NOVEL GPU FEATURES: PERFORMANCE AND PRODUCTIVITY

Peter Messmer
p messmer@nvidia.com
COMPUTATIONAL CHALLENGES IN HEP

Low-Level Trigger

High-Level Trigger

Monte Carlo Analysis

Lattice QCD
COMPUTATIONAL CHALLENGES IN HEP

**Low-Level Trigger**
- Connectivity
- Latency control
- Power limited

**High-Level Trigger**
- Data volume
- Computer vision
- Unsupervised learning

**Monte Carlo Analysis**
- Portability
- Legacy codes
- Limited parallelism
- Geometry processing

**Lattice QCD**
- Connectivity
- Low latency
- Bandwidth, bandwidth, bandwidth, ...
COMPUTATIONAL CHALLENGES IN HEP

Low-Level Trigger
- Connectivity
- Latency control
- Power limited

High-Level Trigger
- Data volume
- Computer vision
- Unsupervised learning

Monte Carlo Analysis
- Portability
- Legacy codes
- Limited parallelism
- Geometry processing

Lattice QCD
- Connectivity
- Low latency
- Bandwidth, bandwidth, bandwidth,..
KEPLER ENABLES NVIDIA GPUDIRECT™ RDMA

http://docs.nvidia.com/cuda/gpudirect-rdma
TESLA K40
WORLD’S FASTEST ACCELERATOR

**FASTER**
1.4 TF | 2880 Cores | 288 GB/s
AMBER Benchmark

**LARGER**
2x Memory Enables More Apps

**SMARTER**
Unlock Extra Performance Using Power Headroom

**GPU Boost**

**AMBER Benchmark: SPFP-Nucleosome**
CPU: Dual E5-2687W @ 3.10GHz, 64GB System Memory, CentOS 6.2, GPU systems: Single Tesla K20X or Single Tesla K40
GPU BOOST ON TESLA K40

*Convert Power Headroom to Higher Performance*

- **Workload #1**: Worst case Reference App
  - **Boost Clock #2**: 875Mhz
  - **Base Clock**: 235W
- **Workload #2**: E.g. AMBER
  - **Boost Clock #1**: 810Mhz
  - **Base Clock**: 235W
- **Workload #3**: E.g. ANSYS Fluent
  - **Base Clock**: 745Mhz
# WORKLOAD BEHAVIOR WITH GPU BOOST

## Non-Tesla

- **Automatic clock switching**

## Tesla K40

- **Deterministic Clocks**

<table>
<thead>
<tr>
<th>Default</th>
<th>Boost</th>
<th>Base</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Preset Options</strong></td>
<td>Lock to base clock</td>
<td>3 Levels: Base, Boost1 or Boost2</td>
</tr>
<tr>
<td><strong>Boost Interface</strong></td>
<td>Control Panel</td>
<td>NV-SMI, NVML</td>
</tr>
<tr>
<td><strong>Target duration for boost clocks</strong></td>
<td>~50% of run-time</td>
<td>100% of workload run time</td>
</tr>
</tbody>
</table>

*Must-have for HPC workload*
TEGRA K1
IMPOSSIBLY ADVANCED

NVIDIA Kepler Architecture

4-Plus-1 Quad-Core A15

192 NVIDIA CUDA Cores

Compute Capability 3.2

326 GFLOPS

5 Watts
JETSON TK1
THE WORLD’S 1st EMBEDDED SUPERCOMPUTER

Development Platform for Embedded Computer Vision, Robotics, Medical

192 Cores · 326 GFLOPS
CUDA Enabled

Available Now
BATCHED 1D FFT PERFORMANCE ON JETSON

- Low power processor on sensor
- Host/Dev Bandwidth: 6 GB/s
- Dev/Dev FFT up to 780 Msamples/s
- Streaming FFT off host: 110 Msamples/s sustained
- About 1/13th of K20 performance
COMPUTATIONAL CHALLENGES IN HEP

Low-Level Trigger:
- Connectivity
- Latency control
- Power limited

High-Level Trigger:
- Data volume
- Computer vision
- Unsupervised learning

Monte Carlo Analysis:
- Portability
- Legacy codes
- Limited parallelism
- Geometry processing

Lattice QCD:
- Connectivity
- Low latency
- Bandwidth, bandwidth, bandwidth,
**TESLA K40**
WORLD’S FASTEST ACCELERATOR

**FASTER**
1.4 TF | 2880 Cores | 288 GB/s

**LARGER**
2x Memory Enables More Apps

**SMARTER**
Unlock Extra Performance Using Power Headroom

**AMBER Benchmark: SPFP-Nucleosome**
CPU: Dual E5-2687W @ 3.10GHz, 64GB System Memory, CentOS 6.2, GPU systems: Single Tesla K20X or Single Tesla K40

**GPU Boost**

- Fluid Dynamics
- Rendering
- Seismic Analysis

6GB

12GB
INTRODUCING NVLINK AND STACKED MEMORY

**NVLINK**
- GPU high speed interconnect
- 80-200 GB/s
- Planned support for POWER CPUs

**Stacked Memory**
- 4x Higher Bandwidth (~1 TB/s)
- 3x Larger Capacity
- 4x More Energy Efficient per bit
NVLink enables data transfer at speed of CPU memory.
UNIFIED MEMORY
DRAMATICALLY LOWER DEVELOPER EFFORT

Developer View Today

System Memory

GPU Memory

Developer View With Unified Memory

Unified Memory
void sortfile(FILE *fp, int N) {
    char *data;
    data = (char *)malloc(N);
    fread(data, 1, N, fp);
    qsort(data, N, 1, compare);
    use_data(data);
    free(data);
}

void sortfile(FILE *fp, int N) {
    char *data;
    cudaMallocManaged(&data, N);
    fread(data, 1, N, fp);
    qsort<<<...>>>(data, N, 1, compare);
    cudaDeviceSynchronize();
    use_data(data);
    cudaFree(data);
}
SIMPLER MEMORY MODEL: ELIMINATE DEEP COPIES

```c
struct dataElem {
    int prop1;
    int prop2;
    char *text;
};
```

CPU Memory
- `dataElem`
  - `prop1`
  - `prop2`
  - `text` → “Hello world”

GPU Memory
SIMPLER MEMORY MODEL: ELIMINATE DEEP COPIES

struct dataElem {
    int prop1;
    int prop2;
    char *text;
};
void launch(dataElem *elem) {
    dataElem *g_elem;
    char *g_text;

    int textlen = strlen(elem->text);

    // Allocate storage for struct and text
    cudaMalloc(&g_elem, sizeof(dataElem));
    cudaMalloc(&g_text, textlen);

    // Copy up each piece separately, including
    // new “text” pointer value
    cudaMemcpy(g_elem, elem, sizeof(dataElem));
    cudaMemcpy(g_text, elem->text, textlen);
    cudaMemcpy((g_elem->text), &g_text, sizeof(g_text));

    // Finally we can launch our kernel, but
    // CPU & GPU use different copies of “elem”
    kernel<<< ... >>>(g_elem);
}
SIMPLER MEMORY MODEL: ELIMINATE DEEP COPIES

```c
void launch(dataElem *elem) {
    kernel<<< ... >>>(elem);
}
```
void sortfile(FILE *in, FILE *out, int N) {
    char *data = (char *)malloc(N);
    fread(data, 1, N, in);
    sort(data, N);
    fwrite(data, 1, N, out);
    free(data);
}

Call Sort on CPU

void sortfile(FILE *in, FILE *out, int N) {
    char *data = (char *)cudaMallocManaged(N);
    fread(data, 1, N, in);
    parallel_sort<<< ... >>>(data, N);
    fwrite(data, 1, N, out);
    cudaFree(gpu_data);
}

Call Sort on Kepler

void sortfile(FILE *in, FILE *out, int N) {
    char *data = (char *)malloc(N);
    fread(data, 1, N, in);
    parallel_sort<<< ... >>>(data, N);
    fwrite(data, 1, N, out);
    free(gpu_data);
}

Call Sort on Pascal

Memory Management Becomes Performance Optimization

No need for opt-in allocator

UNIFIED MEMORY
C++11 IN CUDA 6.5

- Experimental release in CUDA 6.5
  
  `nvcc -std=c++11 my_cpp11_code.cu`

- Support for all C++11 features offered by host compiler in host code

- Currently no support for lambdas passed from host to device
THRUSt: StL-LIKE CUDA TEMPLATE LIBRARY

- GPU(device) and CPU(host) vector class

  ```cpp
  thrust::host_vector<float> H(10, 1.f);
  thrust::device_vector<float> D = H;
  ```

- Iterators

  ```cpp
  thrust::fill(D.begin(), D.begin()+5, 42.f);
  float* raw_ptr = thrust::raw_pointer_cast(D);
  ```

- Algorithms

  - Sort, reduce, transformation, scan, ..

  ```cpp
  thrust::transform(D1.begin(), D1.end(), D2.begin(), D2.end(),
  thrust::plus<float>());  // D2 = D1 + D2
  ```
CUB - CUDA UNBOUND

- Building blocks for kernels via C++ header library
- Data parallel primitives for thread blocks
  - Collective primitives
    - scan, sort, histogram, ..
    - global memory transfer
  
- [http://nvlabs.github.io/cub](http://nvlabs.github.io/cub)
OPENACC DIRECTIVES

Your original Fortran or C code

Program myscience
... serial code ...
!$acc region
do k = 1,n1
do i = 1,n2
... parallel code ...
enddo
enddo
!$acc end region
...
End Program myscience

Easy, Open, Powerful

• Simple Compiler hints
• Works on multicore CPUs & many core GPUs
• Compiler Parallelizes code

http://www.openacc.org
OPENACC: OPEN, SIMPLE, PORTABLE

- Open Standard
- Easy, Compiler-Driven Approach
- Portable on GPUs and Xeon Phi

```c
main() {
    ...
    <serial code>
    ...
#pragma acc kernels
{
    <compute intensive code>
}
...
}
```

CAM-SE Climate
6x Faster on GPU
Top Kernel: 50% of Runtime
Additions for OpenACC 2.0

- Procedure calls
- Separate compilation
- Nested parallelism
- Device-specific tuning, multiple devices
- Data management features and global data
- Multiple host thread support
- Loop directive additions
- Asynchronous behavior additions
- New API routines for target platforms

(CUDA, OpenCL, Intel Coprocessor Offload Infrastructure)
COMPUTATIONAL CHALLENGES IN HEP

Low-Level Trigger
- Connectivity
- Latency control
- Power limited

High-Level Trigger
- Data volume
- Computer vision
- Unsupervised learning

Monte Carlo Analysis
- Portability
- Legacy codes
- Limited parallelism
- Geometry processing

Lattice QCD
- Connectivity
- Low latency
- Bandwidth, bandwidth, bandwidth,...
CUDNN

- Deep Neural Network Library
- Pre-packaged kernels for
  - convolution, pooling, softmax
  - Activations
- ..

developer.nvidia.com/cuDNN
IF YOUR APPLICATION LOOKS LIKE THIS..
.. YOU MIGHT BE INTERESTED IN OPTIX

• Ray-tracing framework
  • Build your own RT application

• Generic Ray-Geometry interaction
  • Rays with arbitrary payloads

• Multi-GPU support
DIFFERENT PROGRAMS GET INVOKED FOR DIFFERENT RAYS

Ray Launcher

Miss program

Any hit program

Closest hit program
OPTIX PRIME: LOW-LEVEL RAY TRACING API

- OptiX simplifies implementation of RT apps
  - Manages memory, data transfers etc

- Sometimes an overkill for simple scene queries
  - E.g. just need visibility of triangulated geometries

- OptiX Prime: Low-Level Tracing API
SUMMARY

- Connectivity
  - GPU Direct RDMA
- Memory limit is non-issue
  - Large memory boards, Unified memory, NVLink
- Portability via abstraction
  - Thrust, CUB, C++11, OpenACC
- Frameworks and libraries for HEP tasks
  - OptiX, cuDNN, ...
- GPUs are ready for HEP tasks!