Survey on Distributed Neural Networks and GPU Programming Frameworks

Razvan Florian Vasile [1]

[1]Computer Science, University of Bologna

The Review Process

The goals of this survey are: 1) to analyze the strengths of **distributed training frameworks** for Deep Learning and the **technical overlaps with GPU parallelization libraries**, and 2) gain **practical experience with popular frameworks** (PyTorch DDP, cuDNN, cuBLAS), while **documenting the literature review process**. This systematic mapping aims to fill gaps and find trends in the field.

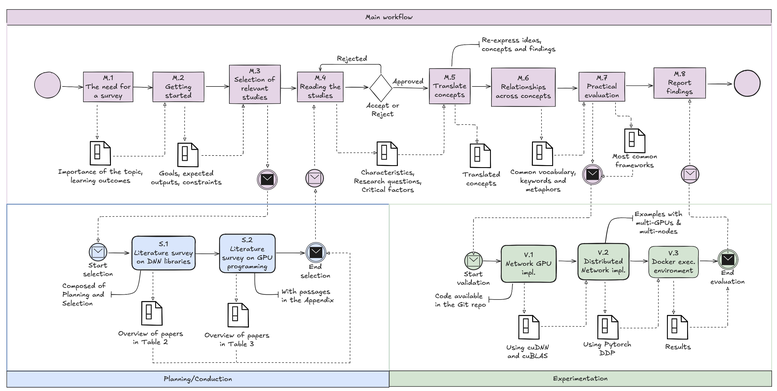
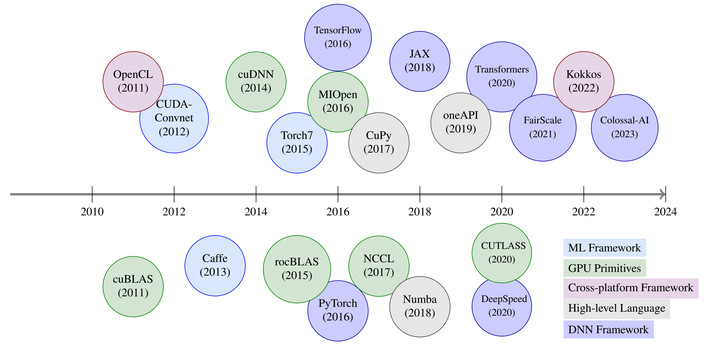


Figure 1. The review workflow documented as a series of steps.

Research Questions

- RQ_1: What are the most commonly cited frameworks for distributed neural network training, and how do their communities vary in size?
- RQ_2: What are the most frequently cited frameworks for GPU programming, and how do their user communities differ in size?
- RQ_3: Which are the overlaps and shared limitations of these areas?
- RQ_4: How can these technologies be applied in practice?

Popular Frameworks Timeline



https://github.com/atomwalk12/deep-bridge-survey

Evaluation Results

Experiments measured forward and backward pass execution times, where the backward pass is performed separately w.r.t. the weights and inputs. A warmup phase is included to accomodate for first-iteration overhead. The network configurations are configured using two files: 1 and 2.

Table 1. Evaluation results for cuDNN and PyTorch networks. Here is the replication code for cuDNN and PyTorch.

Framework	Fwd (ms)	Bwd Input (ms)	Bwd Params (ms)	Total (ms)
cuDNN	0.206760	0.214760	0.028600	0.450120
PyTorch	0.137913	0.112698	0.198538	0.449149

The network configurations include both convolutional and fully connected layers. By adjusting the architectures, it can be seen that the performance is similar for both frameworks. However, Pytorch uses significantly more memory than bare cuDNN.

Connections between the two topics

- Motivations: shared objectives for scalability and performance.
  - **Large models and datasets.** DNNs handle increasingly larger datasets and model capacities, which are especially relevant to NLP tasks.
  - **Hardware optimizations.** GPU programming provides the tools, optimizations and hardware support to achieve better performance. This is done through optimized matrix operations.
- Critical factors: GPU programming as a backbone for DNNs.
  - **Enabling DNNs in critical domains.** GPU libraries like cuDNN allow DNNs to be effective in various critical fields. Without GPUs, DNNs would be infeasible to train in many real-world applications.
  - **Depth (GPU programming) vs. Breadth (DNNs).** GPU programming focuses on providing high-performance within the deep learning domain, while DNNs aim to satisfy the requirements of a wide range of tasks.
- Limitations: heterogenous hardware and algorithmic challenges.
  - **Resource Utilization.** Remains an open-challenge in both areas. While DNNs focus on optimizing bandwidth utilization, GPU programming tries to effectively optimize new architectures across different hardware.
  - **Open-source community.** Collaboration is a key motivation for DNNs. GPU programming ecosystem consists primarily of proprietary libraries as optimized performance requires knowledge of the hardware architecture.

Taxonomy of Popular Frameworks

Figure 3. Taxonomy of distributed deep learning frameworks and GPU programming libraries. References were clear-out due to space constraints but are available in the report.

Relations between the two topics

Table 3. Sample translations displaying connections between the two fields.

ID	Distributed Neural Networks	GPU Programming	Translation
MF2	• The trend to scale up datasets/computational resources is successful in competitions like ImageNet. [D102], [D105], [D103] • The abundance of computation and data are particularly effective in Natural Language Processing (NLP) tasks. [D111]	• Natural parallelizability of deep learning techniques enables training higher capacity networks on larger datasets. [G1012] • Early open-source GPU implementations of CNNs set precedent for code sharing. [G1051]	**Complexity and performance** Effective parallelization techniques yield better performance and widespread adoption of DNNs, especially in NLP tasks. Open-source has accelerated progress.
CF5	A critical factor in the Transformers library is its focus on modular components that simplify pipelines and facilitate ease of use. [D212]	• CuPy is NumPy compatible [G1062] • CuDNN requires more specialized C and CUDA knowledge. [G1015] • Caffe provides flags for easy CPU/GPU switching and clean Python/MATLAB bindings. [G2041]	**Ease of use and hardware flexibility.** • DNN libraries emphasize modularity and ease of extension. • GPU frameworks vary in accessibility - some require specialized knowledge while others provide familiar APIs.
EM1	• Evaluation can be performed behind closed doors for internal processes (speech recognition systems) and subsequently for external applications (Google Search). [D301]	• In many frameworks, the GPU libraries can be switched on and off at compile time using a single flag. • To simplify evaluation and portability, Protocol Buffer files are used. [G3041]	**Deployment:** • Evaluation prioritises staged deployment for safety, and frameworks are designed for flexible deployment across diverse applications and platforms.
LF3	• Tensorflow performs node placement and communication management which results in overhead. [D401] • Some papers emphasizes collaboration in the research community to ensure innovation. [D410]	• Techniques emerge to manage communication overhead by not updating parameters across GPUs on each layer. [G4051] • The existence of CuDNN implies that cross-GPU programming is challenging which requires thorough understanding of the GPU architecture. [G4012], [G4011]	**Communication Overhead & Scalability:** • Requires tradeoffs between communication overhead and performance. • There are no simple universal solutions and choosing the right approach depends on model architectures and hardware. • Community is key to success for DNNs.

Conclusion

- **Distributed training.** The Distributed experiments simulate multi-GPU training workflows locally by leveraging Docker. To facilitate experimentation, the code works with single GPU machines by using the memory of that single GPU.
- **GPU programming.** The GPU experiments implement 2D convolutions, fully connected layers, MSELoss and backpropagation. Future work could expand the code to create pooling layers, dropout and batch normalization modules, enabling the training of more complex models.

Learning outcomes:

- **Experience with literature reviews.** Useful in reading and summarizing scientific literature [1].
- **GPU programming familiarity.** A more thorough understanding of neural network training by utilizing low-level primitives provided by cuDNN [3] and cuBLAS [2].
- **Distributed training experience.** By simulating simple distributed workflows locally, gained practical experience with PyTorch DDP [4] and Docker.

References

[1] Stanford university: How to read a paper. `https://web.archive.org/web/20231216162503/https://web.stanford.edu/class/ee384m/Handouts/HowtoReadPaper.pdf`.

[2] cuBLAS: NVIDIA's GPU-accelerated BLAS library for high-performance linear algebra. `https://developer.nvidia.com/cublas`, 2007.

[3] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cuDNN: Efficient Primitives for Deep Learning. `http://arxiv.org/abs/1410.0759`, December 2014.

[4] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: Experiences on accelerating data parallel training. `https://arxiv.org/abs/2006.15704`. 2020.

razvanflorian.vasile@studio.unibo.it