EXTENDING THE REACH OF PARALLEL COMPUTING WITH CUDA

Mark Harris, NVIDIA

@harrism
#NVSC14
EXTENDING THE REACH OF CUDA

1. Machine Learning
2. Higher Performance
3. New Platforms
4. New Languages
GPUS: THE HOT MACHINE LEARNING PLATFORM

Image Recognition Challenge

1.2M training images • 1000 object categories

Hosted by IMAGENET

GPU Entries

Classification Error Rates
cuDNN  GPU-ACCELERATED DEEP LEARNING

High performance routines for Convolutional Neural Networks

- Optimized for current and future NVIDIA GPUs
- Integrated in major open-source frameworks
  - Caffe, Torch7, Theano
- Flexible and easy-to-use API
- Also available for ARM / Jetson TK1

https://developer.nvidia.com/cuDNN

Baseline Caffe compared to Caffe accelerated by cuDNN on K40

- Caffe (CPU*) 1x
- Caffe (GPU) 11x
- Caffe (cuDNN) 14x

*CPU is 24 core E5-2697v2 @ 2.4GHz Intel MKL 11.1.3
EXTENDING THE REACH OF CUDA

1. Machine Learning
2. Higher Performance
3. New Platforms
4. New Languages
**NVLINK UNLEASHES MULTI-GPU PERFORMANCE**

**GPUs Interconnected with NVLink**

- **CPU**
- **PCIe Switch**
- **TESLA GPU**
- **TESLA GPU**

5x Faster than PCIe Gen3 x16

**Over 2x Application Performance Speedup**
When Next-Gen GPUs Connect via NVLink Versus PCIe

<table>
<thead>
<tr>
<th>Application</th>
<th>Speedup vs PCIe based Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS Fluent</td>
<td>1.00x</td>
</tr>
<tr>
<td>Multi-GPU Sort</td>
<td>1.25x</td>
</tr>
<tr>
<td>LQCD QUDA</td>
<td>1.50x</td>
</tr>
<tr>
<td>AMBER</td>
<td>1.75x</td>
</tr>
<tr>
<td>3D FFT</td>
<td>2.00x</td>
</tr>
</tbody>
</table>

3D FFT, ANSYS: 2 GPU configuration, All other apps comparing 4 GPU configuration
AMBER Cellulose (256x128x128), FFT problem size (256^3)
NVLINK + UNIFIED MEMORY

Simpler, Faster

Share Data Structures at CPU Memory Speeds, not PCIe speeds

Eliminate Multi-GPU Scaling Bottlenecks

NVLink 80 GB/s

5x Faster than PCIe Gen3 x16
EXTENDING THE REACH OF CUDA

1. Machine Learning
2. Higher Performance
3. New Platforms
4. New Languages
Tesla Accelerated Computing Platform

Data Center Infrastructure

- System Solutions
  - CRAY
  - Dell
  - IBM
  - HP
  - EC2
  - Mellanox
  - Adaptive Computing
  - Bright Computing
  - IBM
  - NVidia

- Communication
  - MVAPICH
  - OPEN MPI

- Infrastructure Management
  - GPU Boost
  - GPU Direct
  - NVLink

- System Solutions
  - Compiler Solutions
    - LLVM
  - Interconnect
    - NVML
  - System Management
    - CUDA Debugging API

Development

- Programming Languages
  - C/C++
  - Fortran
  - OpenACC
  - Python

- Development Tools
  - PGI
  - VAMPIR

- Software Solutions
  - allinea DDT
  - Kitware
  - MATLAB
  - Rogue Wave Software

- Libraries
  - cuBLAS
COMMON PROGRAMMING APPROACHES

Across a Variety of Heterogeneous Systems

Libraries
- AmgX
- cuDNN
- cuBLAS
- OpenCV
- Thrust

Compiler Directives
- OpenACC

Programming Languages
- C/C++
- Fortran
- Python
- Java

Power

ARM

x86
EXTENDING THE REACH OF CUDA

1. Machine Learning
2. Higher Performance
3. New Platforms
4. New Languages
MAINSTREAM PARALLEL PROGRAMMING

- Enable more programmers to write parallel software
- Give programmers the choice of language to use
- Embrace and evolve key programming standards
MAINSTREAM PARALLEL PROGRAMMING

- Enable more programmers to write parallel software
- Give programmers the choice of language to use
- Embrace and evolve key programming standards
C++ PARALLEL ALGORITHMS LIBRARY PROGRESS

- Complete set of parallel primitives: for_each, sort, reduce, scan, etc.

- ISO C++ committee voted unanimously to accept as official tech. specification working draft

```cpp
std::vector<int> vec = ...

// previous standard sequential loop
std::for_each(vec.begin(), vec.end(), f);

// explicitly sequential loop
std::for_each(std::seq, vec.begin(), vec.end(), f);

// permitting parallel execution
std::for_each(std::par, vec.begin(), vec.end(), f);
```

N3960 Technical Specification Working Draft:

Prototype:
https://github.com/n3554/n3554
Incorporating OpenACC into GCC is an excellent example of open source and open standards working together to make accelerated computing broadly accessible to all Linux developers.

Oscar Hernandez
Oak Ridge National Laboratories
NUMBA PYTHON COMPILER

Free and open source JIT compiler for array-oriented Python

- numba.cuda module integrates CUDA directly into Python

```python
@cuda.jit("void(float32[:], float32, float32[:], float32[:])")
def saxpy(out, a, x, y):
    i = cuda.grid(1)
    if i < out.size:
        out[i] = a * x[i] + y[i]

# Launch saxpy kernel
saxpy[100, 512](out, a, x, y)
```

- NumbaPro: commercial extension of Numba
  - Python interfaces to CUDA libraries
- http://numba.pydata.org/
ACCELERATING JAVA 3 WAYS

IBM

Drop-In Acceleration

Accelerate Java SE Libraries with CUDA

java.util.Arrays.sort(int[] a)

CUDA C++ Programming Via Java APIs

CUDA4J

Accelerate Pure Java

IBM Developer Kits for Java: ibm.com/java/jdk
void add(int[] a, int[] b, int[] c) throws CudaException, IOException {
    CudaDevice device = new CudaDevice(0);
    CudaModule module = new CudaModule(device,
        getClass().getResourceAsStream("ArrayAdder");
    CudaKernel kernel = new CudaKernel(module, "Cuda_cuda4j_samples_adder");

    CudaGrid grid = new CudaGrid(512, 512);

    try {
        CudaBuffer aBuffer = new CudaBuffer(device, a.length * 4);
        CudaBuffer bBuffer = new CudaBuffer(device, b.length * 4) {
            aBuffer.copyFrom(a, 0, a.length);
            bBuffer.copyFrom(b, 0, b.length);

            kernel.launch(grid, new CudaKernel.Parameters(aBuffer, bBuffer, a.length));

            aBuffer.copyTo(c, 0, a.length);
        }
    }
}
ACCELERATING PURE JAVA ON GPUS

Java 8 Streams and Lambda Expressions

- Express computation as aggregate parallel operations on data streams
- `IntStream.range(0, N).parallel().forEach(i -> c[i] = a[i] + b[i]);`

Benefits

- Standard Java idioms, so no code changes required
- No knowledge of GPU programming model required
- No low-level device manipulation - Java implementation has the controls
- Future JIT smarts do not require application code changes
JIT / GPU OPTIMIZATION OF LAMBDA EXPRESSION

JIT-recognized Java matrix multiplication

```java
Public void multiply() {
    IntStream.range(0, COLS*COLS).parallel().forEach(id -> {
        int i = id / COLS;
        int j = id % COLS;
        int sum = 0;
        for (int k = 0; k < COLS; k++) {
            sum += left[i*COLS + k] * right[k*COLS + j];
        }
        output[i*COLS + j] = sum;
    });
}
```

Speed-up factor when run on a GPU enabled host
COMMON PROGRAMMING APPROACHES

Across a Variety of Heterogeneous Systems

Libraries
- AmgX
- cuDNN
- cuBLAS
- OpenCV
- Thrust

Compiler Directives
- OpenACC

Programming Languages
- C/C++
- Fortran
- Python
- Java

x86

Power

ARM

POWERED