TAL BEN-NUN

ML ↔ HPC: Optimizing Optimizers for Optimization
Workshop on the Convergence of ML & HPC @ ASPLOS-2020 Zoom

WITH CONTRIBUTIONS FROM DAN ALISTARH, NIKOLI DRYDEN, YOSUKE OYAMA, CEDRIC RENGGLI, AND OTHERS AT SPCL
20 TB/night

Source: OpenAI
A brief intro to supervised deep learning

labeled samples $x \in X \subset \mathcal{D}$

$$w^* = \arg\min_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w, x)]$$

$f(x): X \rightarrow Y$

network structure (fixed)
weights $w$ (learned)

layer-wise parameter update

true label $l(x)$

$$\ell_{sq}(w, x) = (f(x) - l(x))^2$$

$$\ell_{ce}(w, x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$
A brief intro to supervised deep learning

\[ w^* = \arg\min_{w} \mathbb{E}_{x \sim \mathcal{D}} \ell_w(x), \]

\[ \ell_{ce}(w, x) = - \sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}} \]

\[ \ell_{sq}(w, x) = (f(x) - l(x))^2 \]

≥TBs of random access

100MiB-32GiB and beyond

30k-billions

Dataset Size (images) vs. Top-5 accuracy (%)
Trends in deep learning: hardware and multi-node

The field is moving fast – trying everything imaginable – survey results from 252 papers in the area of parallel deep learning

Deep Learning is largely on distributed memory today!

T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019
Trends in distributed deep learning: node count and communication

The field is moving fast – trying everything imaginable – survey results from 252 papers in the area of parallel deep learning

Deep Learning research is converging to MPI!

T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019
Computational Principles

Operators

Networks

Distributed training

Training

Agent

Parameter store

Data
Computational Principles
Example: Options for computing convolutional layers

Direct

\[
\begin{bmatrix}
4 & 1 & 9 & 8 \\
5 & 9 & 9 & 8 \\
0 & 7 & 3 & 4 \\
2 & 6 & 3 & 1 \\
\end{bmatrix}
\times
\begin{bmatrix}
1 & -1 & 0 \\
0.1 & -2 & 0 \\
3 & 4 & 1.1 \\
\end{bmatrix}
= \begin{bmatrix}
21.9 & 59.3 & 53.9 & 43.9 \\
6.3 & 16.8 & 22.3 & 12 \\
9.6 & 15.3 & 25.8 & 14 \\
0.4 & 7.1 & 52.1 & 53.1 \\
\end{bmatrix}
\]

\[N \cdot H' \cdot W'\]

im2col

Indirect

FFT

\[\mathcal{F}\]

Winograd

Domain

Channel-wise summation

Element-wise product

\[m \times m', r \times r, m' \times m'\]

\[m = m + r - 1\]
**Operator Design**

Separable convolution

$$\text{Conv} \ast \text{Conv} = \text{Conv} \ast \text{Conv}$$

Table 4. Depthwise Separable vs Full Convolution MobileNet

<table>
<thead>
<tr>
<th>Model</th>
<th>ImageNet Accuracy</th>
<th>Million Mult-Adds</th>
<th>Million Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv MobileNet</td>
<td>71.7%</td>
<td>4866</td>
<td>29.3</td>
</tr>
<tr>
<td>MobileNet</td>
<td>70.6%</td>
<td>569</td>
<td>4.2</td>
</tr>
</tbody>
</table>


F.N. Iandola et al “Squeezenet: alexnet-level accuracy with 50x fewer parameters and <0.5MB model size,” ICLR 2017.
Transformers – Multi-Head Attention


---

**Diagram: Multi-Head Attention**

- **Scaled Dot-Product Attention**
  - **Linear**
  - **Concat**
  - **SoftMax**
  - **Mask (opt.)**
  - **Scale**
  - **MatMul**

- **Add & Norm**
  - **Feed Forward**
  - **Multi-Head Attention**
  - **Masked Multi-Head Attention**

- **Positional Encoding**
  - **Input Embedding**
  - **Outputs (shifted right)**

- **Input**
  - **Output**

---

DNN Compilers

- Use techniques from compiler construction: DNN → Graph → IR → Transformations → HW Mapping

TensorFlow XLA

Facebook Glow / TorchScript JIT
DNN Compilers

- Use techniques from compiler construction: DNN → Graph → IR → Transformations → HW Mapping

Intel nGraph

TVM Stack
Partitioning Computation?

Data Parallelism
Minibatch Stochastic Gradient Descent (SGD)

T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019
Partitioning Computation?

Data Parallelism

Model Parallelism

Channel/Filter

Spatial

Layer

Pipeline Parallelism

Proc 3

Proc 2

Proc 1

1 1

Idle 1

1 1

1 1

1 2 3

1 2 3

1 2 3

1 2 3

Idle 1

3 2 1

3 2 1

Idle 1

3 2 1
Partitioning Computation?

Data Parallelism
- Simple and efficient solution, easy to implement
- Duplicate parameters at all processors
- Affects generalization

Model Parallelism
- Parameters/domain can be distributed across processors
  - Good for: large inputs, wide networks
- Complex communication per-layer
- Performance hinges on implementation

Pipeline Parallelism
- Parameters can be distributed across processors
  - Good for: deep models, few activations
- Sparse communication pattern (only pipeline stages)
- Consistent model introduces idle-time “Bubble”
Hybrid parallelism

- Layers/parameters can be distributed across processors
- Can distribute minibatch
- Often specific to layer-types (e.g., distribute fc layers but handle conv layers data-parallel)
  - Enables arbitrary combinations of data, model, and pipeline parallelism – very powerful!

J. Dean et al.: Large scale distributed deep networks, NIPS’12.
T. Ben-Nun, T. Hoefer: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019
Training is not just Training

<table>
<thead>
<tr>
<th>Stage 1</th>
<th>Stage 2</th>
<th>Stage 3</th>
<th>Stage 4</th>
<th>Stage 5</th>
<th>Stage 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer 1</td>
<td>A₀</td>
<td>A₀, G₁, ∇E₁</td>
<td>A₀, G₁, ∇E₁</td>
<td>A₀⁻¹, G₁⁻¹, ∇E₁</td>
<td>G₁</td>
</tr>
<tr>
<td>GPU 1</td>
<td>A₁</td>
<td>A₁, G₂, ∇E₂</td>
<td>A₁, G₂, ∇E₂</td>
<td>A₁⁻¹, G₂⁻¹, ∇E₂</td>
<td>G₂</td>
</tr>
<tr>
<td>Layer 2</td>
<td>A₂</td>
<td>A₂, G₃, ∇E₃</td>
<td>A₂, G₃, ∇E₃</td>
<td>A₂⁻¹, G₃⁻¹, ∇E₃</td>
<td>G₃</td>
</tr>
<tr>
<td>Layer 3</td>
<td>A₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data/compute redistribution

Nontrivial gradient aggregation

Imbalanced workload over time

Hyperparameter and Architecture search

- Meta-optimization of hyper-parameters (momentum) and DNN architecture
  - Using Reinforcement Learning [1] (explore/exploit different configurations)
  - Genetic Algorithms with modified (specialized) mutations [2]
  - Particle Swarm Optimization [3] and other meta-heuristics
  - Multi-level optimization

[3] P. R. Lorenzo et al.: Hyper-parameter Selection in Deep Neural Networks Using Parallel Particle Swarm Optimization, GECCO’17
Hyperparameter and Architecture search

- Meta-optimization of hyper-parameters (momentum) and DNN architecture

<table>
<thead>
<tr>
<th>Model</th>
<th># Parameters</th>
<th># Multiply-Adds</th>
<th>Top-1 / Top-5 Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incep-ResNet V2 [44]</td>
<td>55.8M</td>
<td>13.2B</td>
<td>80.4 / 95.3</td>
</tr>
<tr>
<td>ResNetXt-101 [48]</td>
<td>83.6M</td>
<td>31.5B</td>
<td>80.9 / 95.6</td>
</tr>
<tr>
<td>PolyNet [51]</td>
<td>92.0M</td>
<td>34.7B</td>
<td>81.3 / 95.8</td>
</tr>
<tr>
<td>Dual-Path-Net-131 [7]</td>
<td>79.5M</td>
<td>32.0B</td>
<td>81.5 / 95.8</td>
</tr>
<tr>
<td>GeNet-2 [47]*</td>
<td>156M</td>
<td>–</td>
<td>72.1 / 90.4</td>
</tr>
<tr>
<td>Block-QNN-B [52]*</td>
<td>64M</td>
<td>23.8B</td>
<td>75.7 / 92.6</td>
</tr>
<tr>
<td>Hierarchical [30]*</td>
<td>88.9M</td>
<td>25.0B</td>
<td>79.7 / 94.8</td>
</tr>
<tr>
<td>NASNet-A [54]</td>
<td>86.1M</td>
<td>23.1B</td>
<td>82.8 / 96.1</td>
</tr>
<tr>
<td>PNASNet-5 [29]</td>
<td>86.7M</td>
<td>23.1B</td>
<td>82.8 / 96.1</td>
</tr>
<tr>
<td>AmoebaNet-A (N=6, F=190)*</td>
<td>469M</td>
<td>104B</td>
<td>83.9 / 96.6</td>
</tr>
<tr>
<td>AmoebaNet-A (N=6, F=448)*</td>
<td>469M</td>
<td>104B</td>
<td>83.9 / 96.6</td>
</tr>
</tbody>
</table>

[3] P. R. Lorenzo et al.: Hyper-parameter Selection in Deep Neural Networks Using Parallel Particle Swarm Optimization, GECCO’17
Updating parameters in **distributed** data parallelism

\[ w' = u(w, \nabla w) \]

\[ T = 2L + 2P \gamma m/s G \]

- Collective operations
- Topologies
- Neighborhood collectives
- RMA?

\[ T = 2L \log_2 P + 2\gamma mG (P - 1)/P \]

**Hierarchical Parameter Server**

S. Gupta et al.: Model Accuracy and Runtime Tradeoff in Distributed Deep Learning: A Systematic Study. ICDM'16
Parameter (and Model) consistency - centralized

- Parameter exchange frequency can be controlled, while still attaining convergence:
  
<table>
<thead>
<tr>
<th>Synchronous</th>
<th>Stale Synchronous / Bounded Asynchronous</th>
<th>Asynchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter Server</td>
<td>Parameter Server</td>
<td>Parameter Server</td>
</tr>
<tr>
<td>Agent 1 (w^{(0)}), (w^{(1)})</td>
<td>Agent 1 (w^{(0)}), (w^{(1)}), (w^{(2)})</td>
<td>Agent 1 (w^{(0)}), (w^{(1)}), (w^{(2)}), (w^{(3)})</td>
</tr>
<tr>
<td>Agent (m) (w^{(0)}), (w^{(1)})</td>
<td>Agent (m) (w^{(0)}), (w^{(1)}), (w^{(2)})</td>
<td>Agent (m) (w^{(0)}), (w^{(1)}), (w^{(2)}), (w^{(3)})</td>
</tr>
<tr>
<td>Time</td>
<td>Time</td>
<td>Time</td>
</tr>
<tr>
<td>Synchronous</td>
<td>Stale Synchronous / Bounded Asynchronous</td>
<td>Asynchronous</td>
</tr>
</tbody>
</table>

- Trades off "statistical performance" for "hardware performance"

<table>
<thead>
<tr>
<th>Synchronous SGD</th>
<th>Stale-Synchronous SGD</th>
<th>Asynchronous SGD (HOGWILD!)</th>
<th>Model Averaging (e.g., elastic)</th>
<th>Ensemble Learning</th>
<th>Inconsistent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consistent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

J. Dean et al.: Large scale distributed deep networks, *NIPS’12.*
Parameter (and Model) consistency - decentralized

- Parameter exchange frequency can be controlled, while still attaining convergence:


<table>
<thead>
<tr>
<th>Synchronous SGD</th>
<th>Stale-Synchronous SGD</th>
<th>Asynchronous SGD (HOGWILD!)</th>
<th>Model Averaging (e.g., elastic)</th>
<th>Ensemble Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consistent</td>
<td>Inconsistent</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Parameter consistency in deep learning

Using physical forces between different versions of $w$:

$$w^{(t+1,i)} = w^{(t,i)} - \eta \nabla w^{(t,i)} - \alpha (w^{(t,i)} - \bar{w}_t)$$

$$\bar{w}_{t+1} = (1 - \beta)\bar{w}_t + \frac{\beta}{m} \sum_{i=1}^{m} w^{(t,i)}$$

S. Zhang et al.: Deep learning with Elastic Averaging SGD, NIPS’15
Parameter consistency in deep learning

- Synchronous SGD
- Stale-Synchronous SGD
- Asynchronous SGD (HOGWILD!)
- Model Averaging (e.g., elastic)

<table>
<thead>
<tr>
<th>Class</th>
<th>Consistent</th>
<th>Inconsistent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cat</td>
<td>0.54</td>
<td>0.02</td>
</tr>
<tr>
<td>Dog</td>
<td>0.28</td>
<td>0.02</td>
</tr>
<tr>
<td>Airplane</td>
<td>0.07</td>
<td>0.02</td>
</tr>
<tr>
<td>Horse</td>
<td>0.04</td>
<td>0.02</td>
</tr>
<tr>
<td>Bicycle</td>
<td>0.33</td>
<td>0.02</td>
</tr>
<tr>
<td>Truck</td>
<td>0.02</td>
<td></td>
</tr>
</tbody>
</table>

T. G. Dietterich: Ensemble Methods in Machine Learning, MCS 2000
Communication optimizations

- Different options how to optimize updates
  - Send $\nabla w$, receive $w$
  - Send FC factors ($o_{l-1}, o_l$), compute $\nabla w$ on parameter server
    - Broadcast factors to not receive full $w$
  - Use lossy compression when sending, accumulate error locally!

- Quantization
  - Quantize weight updates and potentially weights
  - Main trick is stochastic rounding [1] – expectation is more accurate
    - Enables low precision (half, quarter) to become standard
  - TernGrad - ternary weights [2], 1-bit SGD [3], ...

- Sparsification
  - Do not send small weight updates or only send top-k [4]
    - Accumulate omitted gradients locally

---

[3] F. Seide et al. 1-Bit Stochastic Gradient Descent and Application to Data-Parallel Distributed Training of Speech DNNs, In Interspeech 2014

[source: ai.intel.com]
SparCML – Quantized sparse allreduce for decentral updates

\[ \nabla w_1 + \nabla w_2 + \nabla w_3 + \nabla w_4 \]

Microsoft Speech Production Workload Results – 2 weeks → 2 days!
Opportunities

Frontend

C/C++
FORTRAN
Python
Java
CUDA
OpenCL
...

Source Code

SSA Representation (LLVM IR)

Learnable Representation

Neural Code Comprehension

High-Level (Downstream) Tasks

DNN Malicious Code Detection
DNN Guided Programming
DNN Code Optimization
DNN Hardware Mapping

Anti-Virus
IDE
Compiler

Tasks

Frontend

Source Code

SSA Representation (LLVM IR)

Learnable Representation

Neural Code Comprehension

High-Level (Downstream) Tasks

DNN Malicious Code Detection
DNN Guided Programming
DNN Code Optimization
DNN Hardware Mapping

Anti-Virus
IDE
Compiler

Tasks
Inspiration from Natural Language Processing (NLP)

The Naturalness of Software Hypothesis¹

Software is a form of human communication; software corpora have similar statistical properties to natural language corpora; and these properties can be exploited to build better software engineering tools.

The **domestic cat** is a small, typically furry, carnivorous mammal. They are often called **house cats** when kept as indoor pets or simply **cats** when there is no need to distinguish them from other felids and felines. They are often valued by humans for companionship and for their ability to hunt. There are more than seventy cat breeds recognized by various cat registries.

The **domestic cat** is a small, typically furry, carnivorous mammal. They are often called **house cats** when kept as indoor pets or simply **cats** when there is no need to distinguish them from other felids and felines. They are often valued by humans for companionship and for their ability to hunt. There are more than seventy cat breeds recognized by various cat registries.


<table>
<thead>
<tr>
<th>Natural Language</th>
<th>Programming Languages</th>
</tr>
</thead>
</table>
| **The domestic cat** is a small, typically furry, carnivorous mammal. They are often called **house cats** when kept as indoor pets or simply **cats** when there is no need to distinguish them from other felids and felines. They are often valued by humans for companionship and for their ability to hunt. There are more than seventy cat breeds recognized by various cat registries.** | `int fibonacci(int n){
    if ((n==1)||(n==0))
        return (n);
    else
        return fibonacci(n-1) + fibonacci(n-2);
}` |

<table>
<thead>
<tr>
<th>Read sequentially</th>
<th>Distinct structural features</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Local references</strong></td>
<td><strong>Long range dependencies</strong></td>
</tr>
</tbody>
</table>

```cpp
... int f = fibonacci(my_number);
...```
The **domestic cat** is a small, typically furry, carnivorous mammal. They are often called **house cats** when kept as indoor pets or simply **cats** when there is no need to distinguish them from other felids and felines. They are often valued by humans for companionship and for their ability to **hunt**. There are more than seventy cat breeds recognized by various cat registries.


<table>
<thead>
<tr>
<th><strong>Read sequentially</strong></th>
<th><strong>Distinct structural features</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Local references</td>
<td>Long range dependencies</td>
</tr>
<tr>
<td>Words come from a set vocabulary</td>
<td>High rate of neologisms</td>
</tr>
</tbody>
</table>

```cpp
int fibonacci(int n) {
    if ((n==1)||(n==0))
        return (n);
    else
        return fibonacci(n-1) + fibonacci(n-2);
}
```
**Natural Language**

The *domestic cat* is a small, typically furry, carnivorous mammal. They are often called *house cats* when kept as indoor pets or simply *cats* when there is no need to distinguish them from other felids and felines. They are often valued by humans for companionship and for their ability to hunt. There are more than seventy cat breeds recognized by various cat registries.


**Programming Languages**

```c
int fibonacci(int n){
    if ((n==1)||(n==0))
        return(n);
    else
        return fibonacci(n-1) + fibonacci(n-2);
}
```

0 1 1 2 3 5 8 13 21 34 ...

0 1 2 4 8 16 32 64 128 256 ...

<table>
<thead>
<tr>
<th>Read sequentially</th>
<th>Distinct structural features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local references</td>
<td>Long range dependencies</td>
</tr>
<tr>
<td>Words come from a set vocabulary</td>
<td>High rate of neologisms</td>
</tr>
<tr>
<td>Semantically robust</td>
<td>Semantically brittle</td>
</tr>
</tbody>
</table>
Representations of code

Source Code

```c
int fibonacci(int n){
    if ((n==1)||(n==0))
        return(n);
    else
        return fibonacci(n-1) + fibonacci(n-2);
}
```

Abstract Syntax Tree (AST)

```
if
   return
   n +
   fibonacci(n-1) + fibonacci(n-2)
```

Static Single Assignment (SSA)

```
define i32 @_Z9fibonacci(i32 %n) #0
  %1 = or i32 %n, 1, dbg !16
  %2 = icmp eq i32 %1, 1,dbg !16
  br i1 %2, label %9, label %3, dbg !16
  %4 = add nsw i32 %n, -1, dbg !18
  %5 = tail call i32 @_Z9fibonacci(i32 %4), dbg !19
  %6 = add nsw i32 %n, -2, dbg !20
  %7 = tail call i32 @_Z9fibonacci(i32 %6), dbg !21
  %8 = add nsw i32 %7, %5, dbg !22
  ret i32 %8, dbg !23
ret
```

Assembly

```
push rbp
push rbx
push rax
mov ebx, edi
mov eax, ebx
or eax, 1
je .LBB0_1:
    lea edi, [rbx - 1]
call fibonacci(int)
mov ebp, eax
add ebx, -2
mov edi, ebx
call fibonacci(int)
add eax, ebp
.LBB0_3:
add rsp, 8
pop rbx
pop rbp
```
### Representations of code

**Source Code**

```c
Int fibonacci(Int n){
  if ((n==1)||(n==0))
    return fibonacci(n-1)
  return fibonacci(n-1) + fibonacci(n-2);
}
```

**DeepTune** [Cummins et al. 2017]

**Abstract Syntax Tree (AST)**

```
if
AST paths [Raychev et al. 2015]
code2vec [Alon et al. 2018]
```

**Static Single Assignment (SSA)**

```
define i32 @_Z9fibonaccii(i32 %n) #0 { dbg !
  %1 = or i32 %n, 1, dbg !
  %2 = icmp eq i32 %1, 1, dbg !
  br i1 %2, label %3, label %9, !
  %4 = add nsw i32 %n, -1, !
  %5 = tail call i32 @_Z9fibonaccii(i32 %4), !
  %6 = add nsw i32 %n, -2, !
  %7 = tail call i32 @_Z9fibonaccii(i32 %6), !
  %8 = add i32 %7, %5, !
  ret i32 %8, !
  ret i32 %n, !}
```

**Assembly**

```
inst2vec (LLVM) [Ben-Nun et al. 2018] [Le et al. 2018]
IR2Vec (LLVM) [Keerthy et al. 2019]
CDFG (LLVM) [Brauckmann et al. 2020]
ProGraML (LLVM, XLA) [WIP]
```

**References**

Representations of code

```c
int f(int n) {
    if (n == 0) return 1;
    else return n * f(n-1);
}

int f(int x) {
    if (x == 0) return 1;
    else return x * f(x-1);
}

int fibonacci(int n) {
    if (n == 0) return 1;
    else return n * fibonacci(n-1);
}
```

<table>
<thead>
<tr>
<th>Code2Vec</th>
<th>Inst2Vec</th>
<th>CDFG</th>
</tr>
</thead>
<tbody>
<tr>
<td>factorial</td>
<td>sinc</td>
<td>load</td>
</tr>
<tr>
<td>fact</td>
<td>times</td>
<td>load</td>
</tr>
<tr>
<td>pad</td>
<td>isPowerOfTwo</td>
<td>sext</td>
</tr>
<tr>
<td></td>
<td></td>
<td>getelementptr</td>
</tr>
</tbody>
</table>

- `code2vec`: Factorial (50.93%), Fact (19.15%), Pad (8.92%)
- `inst2vec`: Sinc (77.78%), Times (3.89%), IsPowerOfTwo (3.36%)
- `CDFG`: Load, Load, SExt, GetElementPtr
Self-Supervised Models of Code
ProGraML Overview

Input program

```c
int Fib(int x) {
    switch (x) {
        case 0:
            return 0;
        case 1:
            return 1;
        default:
            return Fib(x - 1) + Fib(x - 2);
    }
}
```

Compiler IR

```assembly
define i32 @Fib(i32) #0 {
    switch i32 %0, label %3 {
        i32 0, label %0
        i32 1, label %2
    }
    ; <label>:2:
    br label %9
    ; <label>:3:
    %4 = add nsw i32 %0, -1
    %5 = tail call i32 @Fib(i32 %4)
    %6 = add nsw i32 %0, -2
    %7 = tail call i32 @Fib(i32 %6)
    %8 = add nsw i32 %7, %5
    ret i32 %8
    ; <label>:9:
```
ProGraML Overview
ProGraML Overview
### ProGraML on compiler tasks

<table>
<thead>
<tr>
<th>Problem</th>
<th>Analysis type</th>
<th>Example optimization</th>
<th>Model</th>
<th>Precision</th>
<th>Recall</th>
<th>$F_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reachability</td>
<td>![Diagram]</td>
<td>Dead code elimination</td>
<td>DeepTune$_{IR}$</td>
<td>0.520</td>
<td>0.497</td>
<td>0.504</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ProGraML</td>
<td>0.997</td>
<td>0.995</td>
<td>0.996</td>
</tr>
<tr>
<td>DomTree</td>
<td>![Diagram]</td>
<td>Global Code Motion</td>
<td>DeepTune$_{IR}$</td>
<td>0.721</td>
<td>0.081</td>
<td>0.138</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ProGraML</td>
<td>0.985</td>
<td>0.693</td>
<td>0.781</td>
</tr>
<tr>
<td>DataDep</td>
<td>![Diagram]</td>
<td>Instruction scheduling</td>
<td>DeepTune$_{IR}$</td>
<td>0.999</td>
<td>0.136</td>
<td>0.236</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ProGraML</td>
<td>1.000</td>
<td>0.988</td>
<td>0.993</td>
</tr>
<tr>
<td>Liveness</td>
<td>![Diagram]</td>
<td>Register allocation</td>
<td>DeepTune$_{IR}$</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ProGraML</td>
<td>1.000</td>
<td>0.999</td>
<td>0.999</td>
</tr>
<tr>
<td>Subexpressions</td>
<td>![Diagram]</td>
<td>Global Common Subexpression Elimination</td>
<td>DeepTune$_{IR}$</td>
<td>1.000</td>
<td>0.123</td>
<td>0.214</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ProGraML</td>
<td>0.965</td>
<td>0.925</td>
<td>0.930</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td>DeepTune$_{IR}$</td>
<td>0.810</td>
<td>0.209</td>
<td>0.273</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ProGraML</td>
<td>0.989</td>
<td>0.920</td>
<td>0.940</td>
</tr>
</tbody>
</table>
Case Study: Algorithm Classification (104 Classes)

Figure 1: (a) Illustration of an AST, corresponding to the C code snippet “int a=b+3;” It should be noticed that our model takes as input the entire AST of a program, which is typically much larger. (b) The architecture of the Tree-Based Convolutional Neural Network (TBCNN). The main components in our model include vector representation and coding, tree-based convolution and dynamic pooling; then a fully-connected hidden layer and an output layer (softmax) are added.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>i2v</td>
<td>rnd</td>
<td>GGNN-s</td>
<td>GGNN-(8/1)</td>
</tr>
<tr>
<td>Test Error [%]</td>
<td>6.0</td>
<td>5.17</td>
<td>4.56</td>
<td>3.87</td>
</tr>
<tr>
<td>Improvement [%]</td>
<td>—</td>
<td>0.0</td>
<td>11.8</td>
<td>25.1</td>
</tr>
</tbody>
</table>

Case Study: Heterogeneous Device Mapping

<table>
<thead>
<tr>
<th>Model</th>
<th># Params</th>
<th>AMD (Acc. [%])</th>
<th>NVIDIA (Acc. [%])</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static Mapping</td>
<td>—</td>
<td>58.8</td>
<td>56.9</td>
</tr>
<tr>
<td>DeepTune [20]</td>
<td>77k</td>
<td>83.7</td>
<td>80.3</td>
</tr>
<tr>
<td>inst2vec [13]</td>
<td>650k</td>
<td>82.8</td>
<td>82.1</td>
</tr>
<tr>
<td>inst2vec-imm [13]</td>
<td>650k</td>
<td>88.1</td>
<td>86.6</td>
</tr>
<tr>
<td>ProGRAML Transformer</td>
<td>4.8M</td>
<td>88.3 ± 0.8</td>
<td>87.2 ± 0.8</td>
</tr>
<tr>
<td>ProGRAML Transformer (pretrained)</td>
<td>8600</td>
<td>89.6 ± 0.4</td>
<td>88.0 ± 0.4</td>
</tr>
</tbody>
</table>

CPU or GPU?

<table>
<thead>
<tr>
<th>Computing Platform</th>
<th>DeepTune 77k</th>
<th>DeepTune-TL 69k</th>
<th>inst2vec 650k</th>
<th>ProGraML-transfer 3600</th>
<th>Optimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD Radeon HD 5900</td>
<td>1.10</td>
<td>1.17</td>
<td>1.37</td>
<td>1.28 ± 0.00</td>
<td>1.50</td>
</tr>
<tr>
<td>AMD Tahiti 7970</td>
<td>1.05</td>
<td>1.23</td>
<td>1.10</td>
<td>1.12 ± 0.00</td>
<td>1.40</td>
</tr>
<tr>
<td>Nvidia GTX 480</td>
<td>1.10</td>
<td>1.14</td>
<td>1.07</td>
<td>1.02 ± 0.00</td>
<td>1.34</td>
</tr>
<tr>
<td>Nvidia Tesla K20c</td>
<td>0.99</td>
<td>0.93</td>
<td>1.06</td>
<td>0.96 ± 0.00</td>
<td>1.18</td>
</tr>
</tbody>
</table>
## Case Study: Branch Prediction

<table>
<thead>
<tr>
<th>Model</th>
<th># Parameters</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer (pretrained, random init)</td>
<td>602</td>
<td>0.30</td>
</tr>
<tr>
<td>Transformer (random init)</td>
<td>4.8M</td>
<td>0.19</td>
</tr>
<tr>
<td>Transformer (pretrained)</td>
<td>602</td>
<td>0.11</td>
</tr>
<tr>
<td>Transformer</td>
<td>4.8M</td>
<td>0.01</td>
</tr>
</tbody>
</table>
Summary – HPC → ML

- A supercomputing problem - amenable to established tools and tricks from HPC
- Concurrency is easy to attain, hard to program beyond data-parallelism
- Main bottleneck in distributed is communication – reduction by using the robustness of SGD
- Co-design is prevalent
- Very different environment from traditional HPC
  - Trade-off accuracy for performance!
- Main objective is generalization
  - Performance-centric view in HPC can be harmful for accuracy

https://www.arxiv.org/abs/1802.09941

Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurreny Analysis

TAL BEN-NUN and FOREN HOEFLER, ETH Zurich

Deep Neural Networks (DNNs) are becoming an important tool in modern computing applications. Accelerating their training is a major challenge and techniques range from distributed algorithms to low-level circuit design. In this survey, we describe the problem from a theoretical perspective, followed by approaches for its parallelization. Specifically, we present trends in DNN architectures and the resulting implications on parallelization strategies. We discuss the different types of concurrency in DNNs: synchronous and asynchronous, stochastic gradient descent, distributed system architectures, communication schemes, and performance modeling. Based on these approaches, we extrapolate potential directions for parallelism in deep learning.

CCS Concepts: → General and applied sciences → Theory and practice of computer science → Computing methodologies → Neural networks; → Distributed computing methodologies → Parallel computing methodologies; → Machine learning.

Additional Key Words and Phrases: Deep Learning, Distributed Computing, Parallel Algorithms

ACM Reference Format:
Summary – ML → HPC

- Categorizing and understanding code is essential for various tasks
- Reasoning about code requires different tools than natural languages
- Using classical compiler construction, structure can be recovered
- Results are promising for various classes of downstream tasks