Learning Execution through Neural Code Fusion
Zhan Shi, Kevin Swersky, Danny Tarlow, Parthasarathy Ranganathan, Milad Hashemi
Overview

● Motivation
● Background
● Neural Code Fusion
● Experimental Results
● Conclusion
Motivation
2% Performance/Year is the New Normal

Source: Parthasarathy Ranganathan, More Moore: Thinking Outside the (Server) Box
Motivation

- **Dynamic** speculative execution
  - Branch prediction, value prediction, cache replacement, prefetching...
Motivation

- **Dynamic** speculative execution
  - Branch prediction, value prediction, cache replacement, prefetching...

- **Static** source code
  - Variable naming, finding bugs, algorithm classification, program synthesis...
  - Performance-related tasks: device mapping, thread coarsening, throughput prediction...
Motivation

- **Dynamic** speculative execution
  - Branch prediction, value prediction, cache replacement, prefetching...
- **Static** source code
  - Variable naming, finding bugs, algorithm classification, program synthesis...
  - Performance-related tasks: device mapping, thread coarsening, throughput prediction...
- **Both views provide useful features**
Example: a “Simple” Case for Branch Prediction

for (i = 0; i < k; i++)
{
}

Example: a “Simple” Case for Branch Prediction

```c
for (i = 0; i < k; i++)
{
}
```

Highly biased
Example: a “Simple” Case for Branch Prediction

```c
for (i = 0; i < k; i++)
{
}
```

- Highly biased
- Branch history doesn’t help
Example: a “Simple” Case for Branch Prediction

```plaintext
while(...)
    generate k;
    for (i = 0; i < k; i++)
        {
            
        }
```

- Highly biased
- Branch history doesn’t help
Example: a “Simple” Case for Branch Prediction

while(...){
generate k;
for (i = 0; i < k; i++) {
}
}

- Highly biased
- Branch history doesn’t help
- Jump out when “close enough”
- Predictable if we knew the relation

[Static] i and k are compared
[Dynamic] values of i and k
Background: Graph Neural Networks
Background: Graph Neural Networks

- Typical deep learning operates on IID data points.
Background: Graph Neural Networks

- What if the data points had *relational* information?

Battaglia et al., 2018
Background: Graph Neural Networks

- Message passing
Background: Graph Neural Networks

- Message passing
Background: Graph Neural Networks

- Message passing

Input graph

Step 0  Step 1
Background: Graph Neural Networks

- **Message passing**

![Graph Neural Networks Diagram]

1. **Input graph**
2. **Step 0**
3. **Step 1**
4. **Step 2**
Background: Graph Neural Networks

- Message passing
Programs as Graphs  Allamanis et al., 2017

(a) Simplified syntax graph for line 2 of Fig. 1, where blue rounded boxes are syntax nodes, black rectangular boxes syntax tokens, blue edges Child edges and double black edges NextToken edges.

(b) Data flow edges for $(x_1, y^2) = \text{Foo}()$; while $(x^3 > 0)$ $x^4 = x^5 + y^6$ (indices added for clarity), with red dotted LastUse edges, green dashed LastWrite edges and dashdotted purple ComputedFrom edges.
Representing Static and Dynamic Information

- Graphs are an effective representation for static code

- How do we generally represent dynamic information in a model?
Neural Code Fusion
Full System

Program preprocessing

Source code .cpp, .h

Compiler

Assembly code .asm

Static/Dynamic Data Collection

Static graph

Dynamic snapshots

Learning Control/Data Flow Behavior

Gated Graph neural networks

Linear Mapping

Control flow

Data flow
Assembly vs Source Code

- Highly structured
## Assembly vs Source Code

- Highly structured

| if-else | Ternary | Assembly | Semantics |
|---------|---------|----------|-----------|
| if      | (a<b)   | `i = a;` | # fetch a |
| else    |         | `i = b;` | # compare a and b |

```assembly
4004da:    mov  -0xc(%rbp),%eax
4004dd:    cmp  -0x8(%rbp),%eax
4004e0:    jge  4004ea
4004e2:    mov  -0xc(%rbp),%eax
4004e5:    mov  %eax,-0x4(%rbp)
4004e8:    jmp  4004f0
4004ea:    mov  -0x8(%rbp),%eax
4004ed:    mov  %eax,-0x4(%rbp)
4004f0:    mov  $0x1,%eax
```
### Assembly vs Source Code

- **Highly structured**
- **Directly relate data to program semantics**

| if-else | Ternary | Assembly | Semantics |
|---------|---------|----------|-----------|
| if $(a < b)$ | $i = a$; | 4004da: mov -0xc(%rbp),%eax | # fetch a |
| else | $i = b$; | 4004dd: cmp -0x8(%rbp),%eax | # compare a and b |
| | | 4004e0: jge 4004ea | # jump to $i = b$ if $a >= b$ |
| | | 4004e2: mov -0xc(%rbp),%eax | # fetch a |
| | | 4004e5: mov %eax,-0x4(%rbp) | # $i = a$ |
| | | 4004e8: jmp 4004f0 | # jump out |
| | | 4004ea: mov -0x8(%rbp),%eax | # fetch b |
| | | 4004ed: mov %eax,-0x4(%rbp) | # $i = b$ |
| | | 4004f0: mov $0x1,%eax | |
Assembly vs Source Code

- Highly structured
- Directly relate data to program semantics
- Easy to use for architecture tasks
PC0: mov $0x48(%rbx), %rdi
PC1: cmp (%rdi, %rax, 1), %rsi
PC2: jne PC0
Dynamic Tasks: Control Flow and Data Flow

- Control flow (branch prediction)
  - predict whether a branch statement will be taken or not taken.
  - Set branch instruction node to be the target node.
  - Binary classification
Dynamic Tasks: Control Flow and Data Flow

- **Control flow (branch prediction)**
  - predict whether a branch statement will be taken or not taken.
  - Set branch instruction node to be the target node.
  - Binary classification

- **Data flow (prefetching)**
  - predict which address will be accessed next.
  - Set src node to be the target node.
  - Predict 64-bit address
Multi-Task Representation

- Many other static/dynamic tasks can be defined on the graph simultaneously
  - Value prediction, indirect branch prediction, memory disambiguation, caching...
Dynamic Snapshots

- Snapshots
  - The values of the set of variable nodes
  - Captured during program execution
- Used to initialize the graph neural network
Representation Study

- Number “3” in different representations
  - Categorical: [1, 0, 0, 0]
  - Scalar: 3
  - Binary: 11
Representation Study

- Correctly predict when to jump out
- Sample k values as training data

```java
for(k=0; k < n; k+=3){
    for (i = 0; i < k; i++)
        {
        }
}
```
Representation Study:

- Results
  - Binary > scalar > categorical
Experimental Results
Experimental Setup

- **Benchmarks**
  - SPEC06 INT

- **Tasks**
  - Dynamic: control flow (branch prediction) and data flow (prefetching)
  - Static: algorithm classification

- **Offline evaluation for both NCF and baselines**
  - 70% training
  - 30% testing
Control-flow (Branch Prediction) and Data-flow (Prefetching)
Algorithm Classification

- Test the usefulness of the learned representation
- We pre-train our GNN on the control-flow task
- A simple linear SVM model
- We get 96% vs 95.3% (50M lines of LLVM IR) using 200k lines of assembly with no external data sources.
Summary

- NCF combining static and dynamic information
  - creates useful representations
- Different from the traditional dynamic models in architecture
  - Data is usually purely dynamic
  - Model is history-based
- Enhances static models with dynamic program behavior
  - Learned representation can also transfer to a unseen static task
Thank you!

Questions?