Deep Learning on SHARCNET:
From CPU to GPU cluster

Fei Mao
SHARCNET

E-mail: feimao@sharcnet.ca
List of Topics:

• Deep Learning basics
  – Typical models, computational needs

• HPC and DL
  – Why GPU cluster?
    • Theoretical performance: CPU vs. GPU
    • Demos and benchmarks on DL tools with real world problem
  – Why & how to go deeper and bigger?
    • Monk hardware topology
    • Intra-node comm (GPUDirect P2P)
    • Inter-node comm (CUDA-aware MPI)
Multiple layer networks:
- RBMs (Restricted Boltzmann Machines), DBN (Deep Belief Network)
- Convolutional neural network
Computational needs

- **RBM**:  
  - Do something: MCMC, RNG  
  - Updating weights: minibatch, GEMM

- **CNN**:  
  - Convolutional layers (90-95%): cuDNN, FFT (cuFFT, fbFFT)  
  - Fully connected layers (5-10%)
cuRAND: Up to 70x Faster vs. Intel MKL

<table>
<thead>
<tr>
<th>Distribution</th>
<th>cuRAND</th>
<th>MKL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform Distribution</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>Normal Distribution</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>Log-Normal Distribution</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

**Performance Notes:**
- cuRAND 6.5 on K40c, ECC ON, double-precision input and output data on device
- MKL 11.0.4 on Intel IvyBridge single socket 12-core E5-2697 v2 @ 2.70GHz

Performance may vary based on OS version and motherboard configuration.
Why using GPU?

**cuBLAS: ZGEMM 6x Faster than MKL**

- **cuBLAS 6.5** on K40m, ECC ON, input and output data on device
- **MKL 11.0.4** on Intel IvyBridge single socket 12-core E5-2697 v2 @ 2.70GHz

Performance may vary based on OS version and motherboard configuration.
Why using GPU?

1D used in audio processing and as a foundation for 2D and 3D FFTs

![cuFFT - Single Precision](chart1.png)
![cuFFT - Double Precision](chart2.png)
Why using GPU?

Using Caffe with cuDNN

- Accelerate Caffe layer types by 1.2 - 3x
  Example: AlexNet Layer 2 forward:
  1.9x faster convolution, 2.7x faster pooling

- Integrated into Caffe dev branch today!
  (targeting official release with Caffe 1.0)

Baseline Caffe compared to Caffe accelerated by cuDNN on K40

*CPU is 24 core E5-2697v2 @ 2.4GHz
Intel MKL 11.1.3
Deep Learning tools

• Theano:
  • a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently
  • transparent use of a GPU (defined in ~/.theanorc)
  • efficient symbolic differentiation
  • speed and stability optimizations

• Theano demo and benchmark:
  • Deep Generative Stochastic Networks (GSN)
1. Load python module:
   • On monk, module load python/intel/2.7.8
2. Export PYTHONPATH for other package
   • export PYTHONPATH=/yourpath/python_packages/lib/python2.7/site-packages:$PYTHONPATH
3. Edit “.theanorc” under /home/username
   • Add “device = gpu” under [global]
4. Run/sqsub python run_gsn.py
Theano GPU Speedup

<table>
<thead>
<tr>
<th></th>
<th>Angel (GTX 750Ti)</th>
<th>Monk (M2070)</th>
<th>20 cores, K20</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU (whole node)</td>
<td>1.5</td>
<td>1.5</td>
<td>2.0</td>
</tr>
<tr>
<td>Single GPU</td>
<td>15.0</td>
<td>7.0</td>
<td>15.0</td>
</tr>
</tbody>
</table>

- Angel (GTX 750Ti)
- Monk (M2070)
- 20 cores, K20
Deep Learning tools

• Caffe:
  • A deep learning framework developed with cleanliness, readability, and speed in mind.
  • Fast GPU implementation with cuDNN
  • Designing your CNN without coding

• Caffe demo and benchmark:
  • Training CaffeNet/AlexNet on ImageNet!
1. Load gcc module:
   • Unload intel mkl openmpi and then load gcc/4.8.2
2. Export PATH and LD_LIBRARY_PATH
   • Monk use cuda 6(modified code)
   • Angel use cuda 6.5 with cuDNN
   • Dependencies: /work/feimao/software_installs/
3. Create *LEVELDB* file from images
   • LMDB is default but doesn't work on SHARCNET
4. Modify train_val.prototxt with leveldb read file, batch size, etc.
5. Modify solver.prototxt for lr, momentum, etc.
6. Run/sqsub ./build/tools/caffe train --solver=...
Training on ImageNet with 128 batch size, 40 tiers = 128*40=5120 images
Why go deeper and larger

LeNet-5: 60k Handwritten digits

AlexNet: 1.2m ImageNet (15m pretrain), 7 CNNs, 15.3% top-5 error

GoogLeNet: 6.67% top-5 error
Going deeper is not easy!

- Theano and Caffe support single GPU ONLY!
- Long time to train on big dataset:
  - AlexNet: On a K40 machine, every 20 iterations (5120 images) costs 26.5 seconds to run! 160 hours for 90 epoch!
  - GoogLeNet: K40, 5120 images takes 67.5 seconds! More than a month for 250 epoch!
- How to train huge networks in reasonable time?
GPU cluster is the cure!

• In 2012, Google used a CPU cluster of 1000 nodes (16000 cores) to train a network with 1.8B parameters.

• In 2013, 16-node GPU cluster (COTS HPC, 64 gpus) can train a model with 11.2B parameters!

• In 2015, Baidu used a 36-node GPU cluster (144 gpus) to train a network for ImageNet with larger image size (512x512) and more cropped samples. Top-5 error: 5.98%
Developing on GPU cluster

- Hardware topology: (Monk)

- Infiniband network: ~5GB/S

- PCI-E switch: ~6GB/S
Developing on GPU cluster

• Intra-node comm (GPUDirect P2P)

• w/o P2P: data go to CPU memory
  • CudaSetDevice(0)
  • CudaMalloc(d_0)
  • CudaMemcpy(d_0 to h_0)
  • CudaSetDevice(1)
  • CudaMalloc(d_1)
  • CudaMemcpy(h_0 to d_1)

• w/ P2P: data go through PCI-E bus
  • CudaSetDevice(0)
  • CudaMalloc(d_0)
  • CudaSetDevice(1)
  • CudaMalloc(d_1)
  • CudaMemcpyPeer(d_0 to d_1)
Developing on GPU cluster

• Multi-GPU implementation of Theano:
  – Parallel data loading
  – Data parallelism of AlexNet
  – Using PyCUDA with GPUDirect P2P support
  – ~1.7x speed-up with 2 GPUs
  – [https://github.com/uoguelph-mlrg/theano_alexnet](https://github.com/uoguelph-mlrg/theano_alexnet)
Developing on GPU cluster

- **Inter-node comm (CUDA-aware MPI)**

  - w/o cuda-aware MPI: data go to CPU memory and then IB network
    - CudaMalloc(d_0 on proc_0)
    - CudaMalloc(d_1 on proc_1)
    - CudaMemcpy(d_0 to h_0)
    - MPI_Send(from h_0 to h_1)
    - CudaMemcpy(h_1 to d_1)

  - w/ P2P: GPU data go to IB network through PCI-E bus
    - CudaMalloc(d_0 on proc_0)
    - CudaMalloc(d_1 on proc_11)
    - MPI_Send(from d_0 to d_1)
Conclusion:

Go deeper and deeper and deeper... on GPU cluster!