will provide the computational capability to accomplish breakthrough science for DOE, industry, and the nation
  – Transforming the nation’s energy system and secure U.S. leadership in clean energy technologies.
  – Maintain a vibrant U.S. effort in science and engineering

• Qualifying Risk
  – Significant investment:
    • Phase 1 subcontract ($60M)
    • Phase 2 subcontract ($40M)
  – Significant Architectural Change
    • Hybrid CPU+GPU node
  – Significant Software Challenges
    • OPENACC
    • CUDA
Interim GPU Utilization Measurement on Titan using ALTD

- ORNL examines, on a per-job basis, when jobs use accelerator-equipped nodes.
- Leverages the ORNL-developed Automatic Library Tracking Database (ALTD)
  - At link time, a list of libraries linked against is stored in a database
  - When the resulting program is executed via aprun, a new ALTD record is written that contains the specific executable, to be run, the batch job id, and other info
- Batch jobs are compared against the ALTD to see if they were linked against an accelerator-specific library
  - libacc*, libOpenCL*, libmagma*, libhmpp*, libcuda*, libcupti*, libcublas*
  - Jobs whose executables are linked against one of the above are deemed to have used the accelerator
- Outliers
  - Job run outside of the batch system
    - ALTD knows about them, but we can't tie them to usage because there's no job record
  - ALTD is enabled by default, but if it's disabled we won't capture link/run info
    - We don't advertise how to disable it
    - Happens with debug sessions-debuggers can't handle the wrapped aprun
  - Codes compiled with nvcc aren't captured due to how it links
  - Codes using Open MPI (do not use aprun)

Making sense of an link statement

```bash
% lsms /usr/lib//lib64/crt1.o /usr/lib//lib64/crti.o
/opt/gcc/4.7.0/snos/lib/gcc/x86_64-suse-linux/4.7.0/crtbegin.o
libLSMs.aSystemParameters.o libLSMs.aread_input.o libLSMs.aPotentialIO.o
libLSMs.abuildLZandCommLists.o libLSMs.aenergyContourIntegration.o
libLSMs.asolveSingleScat°erers.o libLSMs.acalculateDensities.o
libLSMs.acalculateChemPot.o
/lustre/00/widow0/scratch/lsms-trunk/lua/lib/liblua.a
/lustre/00/widow0/scratch/lsms-trunk/lib/libLSMSLua.a
/lustre/00/widow0/scratch/lsms-trunk/lib/libCommunica.on.a
... -l cublas /opt/nvidia/cudatoolkit/5.0.28.101/lib64/libcublas.so -lcupi
/opt/nvidia/cudatoolkit/5.0.28.101/extras/CUPTI/lib64/libcudart.so -lcudart
/opt/nvidia/cudatoolkit/5.0.28.101/lib64/libcudart.so -lcuda
/opt/cray/nvidia/default/lib64/libcuda.so
/opt/cray/atp/1.4.4/lib/libAtpSigHComData.a -lAtpSigHandler
/opt/cray/atp/1.4.4/lib/libAtpSigHandler.so -lgfortran
/opt/gcc/4.7.0/snos/lib/gcc/x86_64-suse-linux/4.7.0/..../lib64/libgffortran.so -lhdff5_hl_cpp-gnu
/opt/cray/hdf5/1.8.8/gnu/47/lib/libhdff5_hl_cpp-gnu.so -lhdff5_hl-gnu
/opt/cray/hdf5/1.8.8/gnu/47/lib/libhdff5_hl_cpp_gnu.so -lhdff5_cpp-gnu
/opt/cray/hdf5/1.8.8/gnu/47/lib/libhdff5_cpp_gnu.so -lhdff5-gnu
/opt/cray/hdf5/1.8.8/gnu/47/lib/libhdff5-gnu.so -lz
/usr/lib//lib64/libbz.so -lsciçcpp-gnu
/opt/xt-libscl/11.1.00/gnu/47/interlagos/lib/libsci_cpp-gnu.so -lsci-gnu_mp
/opt/xt-libscl/11.1.00/gnu/47/interlagos/lib/libsci-gnu_mp.so -lgfortran
/opt/gcc/4.7.0/snos/lib/gcc/x86_64-suse-linux/4.7.0/..../lib64/libgffortran.so -lmpichcxx-gnu_47
...
```

Jim Rogers | CUG’13
Summary Results of the ALTD Method

- Sample size reflects two partitions of Titan—one in production, and one reserved for internal software testing and development.

- Outliers (Unknowns) Continue to Dominate
  - The intermediate method using ALTD is not wholly effective, with ~34% of the delivered hours uncategorized by this method.

Uncaptured Edge Cases
- Nvcc
- Open MPI
  - Recently identified a workaround
- Anything outside of aprun
- Dedicated time
- Others? (still digging)
Titan: Cray XK7 System – Addressing the Scale of the GPU Utilization Requirement

System:
- 200 Cabinets
- 18,688 Nodes
- 27 PF
- 710 TB

Cabinet:
- 24 Blades
- 96 Nodes
- 139 TF
- 3.6 TB

Board:
- 4 Compute Nodes
- 5.8 TF
- 152 GB

A Cray XK7 Heterogeneous Compute Node:
- 1.45 TF
- 32 GB DDR3
- 16 GB GDDR5

PCIe Gen2

HT3

Jim Rogers | CUG’13
NVIDIA Kepler K20x Block Diagram

- The K20X is a single FRU based on the NVIDIA GK110 GPU

Images courtesy NVIDIA Corporation

<table>
<thead>
<tr>
<th>Specifications</th>
<th>Tesla K20X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chip</td>
<td>GK110</td>
</tr>
<tr>
<td>Package Size</td>
<td>GPU 45 mm × 45 mm 2397-pin S-FCBGA</td>
</tr>
<tr>
<td>Processor Clock</td>
<td>732 MHz</td>
</tr>
<tr>
<td>Memory Clock</td>
<td>2.6GHz</td>
</tr>
<tr>
<td>Memory Size</td>
<td>6GB</td>
</tr>
<tr>
<td>Memory I/O</td>
<td>384-bit GDDR5</td>
</tr>
<tr>
<td>Board Power</td>
<td>235W (225W power cap on Titan)</td>
</tr>
<tr>
<td>Idle Power</td>
<td>~25W</td>
</tr>
<tr>
<td>Thermal Cooling Solution</td>
<td>Passive Heat Sink</td>
</tr>
</tbody>
</table>
Kepler GK110 Full Chip Block Diagram
The NVIDIA GK110 in Titan

• (From previous slide), the NVIDIA GK110 product family contains 14 or 15 SMXs. The Titan product has 14.

• For each SMX: 192 single-precision CUDA Cores, 64 double-precision units, 32 special function units (SFU), and 32 load/store units (LD/ST).

• For each SMX: four warp schedulers and eight instruction dispatch units, allowing four 32-thread warps to be issued and executed concurrently.
The NVIDIA GK110 Deep Dive

For more information on Kepler that includes all of the hardware deep dive, including the significant feature improvements from Fermi to Kepler: Dynamic Parallelism, Hyper-Q, the Grid Management Unit, and GPU Direct).

Reference:

NVIDIA’s Next Generation CUDA™ Compute Architecture: Kepler™ GK110

The System-Level Interface to the GK110
- the NVIDIA Management Library (NVML)

• NVML is a C-based API for monitoring and managing various states of the NVIDIA GPU devices. The runtime version of NVML ships with the NVIDIA display driver, and the SDK provides the appropriate header, stub libraries and sample applications. Each new version of NVML is backwards compatible and is intended to be a platform for building 3rd party applications.

• States that may be queried include:
  - ECC error counts: Both correctable single bit and detectable double bit errors are reported. Error counts are provided for both the current boot cycle and for the lifetime of the GPU.
  - GPU utilization: Current utilization rates are reported for both the compute resources of the GPU and the memory interface.
  - Active compute process: The list of active processes running on the GPU is reported, along with the corresponding process name/id and allocated GPU memory.
  - Clocks and PState: Max and current clock rates are reported for several important clock domains, as well as the current GPU performance state.
  - Temperature and fan speed: The current core GPU temperature is reported, along with fan speeds for non-passive products.
  - Power management: For supported products, the current board power draw and power limits are reported.
  - Identification: Various dynamic and static information is reported, including board serial numbers, PCI device ids, VBIOS/Inforom version numbers and product names.

• nvidia-smi is an existing application that uses the nvml API. An example follows-

nvml-smi Output (truncated) from a single Titan Kepler GPU

==============NVSMI LOG==============

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Mon Mar 18</th>
</tr>
</thead>
<tbody>
<tr>
<td>Driver Version</td>
<td>304.47.13</td>
</tr>
<tr>
<td>Attached GPUs</td>
<td>1</td>
</tr>
<tr>
<td>Product Name</td>
<td>Tesla K20X</td>
</tr>
<tr>
<td>Display Mode</td>
<td>Disabled</td>
</tr>
<tr>
<td>Persistence Mode</td>
<td>Enabled</td>
</tr>
<tr>
<td>Performance State</td>
<td>P8</td>
</tr>
<tr>
<td>Clocks Throttle Reasons</td>
<td>Idle</td>
</tr>
<tr>
<td></td>
<td>User Defined Clocks: Active</td>
</tr>
<tr>
<td></td>
<td>SW Power Cap: Not Active</td>
</tr>
<tr>
<td></td>
<td>HW Slowdown: Not Active</td>
</tr>
<tr>
<td></td>
<td>Unknown: Not Active</td>
</tr>
<tr>
<td>Memory Usage</td>
<td>5759 MB</td>
</tr>
<tr>
<td></td>
<td>37 MB</td>
</tr>
<tr>
<td></td>
<td>5722 MB</td>
</tr>
<tr>
<td>Compute Mode</td>
<td>Exclusive Process</td>
</tr>
<tr>
<td>Gpu</td>
<td>0 %</td>
</tr>
<tr>
<td>Memory</td>
<td>0 %</td>
</tr>
<tr>
<td>Ecc</td>
<td>Enabled</td>
</tr>
<tr>
<td>Mode</td>
<td>Enable</td>
</tr>
<tr>
<td>Current</td>
<td></td>
</tr>
<tr>
<td>Pending</td>
<td></td>
</tr>
<tr>
<td>Power Management</td>
<td>Supported</td>
</tr>
<tr>
<td>Power Draw</td>
<td>18.08 W</td>
</tr>
<tr>
<td>Max Power Limit</td>
<td>300.00 W</td>
</tr>
<tr>
<td>Kepler Clocks</td>
<td>324 MHz</td>
</tr>
<tr>
<td></td>
<td>324 MHz</td>
</tr>
<tr>
<td></td>
<td>324 MHz</td>
</tr>
<tr>
<td>Graphics</td>
<td>732 MHz</td>
</tr>
<tr>
<td>Memory</td>
<td>2600 MHz</td>
</tr>
<tr>
<td>Max Clocks</td>
<td>84 MHz</td>
</tr>
<tr>
<td></td>
<td>84 MHz</td>
</tr>
<tr>
<td></td>
<td>2600 MHz</td>
</tr>
<tr>
<td>Compute Processes</td>
<td>None</td>
</tr>
</tbody>
</table>

The version for the utilization upgrades will be from the R319 Branch.

Kepler has either a p-state of 0 (busy) or 8 (idle).

6 GB GDDR5

Instantaneous GPU utilization. This is a point-in-time sample.
nvml-smi Output (truncated) from a single Titan Kepler GPU

Kepler detects idle states and throttles clocks and power consumption

The XK7 caps the Kepler power budget at 225W

Throttled clock rate

Configured/normal clock rate

ECC Errors
Temperature
GPU: 25 C

Power Readings
Power Management: Supported
Power Draw: 18.08 W
Power Limit: 225.00 W
Default Power Limit: 225.00 W
Min Power Limit: 55.00 W
Max Power Limit: 300.00 W

Clocks
Graphics: 324 MHz
SM: 324 MHz
Memory: 324 MHz

Applications Clocks
Graphics: 732 MHz
Memory: 2600 MHz

Max Clocks
Graphics: 784 MHz
SM: 784 MHz
Memory: 2600 MHz

Compute Processes: None

Kepler detects idle states and throttles clocks and power consumption.

The XK7 caps the Kepler power budget at 225W.

Configured/normal clock rate.
Requirements Gathering as a Community

• In early 2012, we identified a critical need to understand “GPU utilization” in a broader context.

• At a CUG 2012 BOF, the XK community initiated a discussion about what types of data would be appropriate or helpful. That initial list described items including:
  – Number of GPU contexts opened
    • how many times is cuInit() called?
  – Number of GPU kernels run
    • This self-destructed, since dynamic parallelism added the capability to spawn kernels on the GPU, from the GPU, and eliminated the ability to map this from the CPU-side.
  – Amount of time (GPU-seconds) accrued
    • the amount of time that a job has a GPU context open. This is not the amount of time that a kernel is running on the GPU, and is an indicator of one of the design points of the implemented solution
  – High water mark of memory used – how much of the 6GB of GDDR5 were allocated?

• By Q4 2012, we reached a decision point where we could proceed, defining both method and definition for Measuring NVIDIA GPU Utilization
NVIDIA’s Response to the Need for Utilization Data
- Driver Changes and API Extensions

• ORNL, NVIDIA, and Cray discussed mechanisms for making information on GPU utilization available from any single Kepler, and mechanisms from the Cray perspective on aggregating all of the math so that the utilization of the GPUs could be defined and reported on a per-job basis. The underlying mechanism is an NVIDIA API that includes the function:

\[
\text{nvmlDeviceGetAccountingStats (nvmlDevice_t device, unsigned int pid, nvmlAccountingStats_t *stats)}
\]

• This extension is available for Tesla ™ and Quadro ® products from the Kepler family. The function queries a process’s accounting statistics, capturing GPU utilization across the lifetime of a process. Accounting stats can be queried during life time of the process and after its termination. Accounting stats are kept in a circular buffer; newly created processes overwrite information about old processes.

\[
\text{nvmlAccountingStats_t struct}
\]

\[
\begin{align*}
\text{unsigned int gpuUtilization} & & \text{Percent of time over the process’s lifetime during which one or more kernels was executing on the GPU. Utilization stats just like that returned by nvmlDeviceGetUtilizationRates but for the life time (to date) of a process (not just the last second).} \\
\text{unsigned int memoryUtilization} & & \text{Percent of time over the process’s lifetime during which global (device) memory was being read or written.} \\
\text{unsigned long long maxMemoryUsage} & & \text{Maximum total memory in bytes that was ever allocated by the process.} \\
\text{unsigned int startTimeStamp} & & \text{Timestamp of the moment process allocated GPU (TODO) define how to map timestamp to system time.} \\
\text{unsigned int endTimeStamp} & & \text{Timestamp of the process termination.}
\end{align*}
\]
Defining GPU Utilization for Kepler

• What (exactly) is meant by GPU Utilization?
  – GPU utilization is an assessment of whether work is scheduled.

  *Is work active on ANY of the 14 SMs?*

• This assessment is made O(1000) times per second already. By taking advantage of an existing mechanism, there is no additional overhead required.

• These transient values are not retained.

• At the end of a one second interval, these samples are aggregated and averaged, and the 1-second value is saved as a data point.

• At the end of an application run, each of the 1-second values are aggregated and averaged, and the n-second value is saved in to the 128-deep pid/GPU_utilization buffer.
Kepler GPU – Sampling Per Second

- During a single second, the GPU samples whether any of the 14 SMs are in use, and aggregates on the order of 1000 samples to determine the GPU utilization during that one second.
- Each of those samples is a binary result.
- If there is any instruction executing on any SM, a value of 1 is returned.
- At the end of the time interval (one second), the arithmetic mean for that interval is returned.
- This per-second value that is then averaged over the term of the entire run and returned by the call to (API routine).

**per-Kepler, single-second GPU Utilization**
(binary state, O(1000) samples per second)
Kepler GPU – Sampling Across the Lifetime of an Application

- For each Kepler, each of the samples that are available each second of the life of an application are tracked and aggregated. In addition, the Start and End Times are known, providing the capability to calculate the “rolling average” GPU utilization at any time that the routine `nvmlDeviceGetAccountingStats` is called.
- By calling this routine at the end of an application, the single-Kepler GPU utilization, expressed as a percentage between 0-100% is available.
- By then aggregating all of the single-Kepler GPU utilization results using the pid as the filter, Cray can report on the average GPU utilization across all Keplers associated with an application.

**per-Kepler, per-second GPU Utilization**

![Graph showing per-Kepler, per-second GPU Utilization](image-url)
GPU Reporting and Accounting Update

• Current Status
  • Received GPU Accounting API in early January
  • Received Alpha driver from the CUDA 5.5/R319 Branch on February 15
  • Initial evaluation of the API is underway
  • Working open issues with NVIDIA

• NVIDIA is delivering production GPU-utilization in CUDA 5.5
  • R319 Branch Release candidate planned for early May
  • Target production release planned mid-July.
  • Cray will deliver CUDA 5.5 with CLE 4.2 UP02 on XK

• Resource Utilization Recording (RUR)
  • Cray moving to a more extensible accounting infrastructure
  • RUR targeted for CLE 4.2 UP02 and aligned with CUDA 5.5 release
  • RUR CUG paper dovetails with this presentation
## XK CLE 4.2 and CUDA 5.5 Release Milestones

<table>
<thead>
<tr>
<th>Qtr 4, 2012</th>
<th>Qtr 1, 2013</th>
<th>Qtr 2, 2013</th>
<th>Qtr 3, 2013</th>
<th>Qtr 4, 2013</th>
</tr>
</thead>
<tbody>
<tr>
<td>September</td>
<td>October</td>
<td>November</td>
<td>December</td>
<td>January</td>
</tr>
</tbody>
</table>

- **Glacier SW Releases 2013**
  - CLE 4.2 UP00 Release with CUDA 5.0 (4/4)
  - CLE 4.2 UP01 Release with CUDA 5.0 (7/11)
  - CLE 4.2 UP02 Release with CUDA 5.5 (10/10)
  - CLE 4.2 UP02 Content Defined (6/20)
  - CLE 4.2 UP02 Feature Code Cut-off (7/18)
  - CLE 4.2 UP02 3rd Party Code Cut-off (8/29)
  - CLE 4.2 UP02 Final Code Freeze (last CCB) (9/19)
  - CLE 4.2 UP02 Release to Manufacturing (RTM) (10/10)

- **CUDA 5.5**
  - RFE: GPU Utilization - Draft API Spec (1/2)
  - RFE: GPU Utilization - Alpha (CUDA 5.0) (2/15)
  - RFE: GPU Utilization - Beta (CUDA 5.5) (4/15)
  - CUDA 5.5 Release Candidate (RC) (5/6)
  - CUDA 5.5 Production Goal (6/17)
  - CUDA 5.5 Production Target (PR) (7/15)

*PRELIMINARY, Cray Confidential and Proprietary Information*
Engineering Takeaways

• An elegant solution from an engineering perspective. The changes to the device driver, NVML libraries and API:
  – Take advantage of existing transient data
  – Minimize performance impact to the GPU (near zero impact)
  – Still provides both the pid list and util numbers on a per-Kepler perspective to the recipient

• Energy Efficiency
  – At p-state 0, the chip is operating at 732MHz and consuming somewhere between 160 and 225W.
  – At p-state 8, the chip is idle, and is operating at 324MHz, consuming as little as 20W or less.

• Saving State Information
  – The pid/statistics buffer is 128 deep, per Kepler, saving the pid and GPU utilization information in a round-robin fashion.

• Potential Future Work -
  – Currently, in a single second, there are ~1000 samples taken each second, per GK110. The assumption is that in each sample, the SMs are busy, or they are not.
  – Within the existing hardware, there is a capability to assess during that cycle how many SMs (0-14) are available, i.e. the granularity of the reporting could change from 0/1 (0%-100%) to 0-1 (0%-100%, step 7.14%).
Questions?

Jim Rogers
jrogers@ornl.gov

The activities described herein were performed using resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC0500OR22725.