DPGraph:
Code Optimization with Dynamic Program Graphs using Heterogeneous GNNs

Quazi Ishtiaque Mahmud *Lukas Rothenberger* *Le Chen*
Felix Wolf *Ali Jannesari*

IOWA STATE UNIVERSITY OF SCIENCE AND TECHNOLOGY
TECHNISCHE UNIVERSITÄT DARMSTADT
Argonne NATIONAL LABORATORY

This work is supported by NSF grants #2211982 and #2426580

Motivation

- Existing LLVM IR-based program representations
  - Mostly use static dependencies
  - So, they can not track dynamically allocated memory
  - Also, can not track dynamically calculated array indices
- Dynamic dependence profiling helps to captures those dependencies that occur at runtime.
- Hence, it is important to incorporate dynamic dependence profiling info in program graph representations.

Contributions

- **DPGraph** leverages dynamic analysis information to construct a flow-aware program graph
  - generates program graph embedding consisting of dependency types and useful program features (e.g., input size)
  - identifies optimal configurations within a source program
- The program graph is constructed using fine-grained LLVM-based Intermediate Representation (IR).
- **DPGraph** is evaluated on downstream tasks related to code performance optimization
  - **13.56%** better performance than SOTA models on VF/IF prediction,
  - **7.06%** better performance than SOTA models on CPU/GPU-based parallelism detection,
  - **7.05%** better performance than SOTA models on identifying optimal loop order.

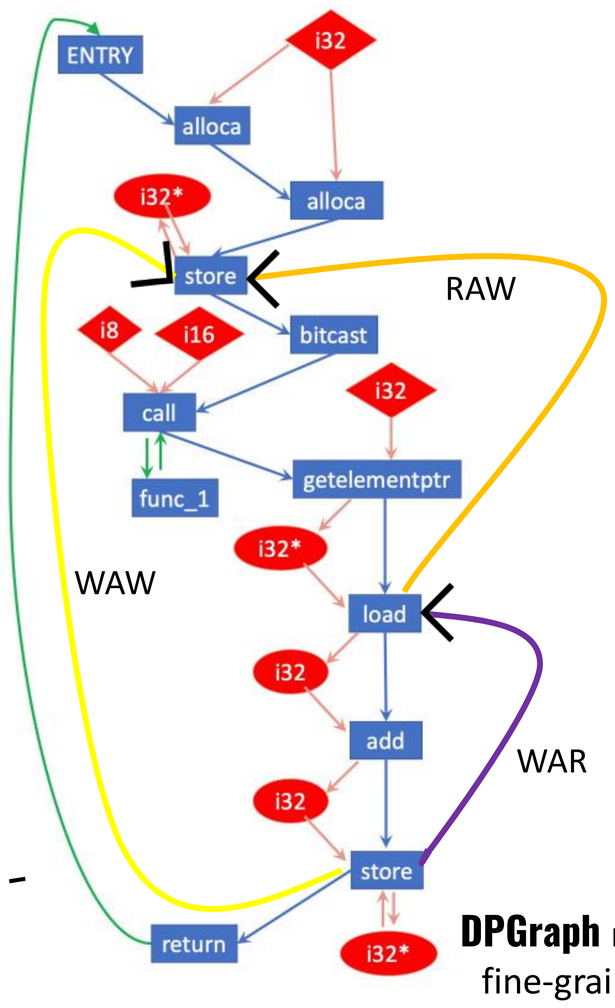
Approach

```
#include <stdio.h>
int main (){
int k = 0, i, s = 0;
for (i = 0; i < 128;
i++){
  k = k + 3;
  s = s + k;}
return 0;}
```

Hybrid analysis Using DiscoPoP

```
START 1:103
….
!426 NOM  RAW !437|b(b) RAW
!354|b(b)
!354 NOM  INIT *|b(b)
!465 NOM  RAW !437|b(b)
!414 NOM  RAW !456|i(S3) RAW
!381|i(S3)
!433 NOM  RAW !443|j(S2) RAW
!399|j(S2)
!428 NOM  RAW !456|i(S3) RAW
!381|i(S3)
!320 NOM  RAW !314|i(S3) RAW
!371|i(S3)
!381 NOM  WAR !320|i(S3)
!300 NOM  INIT *|m(S6)
!369 NOM  RAW !314|i(S3) RAW
!371|i(S3)
!471 NOM  RAW !469|_ret_val_0(S8)
!345 NOM  RAW !314|i(S3) RAW
!371|i(S3)
!441 NOM  RAW !443|j(S2) RAW
!399|j(S2)
……..
```

Profiled IR to DPGraph

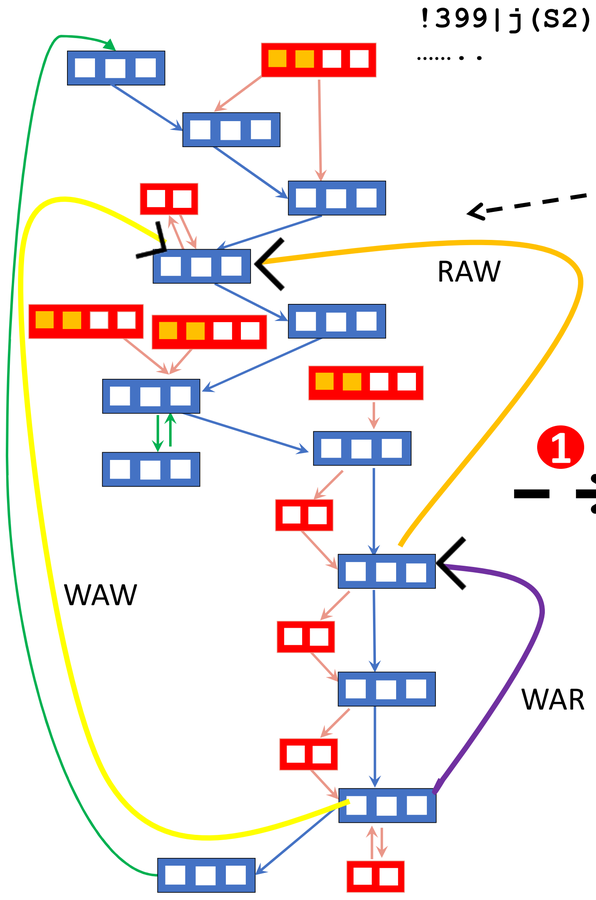


Node Symbol	Type
▭	Control
⬭	Variable
◆	Constant

Edge Symbol	Flow type
→	Control
→	Data flow
→	Call flow

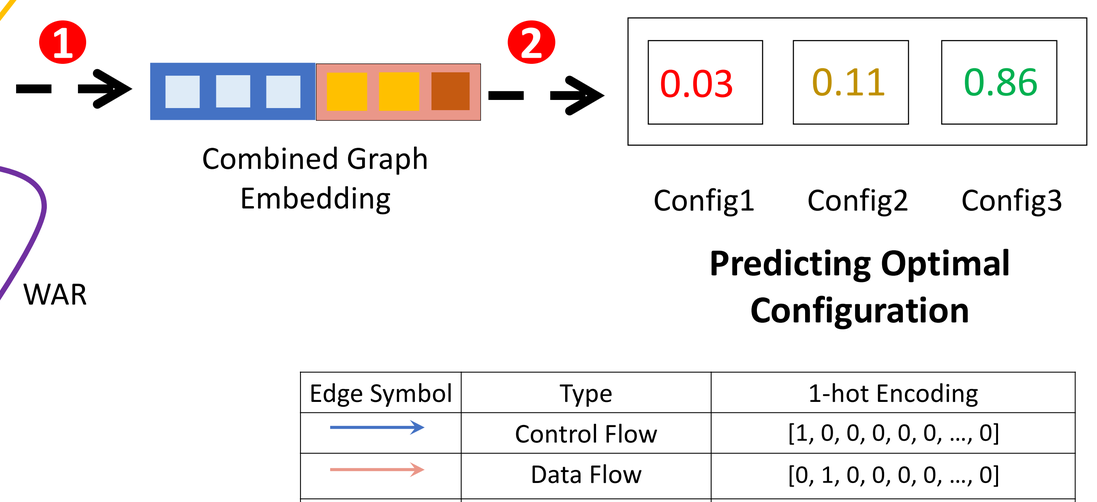
DPGraph representation with fine-grained dependencies

Tokenized IR and Embedding Generation



❶ Apply GNN with graph pooling (Graph Encoder)

❷ Linear Transformation + Softmax



Predicting Optimal Configuration

Node Symbol	Type	Embedding Symbol	Features
▭	Control	▭	Tokenized IR instructions
⬭	Variable	▭	Data type of variable
◆	Constant	▭	Data type & value of constant

Edge Symbol	Type	1-hot Encoding
→	Control Flow	[1, 0, 0, 0, 0, 0, …, 0]
→	Data Flow	[0, 1, 0, 0, 0, 0, …, 0]
→	Call Flow	[0, 0, 1, 0, 0, 0, …, 0]
WAR →	Dependency (WAR)	[0, 0, 0, 1, 0, 0, …, 0]
RAW →	Dependency (RAW)	[0, 0, 0, 0, 1, 0, …, 0]
WAW →	Dependency (WAW)	[0, 0, 0, 0, 0, 1, …, 0]

Code	PerfoGraph	DPGraph	Ground Truth
`__attribute__((noinline)) void example1(unsigned *ret, int n) { int i; unsigned int res = 0; for (i = 0; i < n; i++) { res += (in1[i] + in2[i]); } *ret = res; return; }` n = 64	VF = 4 IF = 1	VF = 4 IF = 1	VF = 4 IF = 1
`__attribute__((noinline)) void example2(unsigned *ret, int n) { int i; unsigned int add = 0; for (i = 0; i < n; i++) { add += (ub[i] + uc[i]); } *ret = add; return; }` n = 128	VF = 4 IF = 1	VF = 4 IF = 2	VF = 4 IF = 2

Ablation

Approach	Accuracy
DPGraph w/o dynamic dependencies	79.01%
DPGraph w/o input size embedding	83.56%
DPGraph	**88.08%**

Ablation performed on 650 test files for VF/IF prediction task from NeuroVectorizer Dataset

Results

VF/IF Prediction

Tool	Speedup
MLIR	1.59x
NeuroVect	1.68x
PerfoGraph	1.71x
MIREncoder	1.77x
DPGraph (our)	**2.01x**

Evaluation performed on 650 test files from NeuroVectorizer Dataset

Simd/target Prediction (CPU/GPU)

Tool	Speedup
PrograML	6.67x
PefoGraph	9.82x
AutoParLLM	10.09x
ConTraPh	14.58x
DPGraph (our)	**15.61x**

Evaluation performed on 159 test files sourced from NPB, Rodinia, DataRaceBench

Loop Order Prediction

Tool	Speedup
Polly	1.52x
PrograML	1.53x
UJOpt	1.54x
PerfoGraph	1.56x
DPGraph (our)	**1.67x**

Evaluation performed on 31 test files sourced from PolyBench-3.2 Dataset