

# 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

# Introduction

## Parallelizing sequential programs is not so easy.

```
1497
1498 static inline void CalcForceForNodes(Domain& domain)
1499 {
1500     Index_t numNode = domain.numNode();
1501
1502     #if USE_MPI
1503     CommRecv(domain, MSG_COMM_SBN, 3,
1504              domain.sizeX() + 1, domain.sizeY() + 1, domain.sizeZ() + 1,
1505              true, false);
1506     #endif
1507
1508     #pragma omp parallel for firstprivate(numNode)
1509     for (Index_t i=0; i<numNode; ++i) {
1510         domain.fx(i) = Real_t(0.0);
1511         domain.fy(i) = Real_t(0.0);
1512         domain.fz(i) = Real_t(0.0);
1513     }
1514
1515     /* Calc force calls partial_force_huang */
1516     CalcVolumeForceForElem(domain);
1517
1518     #if USE_MPI
1519     Domain member fieldData();
1520     fieldData[0] = &domain; fF;
1521     fieldData[1] = &domain; fY;
1522     fieldData[2] = &domain; fZ;
1523
1524     CommSend(domain, MSG_COMM_SBN, 3, fieldData,
1525              domain.sizeX() + 1, domain.sizeY() + 1, domain.sizeZ() + 1,
1526              true, false);
1527     CommSbn(domain, 3, fieldData);
1528     #endif
1529
1530     //.....
1531
1532     static inline
1533     void CalcAccelerationForNodes(Domain &domain, Index_t numNode)
1534     {
1535         #pragma omp parallel for firstprivate(numNode)
1536         for (Index_t i = 0; i < numNode; ++i) {
1537             domain.xdd(i) = domain.fx(i) / domain.nodalMass(i);
1538             domain.ydd(i) = domain.fy(i) / domain.nodalMass(i);
1539             domain.zdd(i) = domain.fz(i) / domain.nodalMass(i);
1540         }
1541     }
1542
1543     //.....
1544
1545     static inline
1546     void ApplyAccelerationBoundaryConditionsForNodes(Domain& domain)
1547     {
1548         Index_t size = domain.sizeX();
1549         Index_t numNodeBC = (size-1)/size;
1550
1551         #pragma omp parallel
1552         {
1553             if (!domain.symZempty()) {
1554                 #pragma omp for nowait firstprivate(numNodeBC)
1555                 for (Index_t i=0; i<numNodeBC; ++i)
1556                     domain.xdd(domain.symZ(i)) = Real_t(0.0);
1557             }
1558             if (!domain.symYempty()) {
1559                 #pragma omp for nowait firstprivate(numNodeBC)
1560                 for (Index_t i=0; i<numNodeBC; ++i)
1561                     domain.ydd(domain.symY(i)) = Real_t(0.0);
1562             }
1563             if (!domain.symXempty()) {
1564                 #pragma omp for nowait firstprivate(numNodeBC)
1565                 for (Index_t i=0; i<numNodeBC; ++i)
1566                     domain.zdd(domain.symX(i)) = Real_t(0.0);
1567             }
1568         }
1569
1570     //.....
1571
1572     static inline
1573     void CalcVelocityForNodes(Domain &domain, const Real_t dt, const Real_t u_cut,
1574                               Index_t numNode)
1575     {
1576         #pragma omp parallel for firstprivate(numNode)
1577         for (Index_t i = 0; i < numNode; ++i)
1578         {
1579             Real_t xstmp, ystmp, zstmp;
1580
1581             xstmp = domain.x(i) + domain.xdd(i) * dt;
1582             if (fabs(xstmp) < u_cut) xstmp = Real_t(0.0);
1583             domain.x(i) = xstmp;
1584
1585             ystmp = domain.y(i) + domain.ydd(i) * dt;
1586             if (fabs(ystmp) < u_cut) ystmp = Real_t(0.0);
1587             domain.y(i) = ystmp;
1588
1589             zstmp = domain.z(i) + domain.zdd(i) * dt;
1590             if (fabs(zstmp) < u_cut) zstmp = Real_t(0.0);
1591             domain.z(i) = zstmp;
1592         }
1593     }
1594 }
```

#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](http://www.discopop.org) , <https://github.com/discopop-project/discopop>
- AutopaR
- etc



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

IOWA STATE  
UNIVERSITY

## Motivation

---

However:

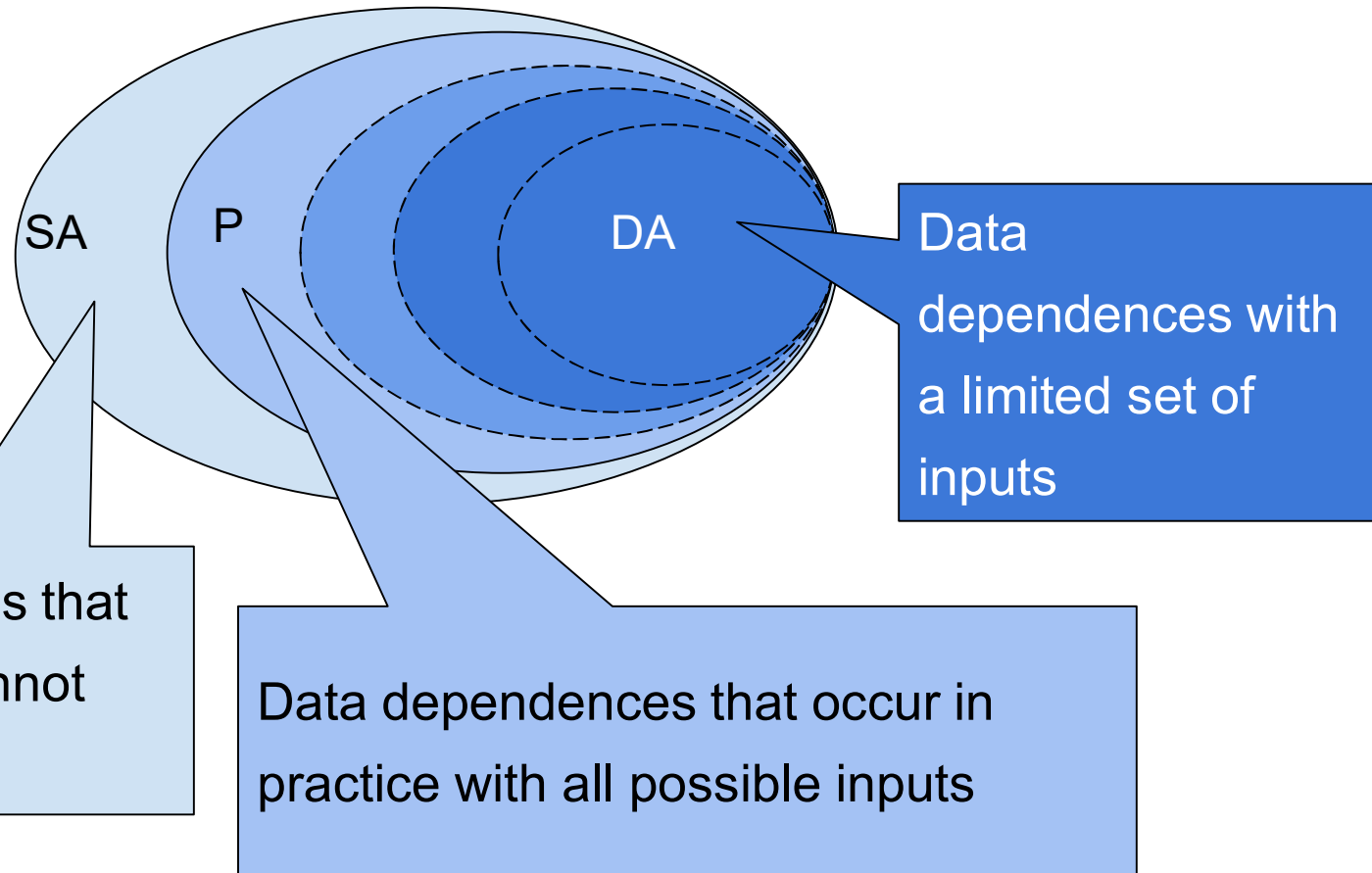
- requires manual tuning of the tools;
- current approach cannot support some common parallelization patterns like stencil

Proposed solution:

- using machine learning techniques for parallelism discovery

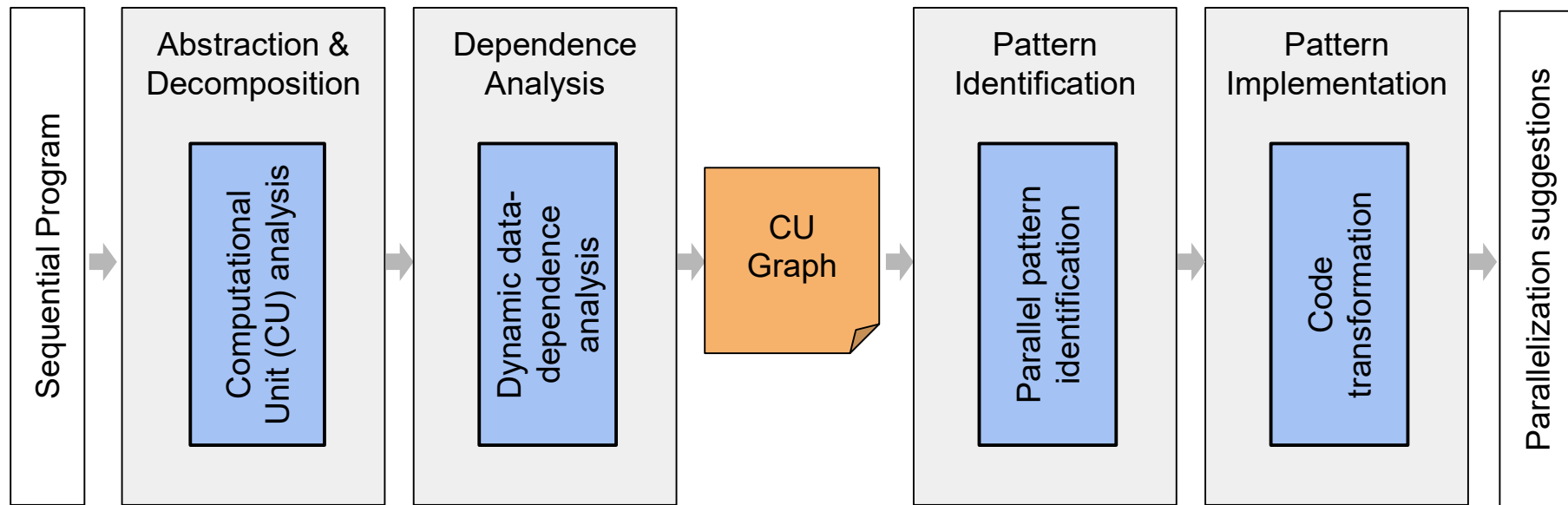
# Background - Static vs. Dynamic data dependence analysis

```
1 for(i=0;i<n;i++){  
2   w = a[f(i)];  
3   a[g(i)]=v;  
4 }
```



# Background - DiscoPop

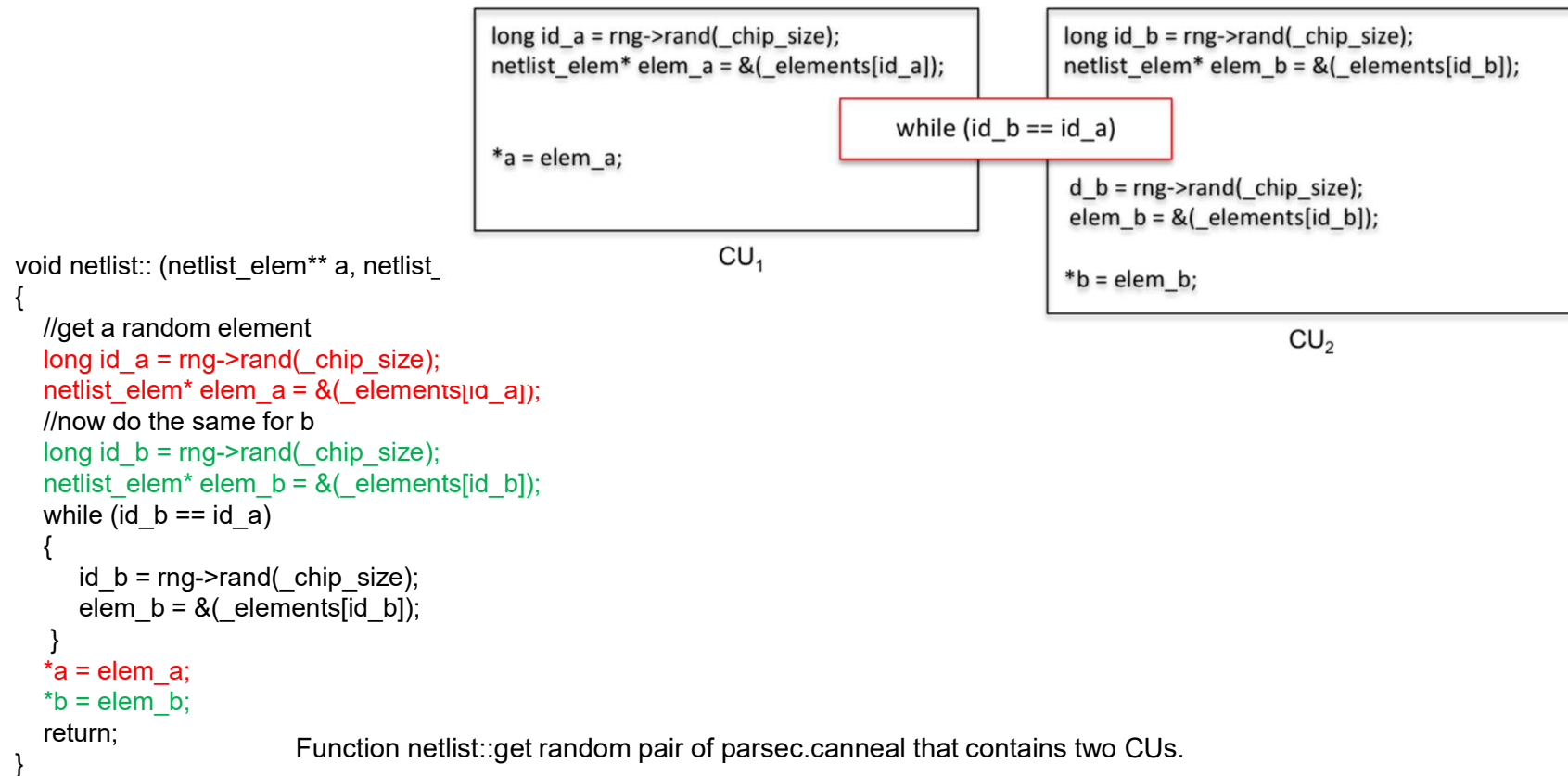
---



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

# Background - DiscoPoP

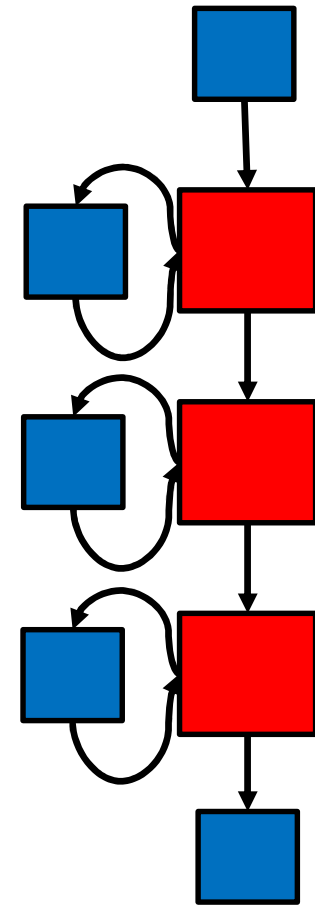
## Computational Unit (CU):



## 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

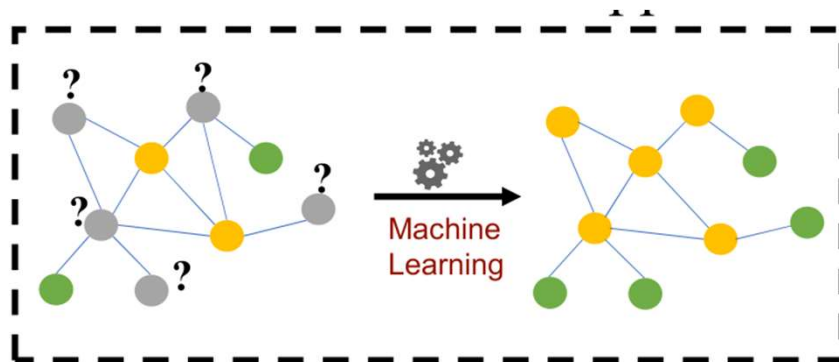




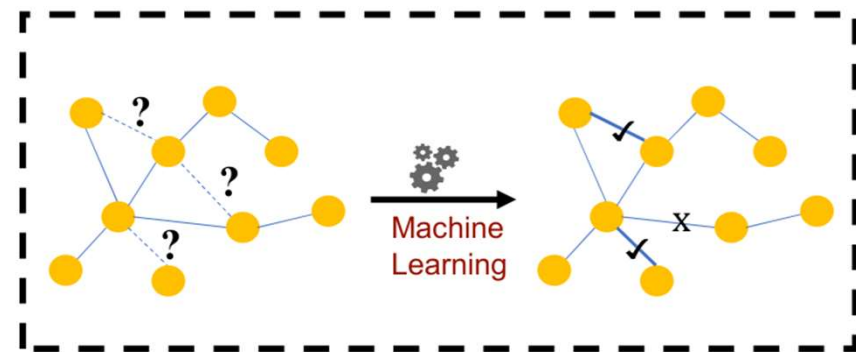
# Background - Graph Convolutional Networks (GCNs)

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

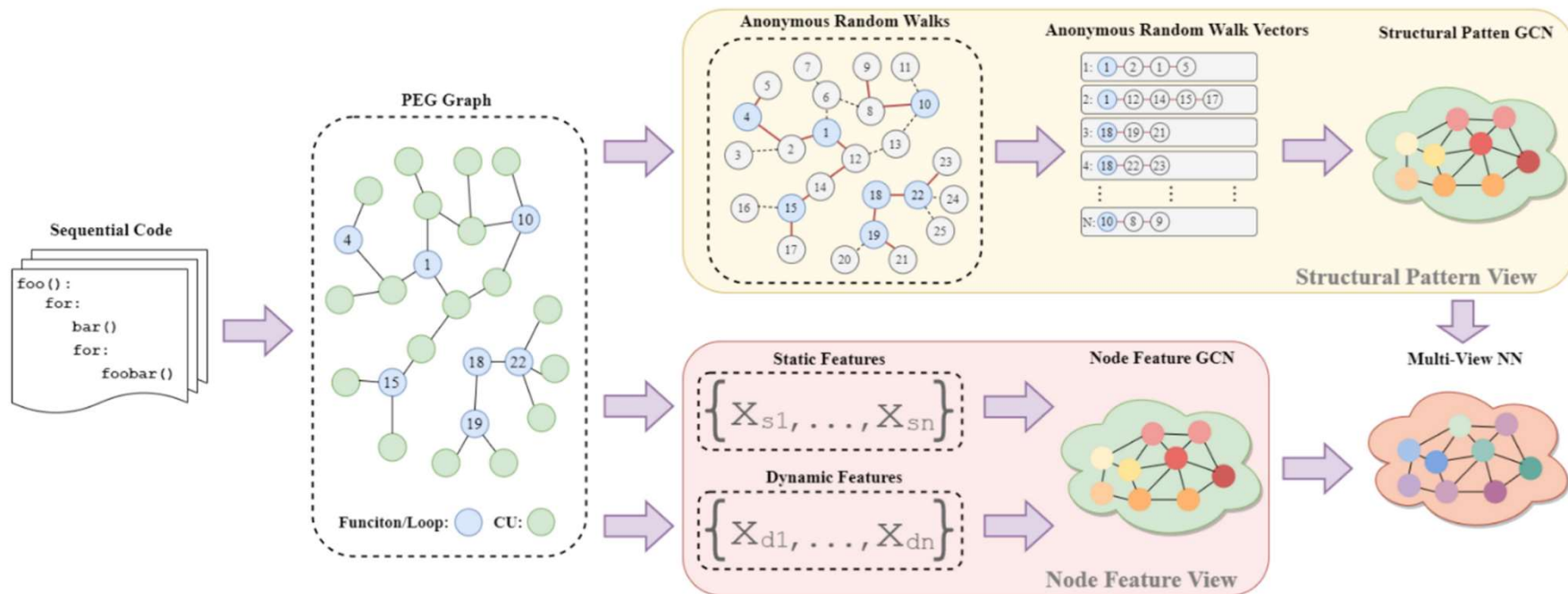
1. Fried, Daniel, et al. "Predicting parallelization of sequential programs using supervised learning." 2013 12th International Conference on Machine Learning and Applications. Vol. 2. IEEE, 2013.
2. Shen, Yuanyuan, et al. "Towards parallelism detection of sequential programs with graph neural network."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

# Challenges

---

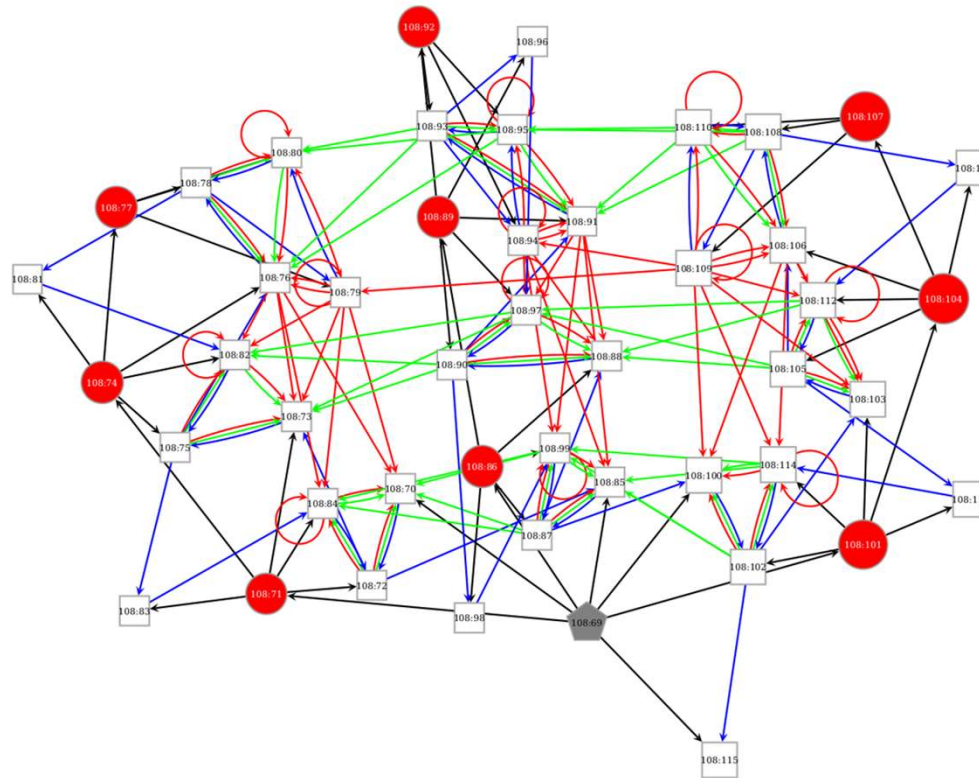
- code representation
- code embedding
- feature selection
- insufficient data

# Approach



# 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

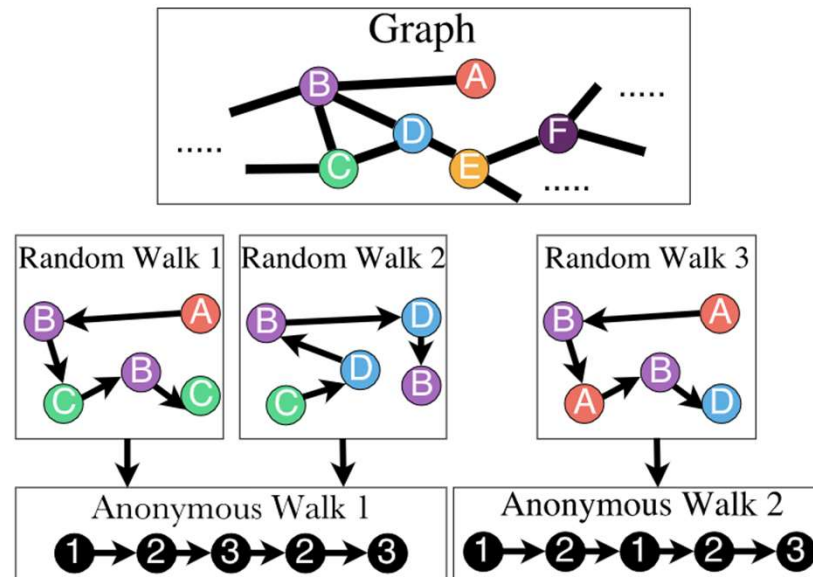
# Approach - Feature Selection and Embedding

Static features:

representation of code semantics with Ben-Nun's work

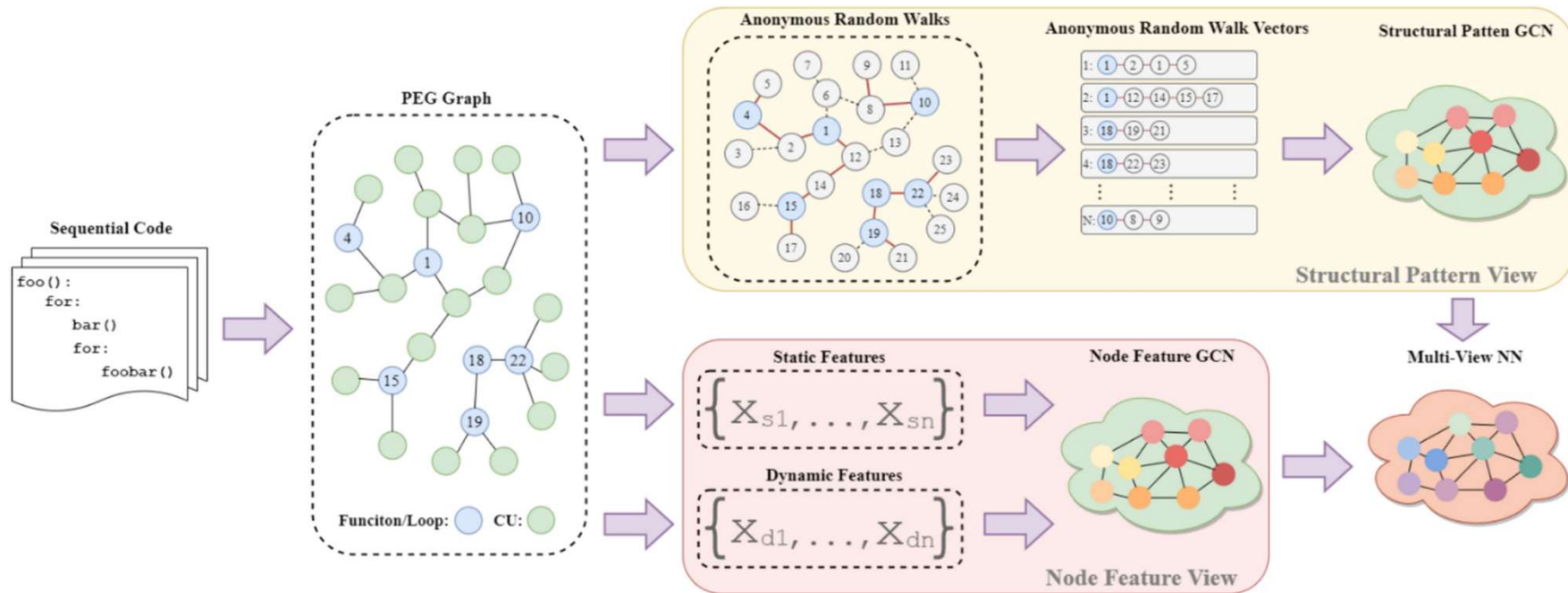
Structural features:

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