IOWA STATE UNIVERSITY

Department of Computer Science

"Where should I start for Parallelization?"

- A Graph Neural Network Based Parallelism Detection Approach

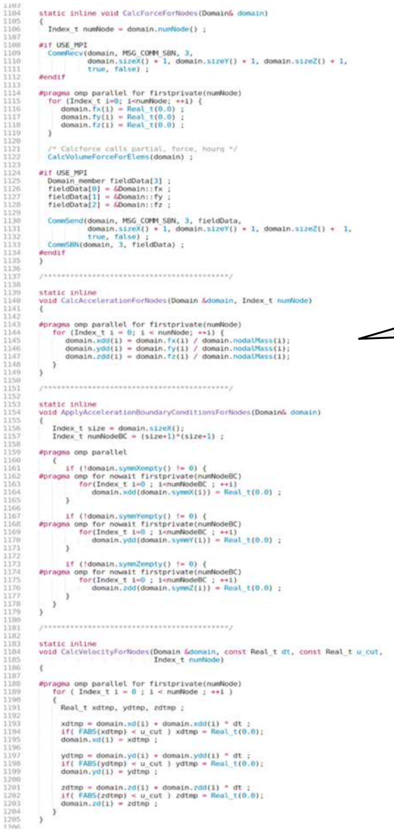
Le Chen, Ali Jannesari

Iowa State University

CnC'21 CnC

Introduction

Parallelizing sequential programs is not so easy.



```
#pragma omp parallel for firstprivate(numNode)
for (Index_t i = 0; i < numNode; ++i) {
            domain.xdd(i) = domain.fx(i) / domain.nodalMass(i);
            domain.ydd(i) = domain.fy(i) / domain.nodalMass(i);
            domain.zdd(i) = domain.fz(i) / domain.nodalMass(i);
   }
}
```

1. discover potential parallelism in sequential applications

⇩

2. apply parallelism

Motivation

Auto parallelization is complicated

- Abandon fully automatic parallelization with compilers
- Instead, we start by pointing programmers to likely parallelization opportunities

Current assisted parallelization tools:

- DiscoPoP: www.discopop.org , https://github.com/discopop-project/discopop
- AutopaR
- etc

DiscoPoP TECHNISCHE UNIVERSITÄT DARMSTADT IOWA STATE UNIVERSITY

Motivation

However:

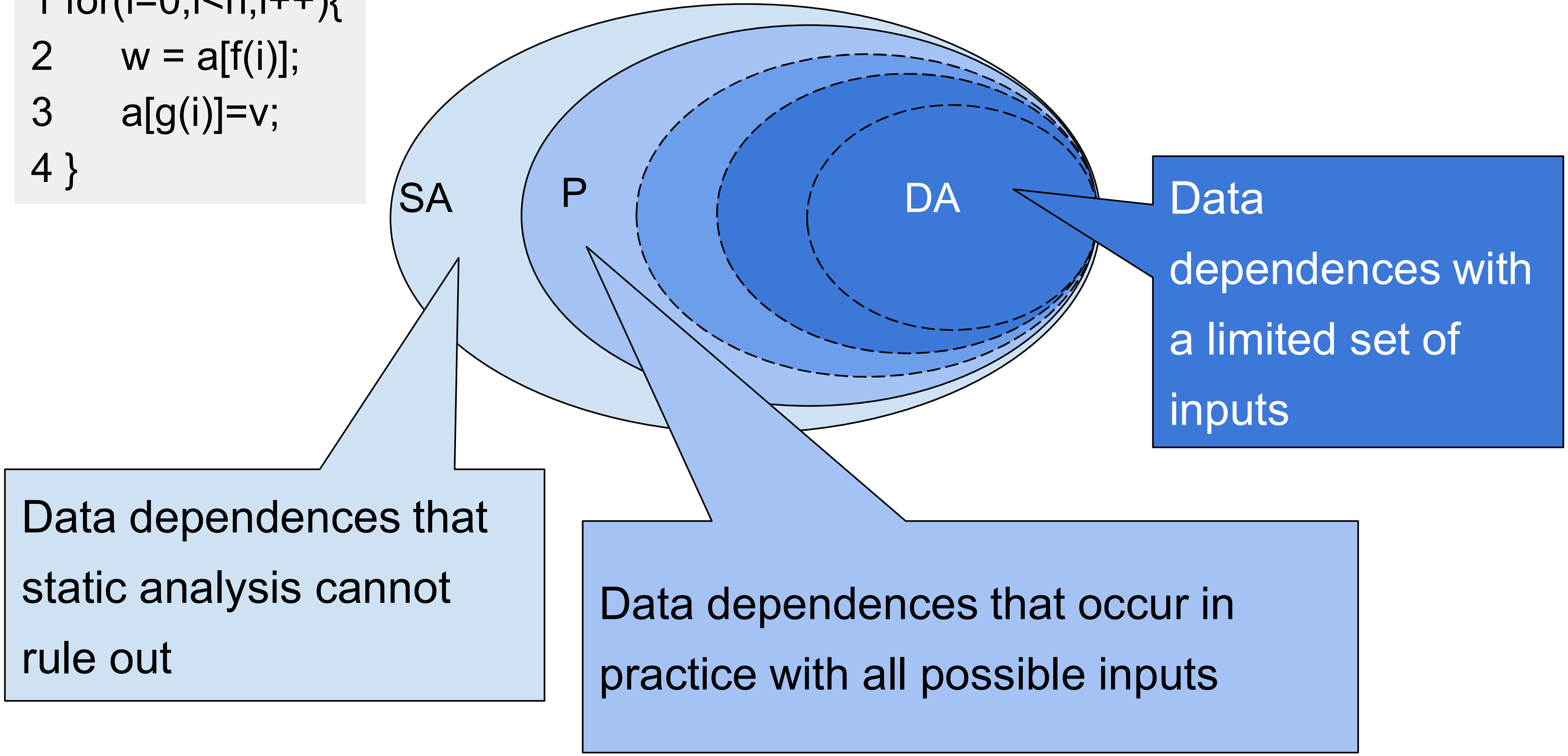
- requires manual tuning of the tools;
- current approach cannot support some common parallelization patterns like stencil

Proposed solution:

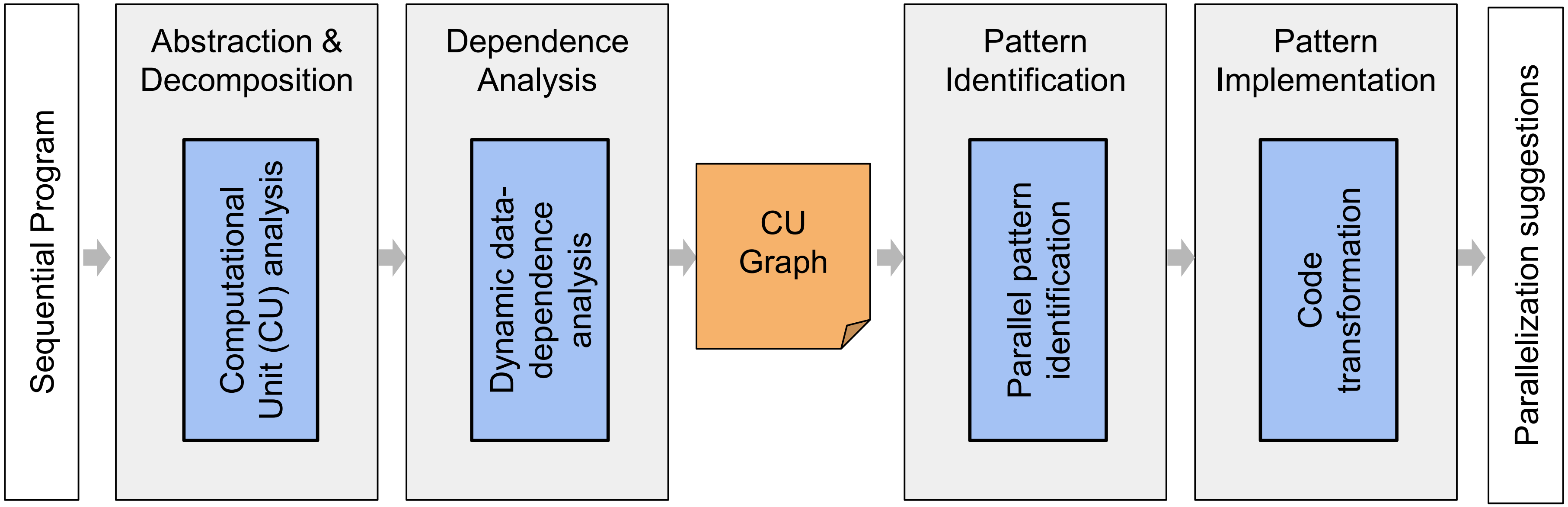
- using machine learning techniques for parallelism discovery

Background - Static vs. Dynamic data dependence analysis

```
for(i=0;i<n;i++){
      w = a[f(i)];
      a[g(i)]=v;
}
```



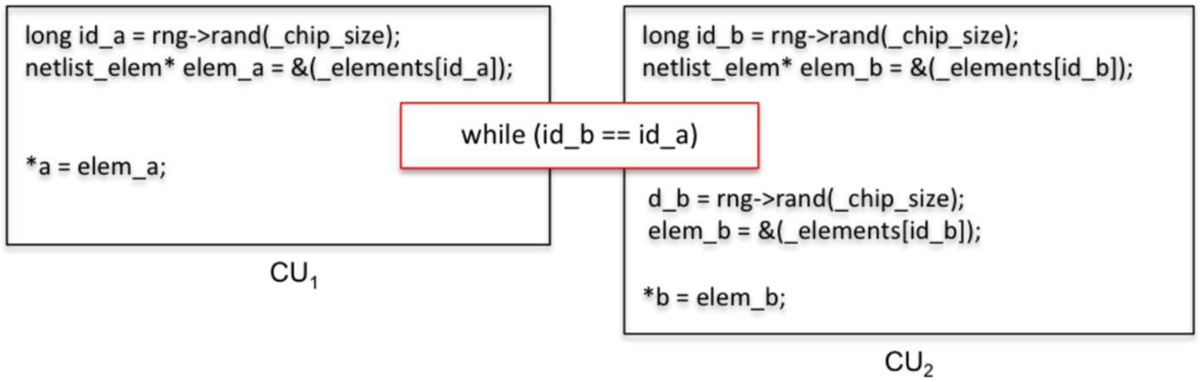
Background - DiscoPoP



https://github.com/discopop-project/discopop

Background - DiscoPoP

Computational Unit (CU):

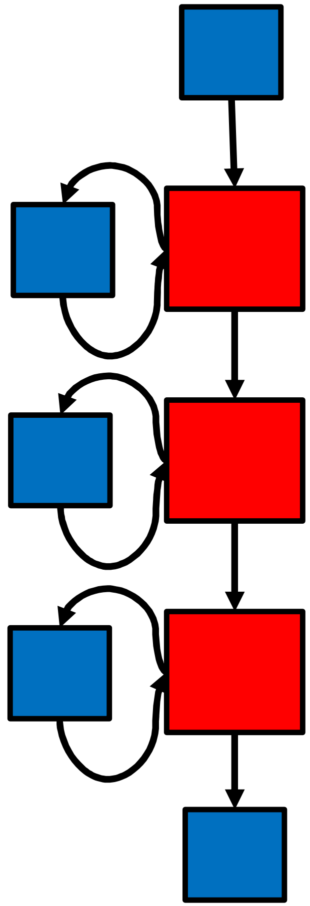



```
void netlist:: (netlist_elem** a, netlist_
{
   //get a random element
   long id_a = rng->rand(_chip_size);
   netlist_elem* elem_a = &(_elements[id_a]);
   //now do the same for b
   long id_b = rng->rand(_chip_size);
   netlist_elem* elem_b = &(_elements[id_b]);
   while (id_b == id_a)
   {
      id_b = rng->rand(_chip_size);
      elem_b = &(_elements[id_b]);
   }
   *a = elem_a;
   *b = elem_b;
   return;
}
```

Function netlist::get random pair of parsec.canneal that contains two CUs.

Background - Parallel design patterns

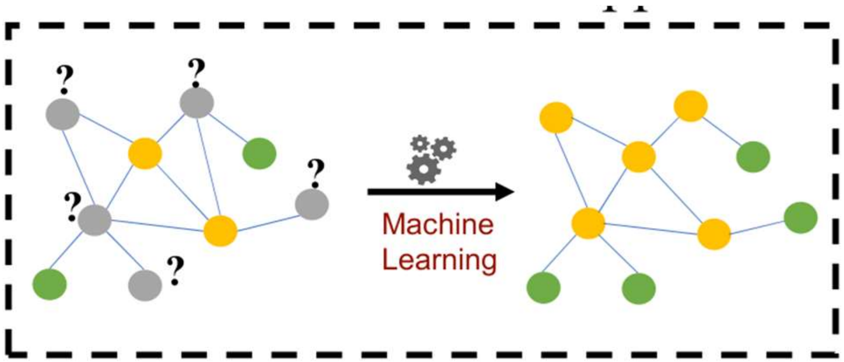
- **Design pattern** = good solution to recurring problem
- Many parallel design patterns have been introduced
  - DOALL, reduction, task parallelism, pipeline, geometric decomposition and many more
- Help avoid concurrency bugs such as deadlocks and data races
- Simplify parallelization process of a sequential code



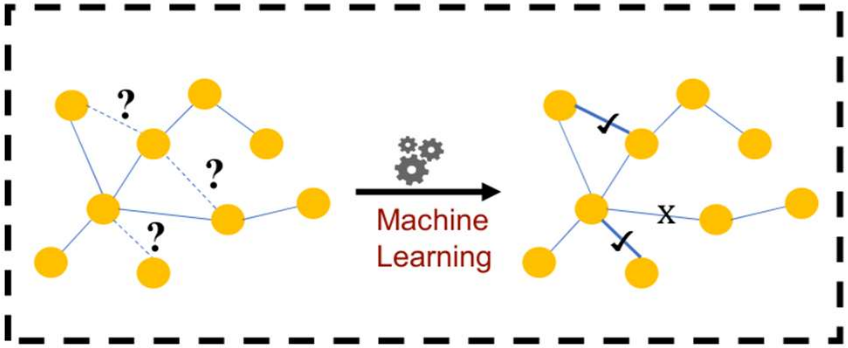
Background - Graph Convolutional Networks (GCNs)

Graph Convolutional Networks (GCNs):

- Main idea: For each layer, information is passed between each other through links, and aggregated by each node.
- Fuse node features with the help of network structures.
- Applications: machine learning tasks in networks



Node Classification



Link Prediction

Background - Graph Convolutional Networks (GCNs)

Why GCNs?

- ML techniques like NLP methods cannot be directly applied
- Codes can be naturally represented by graphs/trees

Background - Parallelism Discovery with Machine Learning

with traditional machine learning techniques:

- Fried's work [1]
- uses DiscoPoP to extract dynamic features
- applies different ML techniques to classify target loops:
    - SVM
    - Decision Tree
    - AdaBoost DT

with GNNs:

Shen's work [2]

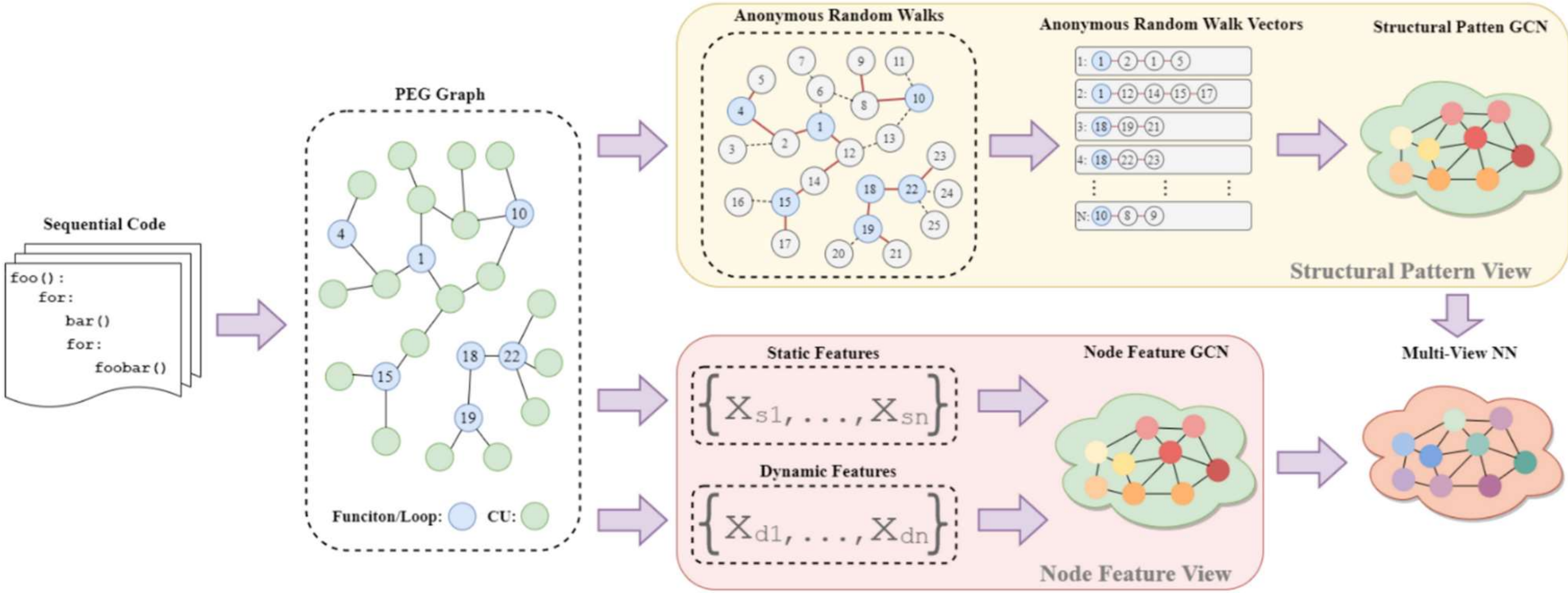
- uses contextual flow graphs to represent the code
- applies a deep graph convolution neural network for graph classification

1. Fried, Daniel, et al. "Predicting parallelization of sequential programs using supervised learning." 2013 12th International Conference on Machine Learning and Applications. Vol. 2. IEEE, 2013.
2. Shen, Yuanyuan, et al. "Towards parallelism detection of sequential programs with graph neural network." Future Generation Computer Systems 125 (2021): 515-525.

Challenges

- code representation
- code embedding
- feature selection
- insufficient data

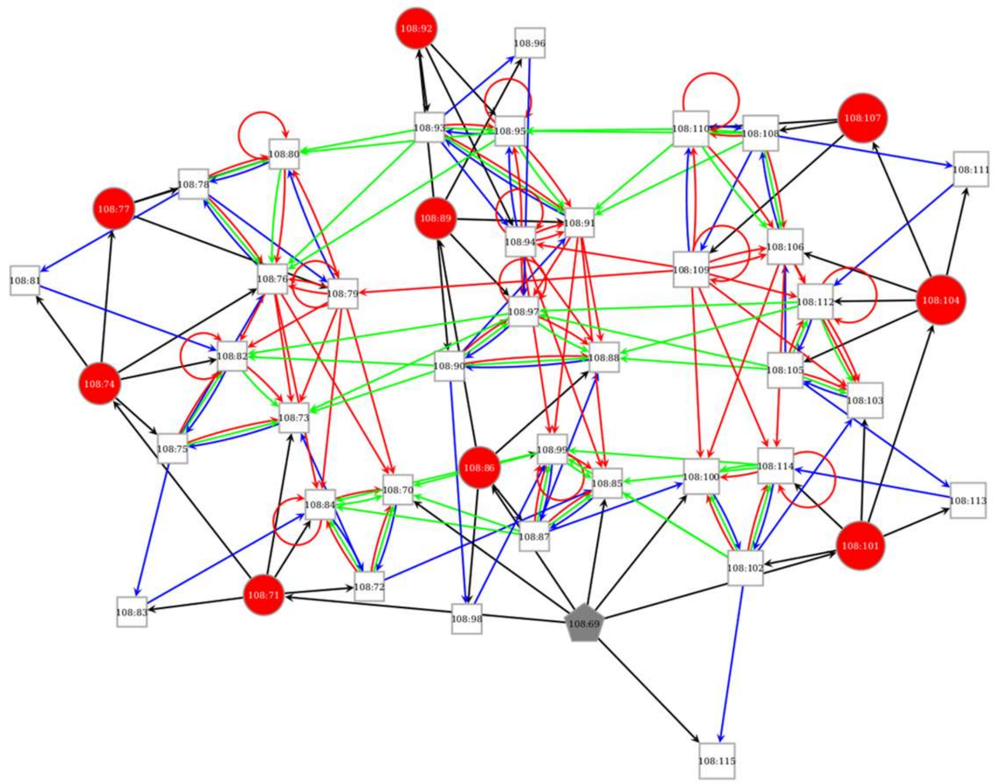
Approach



Sequential Code
foo():
for:
bar()
for:
foobar()
PEG Graph
Funciton/Loop:
CU:
Anonymous Random Walks
Anonymous Random Walk Vectors
Structural Patten GCN
Structural Pattern View
Static Features
Dynamic Features
Node Feature GCN
Node Feature View
Multi-View NN

Approach - Code Representation

PEG: program execution graphs based on the CU graph generated by DiscoPoP



Approach - Feature Selection and Embedding

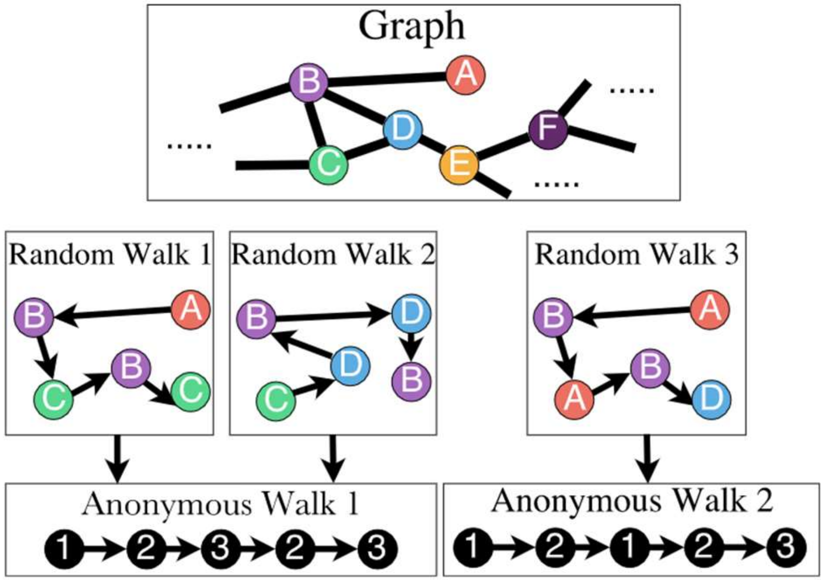
Dynamic features:

Feature name	Description
N Inst	Number of instructions within the loop
exec times	Total number of times the loop is executed
CFL	Critical path length
ESP	Estimated speedup
incoming dep	Dependency count of external instructions on loop instructions
internal dep	Dependency count between loop instructions
outgoing dep	Dependency count of loop instructions on external instructions

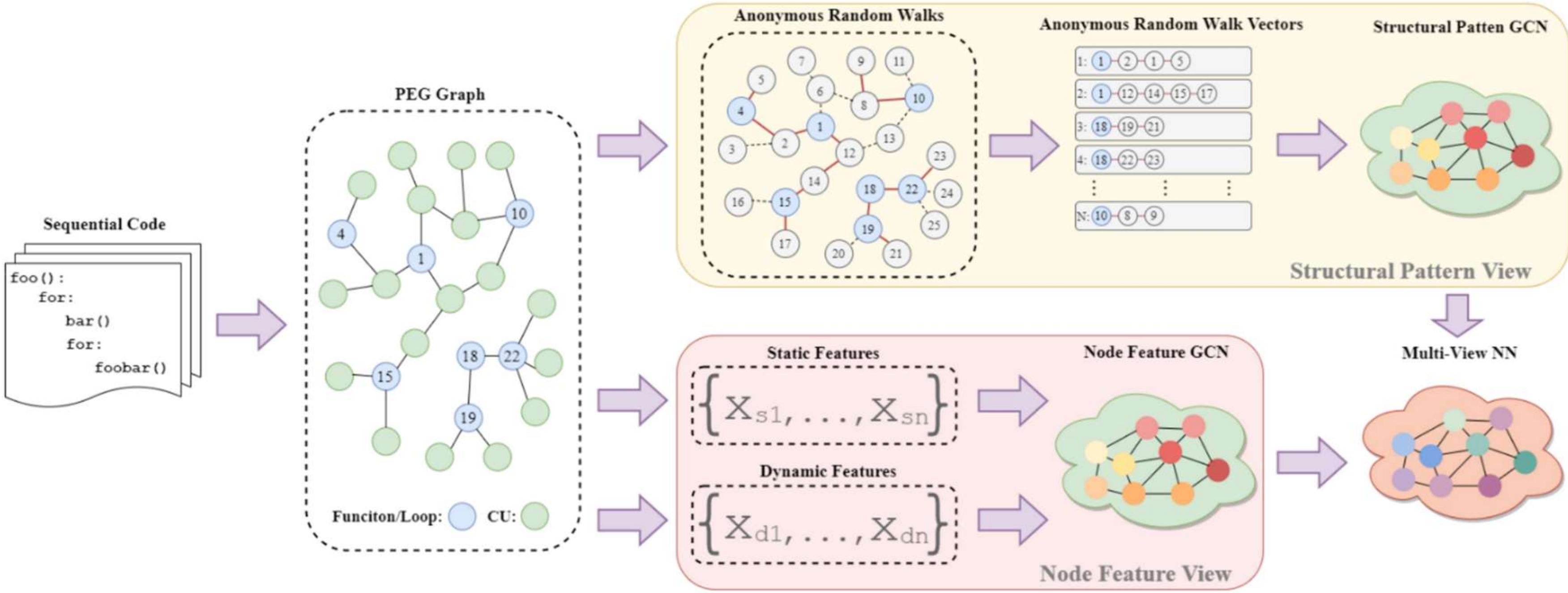
Approach - Feature Selection and Embedding

Static features:
representation of code semantics with Ben-Nun's work

Structural features:
embedding with anonymous walks:



Approach - A Multi-view GNN Approach



Approach - Dataset

Benchmarks:

- NAS
- BOTs
- PolyBench
- Synthetic dataset

Evaluation

Benchmark	Model	Acc (%)
NPB	Multi-view	86.4
	CNN+LSTM	76.1
PolyBench	Multi-view	82.1
	CNN+LSTM	74.5
BOTS	Multi-view	81.9
	CNN+LSTM	71.4

Conclusion

- We propose a GNN based framework towards the discovery of parallelism in sequential programs
- It achieves comparable results comparing with traditional methods
- Our framework can be used as a foundation for downstream tasks like pattern recognition.
- Our work is limited by insufficient data. In future work, we plan to solve this problem by
  - creating synthetic dataset, and
  - applying ML techniques that works with limited training data.

Future work

- synthetic dataset generation
- structural information embedding for code
- better dynamic representation and embedding for code
- apply our model with CnC graph
- open for collaboration :)

Q & A

Thank you!

Motivation

Traditional methods for parallelism discovery:

- static analysis: is unable to detect any run-time features and is proved overly conservative for identifying parallelism in general-purpose programs
- dynamic analysis: Optimistic approach based on actual (dynamic) dependences has shown to allow reproduction of manual parallelization strategies

However:

- requires manual tuning of the tools;
- current approach cannot support some common parallelization patterns like stencil