Scaling Deep Learning Recommender Models (DLRM) with UCC
WHAT’S NEXT?
Accelerating DLRM

Understanding the DLRM model

In the DLRM model, categorical features are processed using embeddings, while continuous features are processed with a bottom multilayer perceptron (MLP). Then, second-order interactions of different features are computed explicitly. Finally, the results are processed with a top MLP and fed into a sigmoid function in order to give a probability of a click. (The schematic representation of this model was discussed in detail in this [Open Compute Project Summit keynote address](#).)
Technical Team

**Nvidia**

**HPC Middleware R&D Team**

- Josh Ladd, Senior Director
- Sergey Lebedev, ML/AI framework acceleration Lead
- Valentin Petrov, UCC SW architect
- Devendar Bureddy, Principal HPC SW Eng.
- Tommy Janjusic, Senior HPC SW Eng.
- Manjunath GV, UCC SW architect

**Facebook**

- Srinivas Sridharan, AI HW/SW codesign
- Pallab Bhattacharya, CEA
- Petr Lapukhov, Net Engg.
- Dimitry Melts, TPM
- Ching-Hsiang Chu, AI HW/SW codesign
- Misha Smelyanskiy, AI HW/SW codesign
Co-designing with UCC

- **AI Workloads/Benchmarks**
  - DLRM
  - PARAM-Comms

- **Communication Library**
  - Performance
  - Robustness

- **Distributed PyTorch**
  - Productivity
  - Portability – GPU/CPU/…
ML Growth and Scale at Facebook

ML data growth
- Usage in 2018: 30%
- Usage today: 50%
- Growth in one year: 3X

1-year Training growth
- Ranking engineers: 2X
- Workflows trained: 3X
- Compute consumed: 3X

Inference Scale per Day
- # of predictions: 400T
- # of translations: 6.5B
- Fake accounts removed: 99%

Maxim Naumov et al., DLRM workloads with implications on hardware and system platforms, OCP Global Summit, 2020
Relative model importance

Maxim Naumov et al. Deep Learning Training in Facebook Data Centers: Design of Scale-up and Scale-out Systems, ArXiv, 2020
Workload characteristics

CV and NLP models are smaller and compute intensive
  • Need high flops
  • Larger on-chip memory helps

Recommender models are huge but have lower compute intensity
  • Need high capacity, high bandwidth memory
  • Unstructured accesses benefit from caches
  • High Network and IO requirements
Deep Learning Recommendation Models

Open sourced https://github.com/facebookresearch/dlrm
Distributed PyTorch
PyTorch

A machine learning framework with an emphasis on:

- Eager & graph-based execution
- Dynamic neural networks
- Distributed training
- Hardware accelerated inference
- Simplicity over complexity
PyTorch Process Group

- **PyTorch Process Group Interface**
  - Abstract collective and p2p interface
  - Portable across platforms (e.g. NVIDIA, x86 CPUs, Google TPU etc.) and comms libraries

https://github.com/pytorch/pytorch/blob/master/torch/distributed/CONTRIBUTING.md
## PyTorch Process Group

<table>
<thead>
<tr>
<th>Backend</th>
<th>gloo</th>
<th>mpi</th>
<th>nccl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>CPU</td>
<td>GPU</td>
<td>CPU</td>
</tr>
<tr>
<td>send</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>recv</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>broadcast</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>all_reduce</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>reduce</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>all_gather</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>gather</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>scatter</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>reduce_scatter</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>all_to_all</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>barrier</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>

**PyTorch UCC Process Group**

- PyTorch UCC Process Group aka Torch-UCC
  - Out-of-box performance, portability, and robustness – across platforms
  - Deploy in FB AI-training systems in production
PARAM-Comms: AI training Comms benchmarks

- PARAM-Comms – https://github.com/facebookresearch/param
  - Blocking vs. non-blocking, single collective – similar to NCCL-tests, OSU/Intel MPI benchmarks
  - Captures DLRM comms behavior – collective pattern + message sizes + training loop
WHY UCC?

- A single PyTorch process group supporting CPU, GPU, and DPU transparently
- UCX building blocks; ultra stable, high mileage
- Vehicle for HW/SW codesign
  - Adaptable to any supported fabric, bus, switch, compute
  - Scalable/Extensible design (future proof)
  - Enable rapid prototyping of new HW designs with production stack
- Open source, community driven
PHASE 1: JOINT OBJECTIVES
DUE DATE: DECEMBER 2020

- Implement Torch-UCC as a 3rd party plugin for PyTorch.
- Support a single UCC Process Group that transparently supports GPU and CPU buffers for allreduce and alltoall.
  - UCC NCCL Team for Allreduce
  - UCC UCX Team for AlltoAll
- Add support for UCC in Facebook/PARAM-Comms benchmark suite.
- Support GPU, CPU, and preliminary POC level support for DPU.
- Evaluate performance on up to 256 GPUs with PARAM-Comms and DLRM on Selene (NV.)
- Deploy in Facebook production environments and evaluate performance and stability.
PHASE 1 DELIVERABLES

Deep Learning Recommender Model

https://github.com/facebookresearch/dlrm

Param Benchmark Support

https://github.com/facebookresearch/param

Torch UCC plugin

https://github.com/openucx/torch-ucc

XCCL - Reference implementation of UCC draft API alltoall and allreduce

https://github.com/openucx/xcl

UCC - Core UCC

https://github.com/openucx/ucc.git
PHASE 2: JOINT OBJECTIVES
DUE DATE: MARCH 2021

- Stable UCC v1.0 API.
- XCCL transitioned to UCC core.
- UCC v1.0 GA release supporting allreduce and alltoall.
- Support GPU, CPU, and alpha-level support for DPU.
- Validate performance and scalability with Param-Comms and DLRM on Selene full scale and deploy in production at Facebook.
PHASE 3: JOINT OBJECTIVES
DUE DATE: SEPT 2021

- UCC v1.2 GA release supporting allreduce and alltoall.
- Support GPU, CPU, DPU.
- Support for NDR offloads (SHARPv3, all-to-all.)
- Best Out-of-Box experience.
- Validate performance and scalability with PARAM-Comms and DLRM on Selene full scale and deploy in production at Facebook.
SELENE SUPERPOD

#5 on Top500

256 Nodes / 2,048 A100 GPUs / 2,048 HDR Adapters / Quantum IB Switch Rails Optimized Topology / fully non-blocking fat-tree

Each Node:

- 8 ConnectX-6 HCAs
- 8 A100 GPUs
- 2 AMD EPYC CPUs
- PCIe Switch
- NVSwitch
Selene RAILS OPTIMIZED TOPOLOGY

Selene 4k SuperPOD
Fat-tree IB Network

FT 3 Layer Solution
Full plane per POD approach
- Fat tree of 140 plane ports

Core wiring
- S1.1 (14) → C1:14 (1)
- Repeat 10 times (10 S connect to 14 C)

Expansion
- Core switches split between Q1 and Q2
- Q1 closet 1 - 28 (L) + 40 (S) + 70 (C)
  - 138 * 650W = 90 KW (all opt)
- Q1 closet 2 - 28 (L) + 40 (S)
  - 68 * 650W = 44 KW (all opt)
- Q1 switch power = 90+44=134 KW

Total Size:
4 Quad * (56 L + 80 S + 36 C) = 684 switches
In all options below: 1:1 in a rail on a 140 node job

Quad to Quad
Quad to Quad core split
- Q1Q2 core supports up to 4 Quads
- Each quad switch power
  - 56 + 80 + 70 = 206 * 650W = 134 KW
- Example bw of 4 quads: bw between in a rail 1:1 in all cases, bw between rails is below:
  - 1 Quad -> 2:1
  - 2, 3 or 4 Quads -> 1:1
UCC ALL-TO-ALL/ALLREDUCE ALGORITHMS

UCX Team

  AlltoAll - Pairwise chunk

  Allreduce - scatter reduce allgather (SRA)

NCCL Team

  Latest v2.8 for both allreduce and alltoall

Hierarchical Team

  AlltoAll - Pairwise exchange using: NCCL Team in-node + UCX Team between nodes
Param-Comms / All-to-All / GPU

Cmd line:
UCX_RNDV_SCHEME=get_zcopy UCX_RNDV_THRESH=64 UCX_TLS=rc,cuda_copy,cuda_ipc,self UCX_IB_SL=1
XCCL_TEAM_UCX_ALLTOALL_PAIRWISE_CHUNK=${chunk} TORCH_UCC_THREAD_ENABLE=1 TORCH_UCC_XCCL_TLS=ucx
/workspace/run-param.sh --device cuda --backend ucc --z 0 --b 524288 --e 536870912 --collective all_reduce -w 50 -i 50
PARAM-COMMS / ALL REDUCE / GPU

Cmd line:
UCX_RNDV_SCHEME=get_zcopy UCX_RNDV_THRESH=64 UCX_TLS=rc,cuda_copy,cuda_ipc,self UCX_IB_SL=1
XCCL_TEAM_UCX_ALLTOALLPAIRWISE_CHUNK=$chunk TORCH_UCC_THREAD_ENABLE=1 TORCH_UCC_XCCL_TLS=ucx /workspace/run-param.sh --device cuda --backend ucc --z 0 --b 524288 --e 536870912 --collective all_reduce -w 50 -i 50
Cmd line:
```
UCX_RNDV_SCHEME=get_zcopy UCX_RNDV_THRESH=2048 UCX_TLS=rc,self XCCL_TEAM_UCX_ALLTOALL_PAIRWISE_CHUNK=$chunk TORCH_UCC_THREAD_ENABLE=1 TORCH_UCC_XCCL_TLS=ucx /workspace/run-param.sh --device cpu --backend ucc --z 0 --b 524288 --e 536870912 --collective all_to_all -w 50 -n 50
```
PARAM-COMMS / ALL REDUCE / CPU

Cmd line:
UCX_RNDV_SCHEME=get_zcopy UCX_RNDV_THRESH=2048 UCX_TLS=rc,self
XCCL_TEAM_UCX_ALLTOALL_PAIRWISE_CHUNK=$chunk TORCH_UCC_THREAD_ENABLE=1 TORCH_UCC_XCCL_TLS=ucx
/workspace/run-param.sh --device cpu --backend ucc --z 0 --b 524288 --e 536870912 --collective all_reduce -w 50 -n 50
DLRM

UCX profile for first 30 iterations.

<table>
<thead>
<tr>
<th>Function</th>
<th>Avg time (us)</th>
<th>Total time (us)</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>ucs_rcache_create_region</td>
<td>28487.69</td>
<td>34128255</td>
<td>1198</td>
</tr>
<tr>
<td>mem_reg</td>
<td>27769.44</td>
<td>33267792</td>
<td>1198</td>
</tr>
<tr>
<td>ibv_reg_mr</td>
<td>27769.16</td>
<td>33267448</td>
<td>1198</td>
</tr>
<tr>
<td>ucp_rndv_receive</td>
<td>4125.605</td>
<td>16362151</td>
<td>3966</td>
</tr>
<tr>
<td>mem_dereg</td>
<td>4060.611</td>
<td>4864613</td>
<td>1198</td>
</tr>
<tr>
<td>ibv_dereg_mr</td>
<td>4060.472</td>
<td>4864445</td>
<td>1198</td>
</tr>
<tr>
<td>ucs_rcache_check_overlap</td>
<td>717.23</td>
<td>859241.2</td>
<td>1198</td>
</tr>
<tr>
<td>ucp_request_memory_dereg</td>
<td>60.382</td>
<td>487106.6</td>
<td>8067</td>
</tr>
<tr>
<td>ucp_rndv_get_completion</td>
<td>46.075</td>
<td>182735.6</td>
<td>3966</td>
</tr>
<tr>
<td>ucp_rndv_progress_rma_get_scapy</td>
<td>0.197</td>
<td>804.87</td>
<td>4086</td>
</tr>
<tr>
<td>ucs_pgttable_insert</td>
<td>0.575</td>
<td>689.16</td>
<td>1198</td>
</tr>
<tr>
<td>ucs_pgttable_lookup</td>
<td>0.046</td>
<td>445.82</td>
<td>9572</td>
</tr>
<tr>
<td>ucp_request_free</td>
<td>0.04</td>
<td>366.25</td>
<td>9189</td>
</tr>
<tr>
<td>ucs_methype_cache_update intern</td>
<td>1.209</td>
<td>313.17</td>
<td>259</td>
</tr>
<tr>
<td>ucs_pgttable_lookup</td>
<td>0.041</td>
<td>311.48</td>
<td>7653</td>
</tr>
<tr>
<td>ucs_pgttable_insert</td>
<td>0.677</td>
<td>175.33</td>
<td>259</td>
</tr>
<tr>
<td>ucp_tag_send_inline</td>
<td>0.032</td>
<td>161.63</td>
<td>5074</td>
</tr>
<tr>
<td>ucs_methype_cache_lookup</td>
<td>0.015</td>
<td>147.24</td>
<td>9572</td>
</tr>
<tr>
<td>ucp_request_memory_reg</td>
<td>0.016</td>
<td>122.53</td>
<td>7707</td>
</tr>
<tr>
<td>uct_iface_mp_chunk_release</td>
<td>10.159</td>
<td>121.91</td>
<td>12</td>
</tr>
</tbody>
</table>
PAIN POINTS

- PyTorch process group implicit completion semantics.
  - Dummy GPU kernel while p2p or collective operations is in progress and finish that kernel once operation is finished.
  - Can’t work with cuda_ipc transport because dummy kernel and cuda ipc calls potentially might block each other and that why we
- Can’t support non-blocking wait mode where computation operations can be inserted into stream before collective is done.
  - Do we need to UCC to be stream aware?
- Node and fabric topology detection missing. Need global view of HCA locality detection algorithms; e.g. for mixed CPU/GPU all-to-all probably need to select HCA based on NUMA locality on sender side and based on GPU locality on receiver side.
- Support graceful abort() in process group; in UCX we have ucp_request_cancel but in UCC we don’t have similar functionality.
- Better resource sharing between core UCX and teams.
THANK YOU