CS 598

AI Efficiency: Systems and Algorithms

Minjia Zhang

Computer Science Department
University of Illinois at Urbana-Champaign
The Large Language Model Revolution
What is Language Model?
Evolution of DNN Models

- Larger models → better accuracy
- Model size is still growing
- Not reached the accuracy limit yet
- More compute-efficient to train larger models than smaller ones to same accuracy
```

"""
Python 3
Get the current value of a Bitcoin in US dollars using the bitcoincharts api
"""

import requests
import json

def get_bitcoin_price():
    url = 'http://api.bitcoincharts.com/v1/weighted_prices.json'
    response = requests.get(url)
    data = json.loads(response.text)
    return data['USD']['7d']

if __name__ == '__main__':
    print(get_bitcoin_price())
```
ChatGPT: Optimizing Language Models for Dialogue
DALL·E: Creating Images from Text - OpenAI
Autonomous Driving

A Language Agent for Autonomous Driving
Transformers for Language Modeling

---


AI Efficiency Challenges
AI Efficiency Challenges

• Too slow to train high-quality models on massive data
  • More hardware ≠ higher throughput, bigger model
  • Higher throughput ≠ better accuracy, faster convergence, lower cost
  • Better techniques ≠ handy to use

• Slow and expensive to deploy the models
DL System Desired Capabilities (3E)

**Efficiency:** Efficient use of hardware for high scalability and throughput

**Effectiveness:** High accuracy and fast convergence, lowering cost

**Easy to use:** Improve development productivity of model scientists
Research Focus (3Es)

**Efficiency**

Efficient use of hardware for low latency, high scalability and throughput

- **Training efficiency**
  - High performance and cost-efficient training [ATC’21]
    (Training 10x larger model with same GPUs, adopted by major DL frameworks)
  - Graph neural networks [ASPLOS’23]
  - Training w. spot instances [NSDI’23, SOSP’24]

- **Inference efficiency**
  - Recurrent neural networks [ATC’18]
    (10x faster latency and hundreds of millions of dollar saving, Microsoft 2018 top-3 “Cool Tech” showcase)
  - Transformers [SC’22]
Research Focus (3Es)

### Efficiency

- **Training efficiency**
  - High performance and cost-efficient training [ATC’21]
    (Training 10x larger model with same GPUs, adopted by major DL frameworks)
  - Graph neural networks [ASPLOS’23]
  - Training w. spot instances [NSDI’23, SOSP’24]

- **Inference efficiency**
  - Recurrent neural networks [ATC’18]
    (10x faster latency and hundreds of millions of dollar saving, Microsoft 2018 top-3 “Cool Tech” showcase)
  - Transformers [SC’22]

### Effectiveness

- **Model compression**
  - Extreme compression [NeurIPS’22 Oral]
    (50x smaller model size, similar accuracy)
  - Zero-cost quant. [NeurIPS’22 spotlight]
  - 1-bit communication [ICLR’23]

- **Data efficiency**
  - Curriculum learning [NeurIPS’22, spotlight]
    (Retaining 99% accuracy with 10x less data)
  - Adversarial learning [AAAI’22]

- **Efficient architecture**
  - Mixture-of-Experts [ICML’22]
Research Focus (3Es)

**Efficiency**
Efficient use of hardware for low latency, high scalability and throughput

- **Training efficiency**
  - High performance and cost-efficient training [ATC’21]
    (Training *10x larger* model with *same GPUs*, adopted by major DL frameworks)
  - Graph neural networks [ASPLOS’23]
  - Training w. spot instances [NSDI’23, SOSP’24]

- **Inference efficiency**
  - Recurrent neural networks [ATC’18]
    (*10x faster* latency and *hundreds of millions of dollar saving*, Microsoft 2018 top-3 “Cool Tech” showcase)
  - Transformers [SC’22]

**Effectiveness**
High accuracy and fast convergence, lowering cost

- **Model compression**
  - Extreme compression [NeurIPS’22 Oral]
    (*50x smaller* model size, similar accuracy)
  - Zero-cost quant. [NeurIPS’22 spotlight]
  - 1-bit communication [ICLR’23]

- **Data efficiency**
  - Curriculum learning [NeurIPS’22, spotlight]
    (Retaining 99% accuracy with *10x less data*)
  - Adversarial learning [AAAI’22]

- **Efficient architecture**
  - Mixture-of-Experts [ICML’22]

**Easy-to-Use**
Improve development productivity of model scientists

- **DL Compilation**
  - Hardware heterogeneity [IPDPS’21]

- **Auto-Tuner**
  - Adaptive tuning [NeurIPS’20]
    (*3.9x faster* optimization speed)
  - Multi-task tuning [ICLR’21]
Research Focus (3Es)

**Efficiency**
Efficient use of hardware for low latency, high scalability and throughput

- **Training efficiency**
  - High performance and cost-efficient training [ATC’21]
    (Training 10x larger model with same GPUs, adopted by major DL frameworks)
  - Graph neural networks [ASPLOS’23]
  - Training w. spot instances [NSDI’23, SOSP’24]

- **Inference efficiency**
  - Recurrent neural networks [ATC’18]
    (10x faster latency and hundreds of millions of dollar saving, Microsoft 2018 top-3 “Cool Tech” showcase)
  - Transformers [SC’22]

**Effectiveness**
High accuracy and fast convergence, lowering cost

- **Model compression**
  - Extreme compression [NeurIPS’22 Oral]
    (50x smaller model size, similar accuracy)
  - Zero-cost quant. [NeurIPS’22 spotlight]
  - 1-bit communication [ICLR’23]

- **Data efficiency**
  - Curriculum learning [NeurIPS’22, spotlight]
    (Retaining 99% accuracy with 10x less data)
  - Adversarial learning [AAAI’22]

- **Efficient architecture**
  - Mixture-of-Experts [ICML’22]

**Easy-to-Use**
Improve development productivity of model scientists

- **DL Compilation**
  - Hardware heterogeneity [IPDPS’21]

- **Auto-Tuner**
  - Adaptive tuning [NeurIPS’20]
    (3.9x faster optimization speed)
  - Multi-task tuning [ICLR’21]

Industry products: Bing, Ads, Azure, Office

Open-source software

DeepSpeed

tvm

[Image 270x28 to 320x77]
[Image 688x40 to 749x65]
Training Efficiency: Breaking the Memory Wall
ML/DL Training Problem Definition Recap

• Given model f, data set \( \{x_i, y_i\}_{i=1}^N \)

• Minimize the loss between predicted labels and true labels:
  \[
  \text{Min} \frac{1}{N} \sum_{i=1}^{N} \text{loss}(f(x_i, y_i))
  \]

• Common loss function
  • Cross-entropy, MSE (mean squared error)

• Common way to solve the minimization problem
  • Stochastic gradient descent (SGD)
  • Adaptive learning rates optimizers (e.g., Adam)
Gradient Descent

- Model $f_w$ is parameterized by weight $w$
- $\eta > 0$ is the learning rate

For $t = 1$ to $T$

$$\Delta w = \eta \times \frac{1}{N} \sum_{i=1}^{N} \nabla \left( \text{loss} \left( f_w (x_i, y_i) \right) \right) \quad \text{// compute derivative and update}$$

$$w \leftarrow \Delta w \quad \text{// apply update}$$

End
Adaptive Learning Rates (Adam)

- Model $f_w$ is parameterized by weight $w$
- $\eta > 0$ is the learning rate

For $t = 1$ to $T$

$$\Delta w = \eta \times \frac{1}{N} \sum_{i=1}^{N} \nabla \left( \text{loss}(f_w(x_i, y_i)) \right)$$

$$w = w + \Delta w \quad // \text{apply update}$$

End

$\nu_t = \beta_1 \cdot \nu_{t-1} - (1 - \beta_1) \cdot g_t$

$s_t = \beta_2 \cdot s_{t-1} - (1 - \beta_2) \cdot g_t^2$

$$\Delta \omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} \cdot g_t$$

$g_t$: Gradient at time $t$ along $\omega^j$
$\nu_t$: Exponential Average of gradients along $\omega_j$
$s_t$: Exponential Average of squares of gradients along $\omega_j$
$\beta_1, \beta_2$: Hyperparameters

Parallel/Distributed Gradient Descent

• Model $f_w$ is parameterized by weight $w$
• $\eta > 0$ is the learning rate

For $t = 1$ to $T$

\[
\Delta w = \eta \times \frac{1}{N} \sum_{i=1}^{N} \nabla \left( \text{loss} \left( f_w(x_i, y_i) \right) \right)
\]  
// compute derivative and update

$w \leftarrow \Delta w$  
// apply update

End

Can we parallelize it?
Data Parallelism (DP)

1. Partition the training data
2. Parallel training on different machines
3. Synchronize the local updates
4. Refresh local model with new parameters, then go to 2

Implemented as standard component in DL training frameworks, such as PyTorch DDP
Distributed Data Parallel Training in GPU Clusters

Data Parallel Training Loop

Distributed GPU Cluster
# Large Model Training Challenges

<table>
<thead>
<tr>
<th></th>
<th>Bert-Large</th>
<th>GPT-2</th>
<th>Turing 17.2 NLG</th>
<th>GPT-3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameters</strong></td>
<td>0.32B</td>
<td>1.5B</td>
<td>17.2B</td>
<td>175B</td>
</tr>
<tr>
<td><strong>Layers</strong></td>
<td>24</td>
<td>48</td>
<td>78</td>
<td>96</td>
</tr>
<tr>
<td><strong>Hidden Dimension</strong></td>
<td>1024</td>
<td>1600</td>
<td>4256</td>
<td>12288</td>
</tr>
<tr>
<td><strong>Relative Computation</strong></td>
<td>1x</td>
<td>4.7x</td>
<td><strong>54x</strong></td>
<td><strong>547x</strong></td>
</tr>
<tr>
<td><strong>Memory Footprint</strong></td>
<td>5.12GB</td>
<td>24GB</td>
<td><strong>275GB</strong></td>
<td><strong>2800GB</strong></td>
</tr>
</tbody>
</table>
### Large Model Training Challenges

<table>
<thead>
<tr>
<th></th>
<th>Bert-Large</th>
<th>GPT-2</th>
<th>Turing 17.2 NLG</th>
<th>GPT-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>0.32B</td>
<td>1.5B</td>
<td>17.2B</td>
<td>175B</td>
</tr>
<tr>
<td>Layers</td>
<td>24</td>
<td>48</td>
<td>78</td>
<td>96</td>
</tr>
<tr>
<td>Hidden Dimension</td>
<td>1024</td>
<td>1600</td>
<td>4256</td>
<td>12288</td>
</tr>
<tr>
<td>Relative Computation</td>
<td>1x</td>
<td>4.7x</td>
<td>54x</td>
<td>547x</td>
</tr>
<tr>
<td>Memory Footprint</td>
<td>5.12GB</td>
<td>24GB</td>
<td>275GB</td>
<td>2800GB</td>
</tr>
</tbody>
</table>

NVIDIA V100 GPU memory capacity: 16G/32G
NVIDIA A100 GPU memory capacity: 40G/80G

Out of Memory
DNN Training Hit the Memory Wall

Transformer Size: 240x / 2 yrs
AI HW Memory: 2x / 2 yrs

Transformer size: 240x / 2 yrs
AI HW memory: 2.5x / 2 yrs

*Al and Memory Wall. (This blogpost has been written in... | by Amir Gholami | riselab | Medium
How to Break the Memory Wall?
Understanding Memory Consumption

A 16-layer transformer model \( = 1 \text{ layer} \)

*Mixed Precision Training* (ICLR '18) with Adam Optimizer
Understanding Memory Consumption

Each cell represents GPU memory used by its corresponding transformer layer.

*Mixed Precision Training* (ICLR ‘18) with Adam Optimizer
Understanding Memory Consumption

- FP16 parameter

*Mixed Precision Training* (ICLR ‘18) with Adam Optimizer
Understanding Memory Consumption

- FP16 parameter
- FP16 Gradients

*Mixed Precision Training* (ICLR ‘18) with Adam Optimizer
Understanding Memory Consumption

- FP16 parameter
- FP16 Gradients
- FP32 Optimizer States
  - Gradients, Variance, Momentum, Parameters

*Mixed Precision Training* (ICLR ‘18) with Adam Optimizer
Understanding Memory Consumption

- FP16 parameter: 2M bytes
- FP16 Gradients: 2M bytes
- FP32 Optimizer States: 16M bytes
  - Gradients, Variance, Momentum, Parameters

Example 1B parameter model -> 20GB/GPU

Memory consumption doesn’t include:
- Input batch + activations

$M = \text{number of parameters in the model}$

*Mixed Precision Training* (ICLR ‘18) with Adam Optimizer
Distributed Training Strategies

• Pipeline Parallelism

• Tensor Parallelism

• 3D Parallelism

• ZeRO-Style Data Parallelism
Pipeline Parallelism

- Naïve model parallelism leads to severe underutilization
- Gpipe divides batch into micro-batches, enabling different device to work on different micro-batches, reducing pipeline bubbles and improving utilization

**GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism**
Tensor Parallelism

Splice tensors across GPUs

+ synchronization primitives (e.g., all-reduce)

Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism

Supported in:
- DeepSpeed
- Megatron-LM
3D Parallelism

Supported in:
- DeepSpeed
- Megatron-LM

DeepSpeed-extreme-scale-model-training-for-everyone
ZeRO-Style Data Parallelism

- ZeRO removes the redundancy across data parallel process
- Partitioning optimizer states, gradients and parameters (3 stages)

<table>
<thead>
<tr>
<th>Baseline</th>
<th>gpu₀</th>
<th>...</th>
<th>gpuᵢ</th>
<th>...</th>
<th>gpu_N-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_os</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P_os+g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P_os+g+p</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Memory Consumed

- Baseline: \( (2 + 2 + K) \times \Psi \) = 120GB
- P_os: \( 2\Psi + 2\Psi + \frac{K \times \Psi}{N_d} \) = 31.4GB
- P_os+g: \( 2\Psi + \frac{(2+K) \times \Psi}{N_d} \) = 16.6GB
- P_os+g+p: \( \frac{(2+2+K) \times \Psi}{N_d} \) = 1.9GB

Supported in:
- DeepSpeed
- PyTorch
## Large Models Need Parallelism

<table>
<thead>
<tr>
<th>Method</th>
<th>Max Parameter (in billions)</th>
<th>Max Parallelism</th>
<th>Compute Efficiency</th>
<th>Usability (Model Rewrite)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Parallel (DP)</td>
<td>Approx. 1.2</td>
<td>&gt;1000</td>
<td>Very Good</td>
<td>Great</td>
</tr>
<tr>
<td>Tensor Parallel (TP)</td>
<td>Approx. 20</td>
<td>Approx. 16</td>
<td>Good</td>
<td>Needs Model Rewrite</td>
</tr>
<tr>
<td>TP + DP</td>
<td>Approx. 20</td>
<td>&gt; 1000</td>
<td>Good</td>
<td>Needs Model Rewrite</td>
</tr>
<tr>
<td>Pipeline Parallel (PP)</td>
<td>Approx. 100</td>
<td>Approx. 128</td>
<td>Very Good</td>
<td>Needs Model Rewrite</td>
</tr>
<tr>
<td>PP + DP</td>
<td>Approx. 100</td>
<td>&gt; 1000</td>
<td>Very Good</td>
<td>Needs Model Rewrite</td>
</tr>
<tr>
<td>TP + PP + DP</td>
<td>&gt; 1000</td>
<td>&gt; 1000</td>
<td>Very Good</td>
<td>Needs Significant Model Rewrite</td>
</tr>
<tr>
<td>ZeRO</td>
<td>&gt; 1000</td>
<td>&gt; 1000</td>
<td>Very Good</td>
<td>Great</td>
</tr>
</tbody>
</table>
More Interesting Work on Training

• Alpa: Automating Inter- and Intra-Operator Parallelism for Distributed Deep Learning, OSDI 22
• Sequence Parallelism: Making 4D Parallelism Possible, ACL 2023
• Tutel: An efficient mixture-of-experts implementation for large DNN model training, 2023
• Bamboo: Making Preemptible Instances Resilient for Affordable Training of Large DNNs, NSDI 2023
• ...


Tentative Schedule
Hardware Resource
Delta

• Home page: https://www.ncsa.illinois.edu/research/project-highlights/delta/

• 100 quad A100 GPU node, each with 4 A100
• 100 quad A40 GPU node, each with 4 A40
• 5 8-way A100 GPU, each with 8 A100
• 1 MI100 node, 8 MI100
Delta Onboarding Page

• https://docs.ncsa.illinois.edu/systems/delta/en/latest/user_guide/accessing.html
Step 1: Create ACCESS ID

Need access to computing, data analysis, or storage resources?  
You're in the right place! Read more below, or **login** to get started.

**What is an allocation?**
To get started, you need an ACCESS project and some resource units you can spend. **Your ACCESS project and resource units are what we refer to as an Allocation.** An allocation is your project to use a

**Which resources?**
We've got modeling and analysis systems, GPU-oriented systems, large-memory nodes, storage, and more. Resource providers have designed their systems to serve a wide range of research and education needs — including

**Ready to get started?**
It costs you nothing (really!), and you don't need an NSF award. To begin, you just need to

---

ACCESS Allocations: ACCESS (access-ci.org)
Step 2: Submit Resource Requests
Step 2: Submit Resource Requests

- **EXPLORE** — Great for resource evaluation, graduate student projects, small classes and training events, benchmarking, code development and porting, and similar small-scale uses.
- **DISCOVER** — Designed for research grants with modest resource needs, Campus Champions, large classes and training events, NSF graduate fellowships, benchmarking and code testing at scale, and gateway development.
- **ACCELERATE** — Best for experienced users with mid-scale resource needs, consolidating multi-grant programs, collaborative projects, preparing for Maximize ACCESS requests, and gateways with growing communities.
- **MAXIMIZE** — The choice for large-scale research activities that need more resources than the limit for Accelerate ACCESS projects.
Step 3: SSH Login

- Support ssh login
- Maintaining Persistent Sessions: tmux
Delta

• Please give it a try
  • Request access (A100, A40, AMD,...)
  • Ssh login
  • Run a small training/inference job, say PyTorch examples
  • Do preliminary performance profiling

• Let me know if you run into any issues
  • Single GPU allocation
  • Multi-GPU allocation
  • Interactive development
  • Isolation
  • Persistent storage