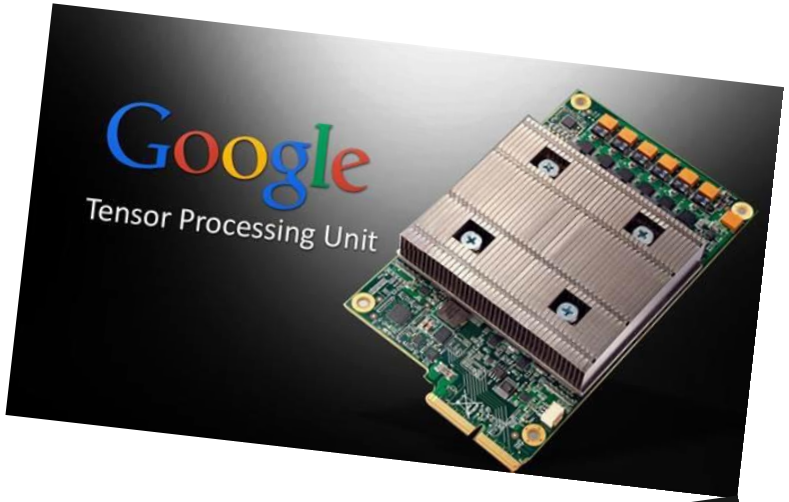
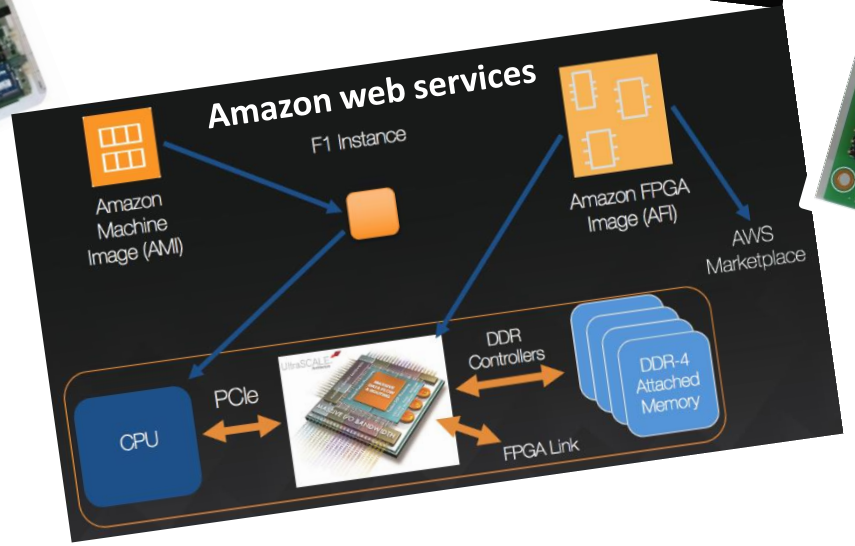
Estimating Circuit Quality Directly from HLS Source Code

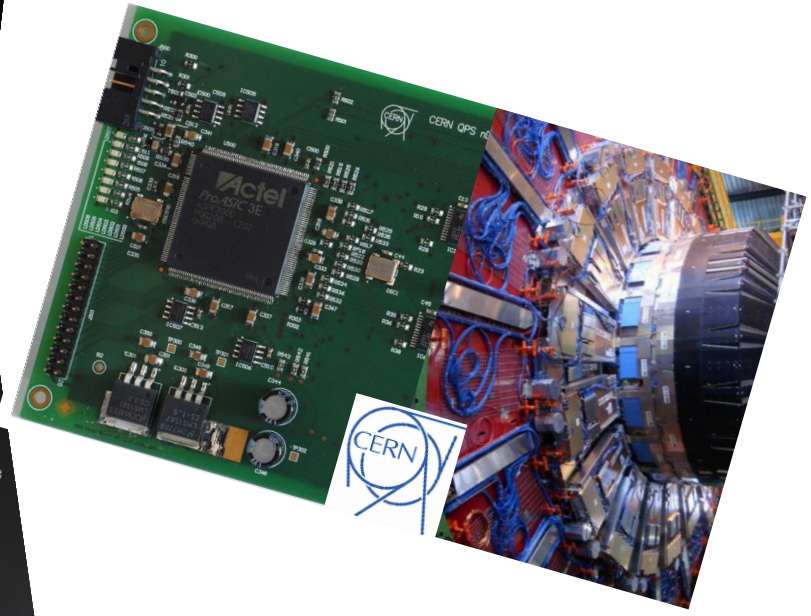
Lana Josipović

June 2025

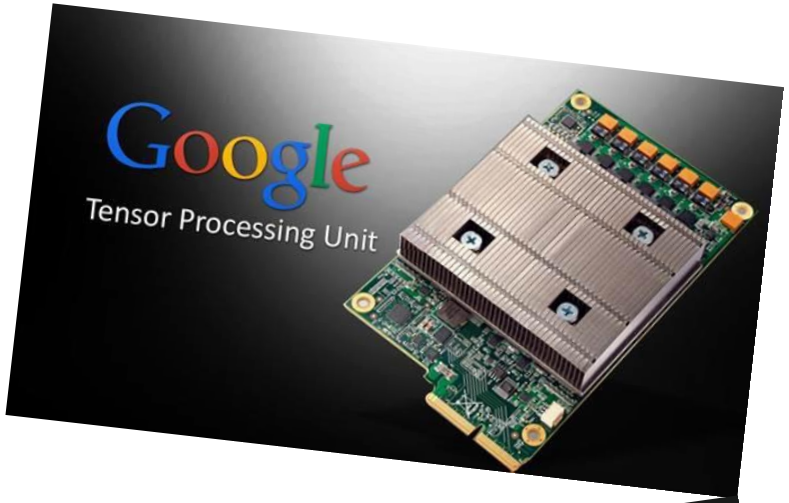
ETH zürich

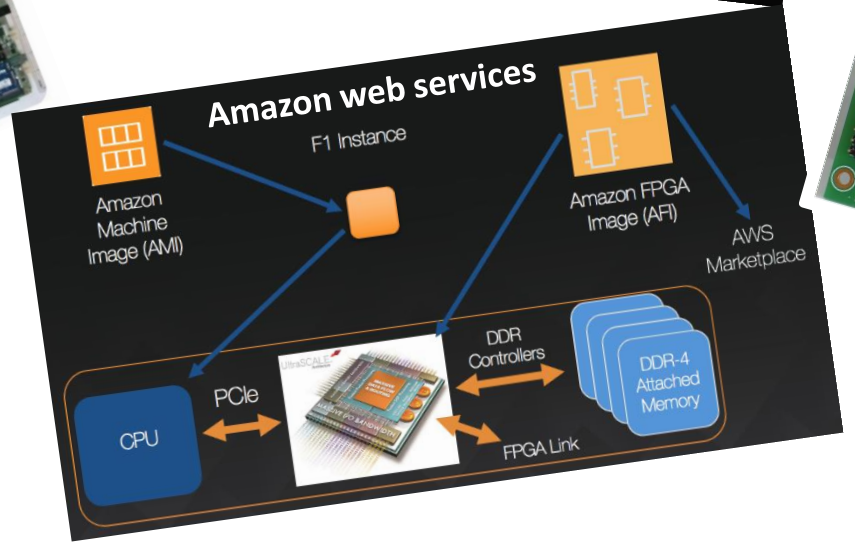


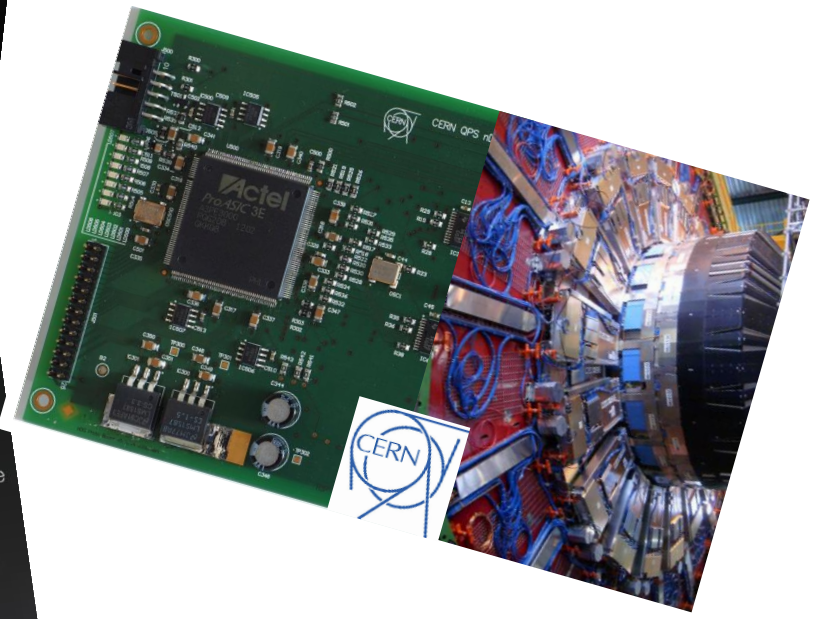




Hardware acceleration for high parallelism and energy efficiency







How to perform hardware design?

*… circuit design is often considered a **"black art",** restricted to only those with years of training in electrical engineering…*
[cacm.acm.org/magazines/2023/1/]

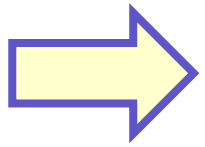
*… chips take **years to design**, resulting in the need to **speculate about how to optimize** the next generation of chips…*
[ai.googleblog.com/2020/04]

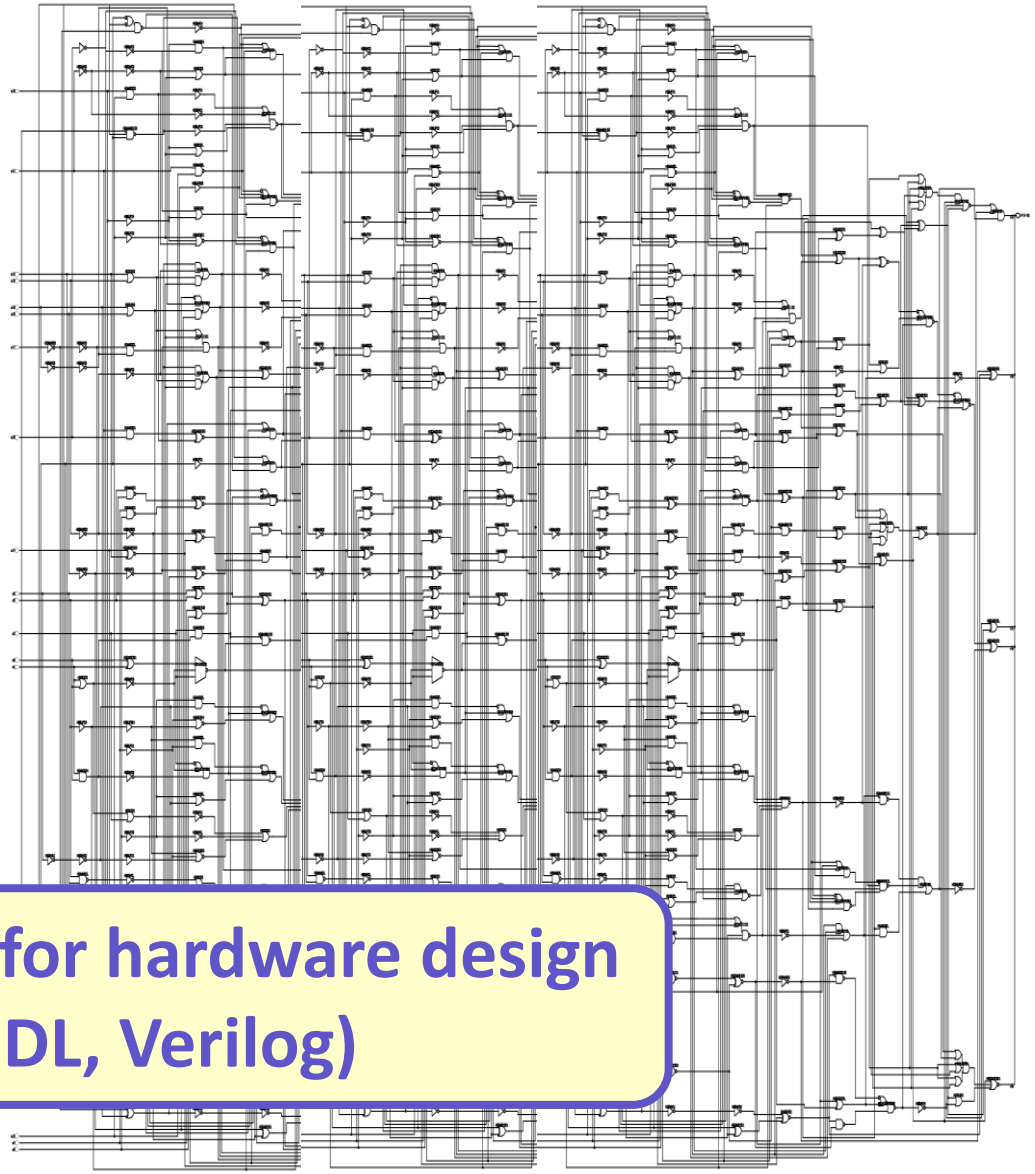
High-Level Synthesis: From Programs to Circuits

```
#define PI 3.14159265358979323846264338

complex* DFT_naive(complex* x, int N) {
  complex* X = (complex*) malloc(sizeof(struct complex_t) * N);
  int k, n;
  for(k = 0; k < N; k++) {
    X[k].re = 0.0;
    X[k].im = 0.0;
    for(n = 0; n < N; n++) {
      X[k] = add(X[k], multiply(x[n],
                                conv_from_polar(1,
                                                -2*PI*n*k/N)));
    }
  }

  return X;
}
```





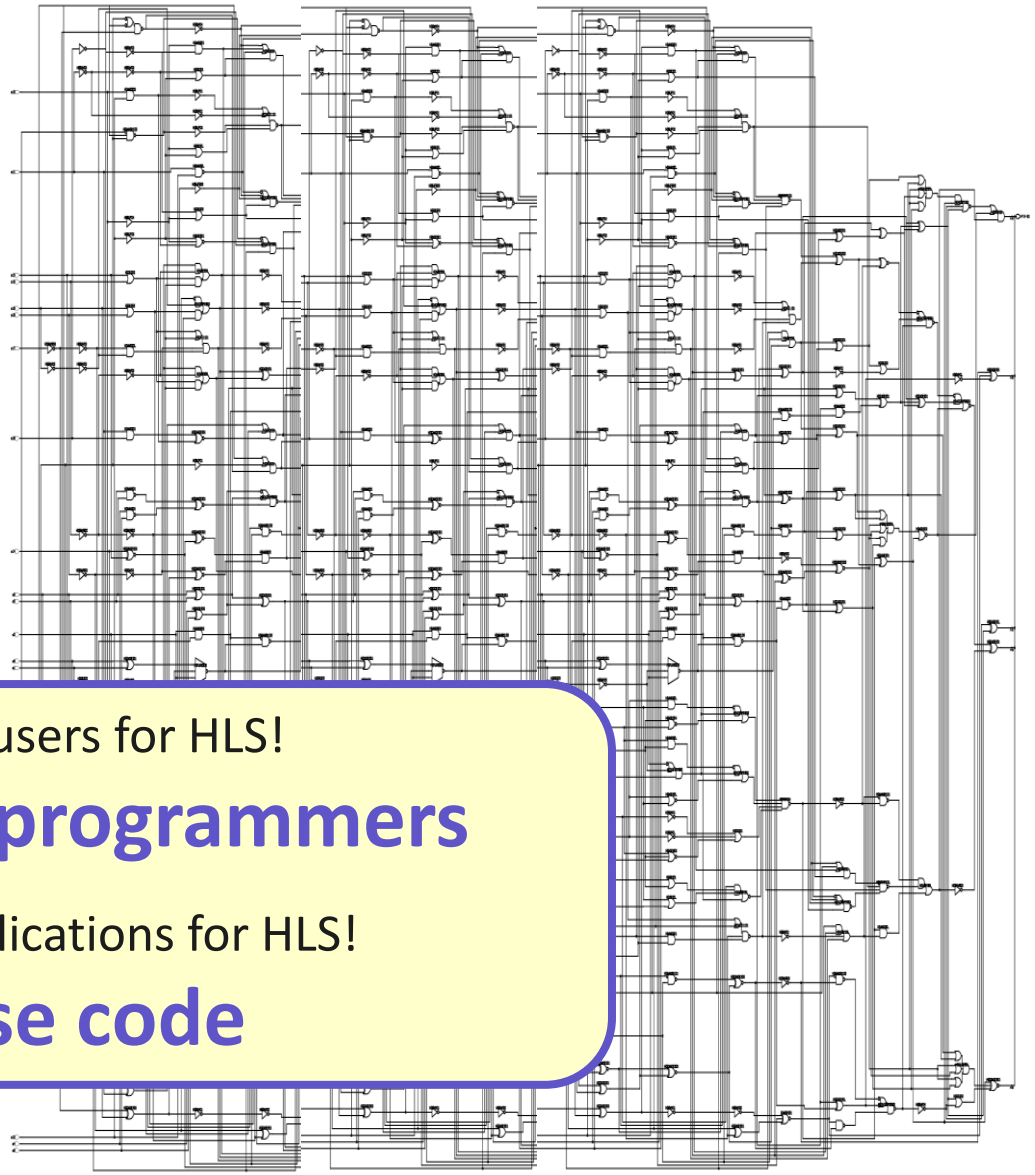
Raise the level of abstraction for hardware design beyond RTL level (VHDL, Verilog)

High-Level Synthesis: From Programs to Circuits

```
#define PI 3.14159265358979323846264338

complex* DFT_naive(complex* x, int N) {
  complex* X = (complex*) malloc(sizeof(struct complex_t) * N);
  int k, n;
  for(k = 0; k < N; k++) {
    X[k].re = 0.0;
    X[k].im = 0.0;
    for(n = 0; n < N; n++) {
      X[k] = add(X[k], multiply(x[n],
                                conv_from_polar(1,
                                                -2*PI*n*k/N)));
    }
  }

  return X;
}
```



A completely new type of users for HLS!

Software application programmers

A completely new type of applications for HLS!

General-purpose code

Bridging the Gap Between Software and Hardware

SW

HW

How to make hardware design accessible to non-experts?

High-level synthesis (HLS): from software descriptions to hardware design

```
1  void(int* mem){
2    mem[512] = 0;
3    for(int i=0; i<512; i++)
4       mem[512] += mem[i];
5  }
```

Unoptimized software program, execution time = 27,236 clock cycles

```
1  // Width of MPort = 16 * sizeof(int)
2  #define ChunkSize (sizeof(MPort)/sizeof(int))
3  #define LoopCount (512/ChunkSize)
4  // Maximize data width from memory
5  void(MPort* mem){
6     // Use a local buffer and burst access
7     MPort buff[LoopCount];
8     memcpy(buff, mem, LoopCount);
9     // Use a local variable for accumulation
10    int sum=0;
11    for(int i=1; i<LoopCount; i++){
12    // Use additional directives where useful
13    // e.g. pipeline and unroll for parallel exec.
14    #pragma PIPELINE
15       for(int j=0; j<ChunkSize; j++){
16       #pragma UNROLL
17          sum+=(int)(buff[i]>>j*sizeof(int)*8);}}
18     mem[512]=sum;
19 }
```

Optimized software program, execution time = 302 clock cycles

George et al. FPL 2014.

Bridging the Gap Between Software and Hardware

SW

How to make hardware design accessible to non-experts?

How to automatically extract parallelism from software code?

HW

Typical software features prevent **hardware parallelism**

```
for (i = 0; i < num_rows, i++) {
  tmp = 0;
  s = row[i]; e = row[i+1];
  for (c = s; c < e; c++) {
    cid = col[c];
    tmp += val[c] * vec[cid];
  }

  out[i] = tmp;
}
```

Variable memory latency

Variable loop bounds

Irregular memory access patterns

Sparse-matrix dense-vector multiplication (SpMV)

Bridging the Gap Between Software and Hardware

SW

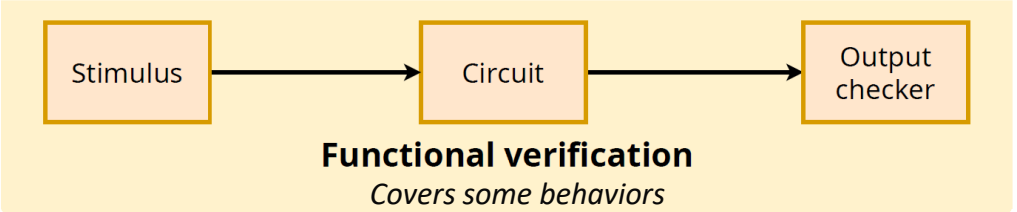
How to make hardware design accessible to non-experts?

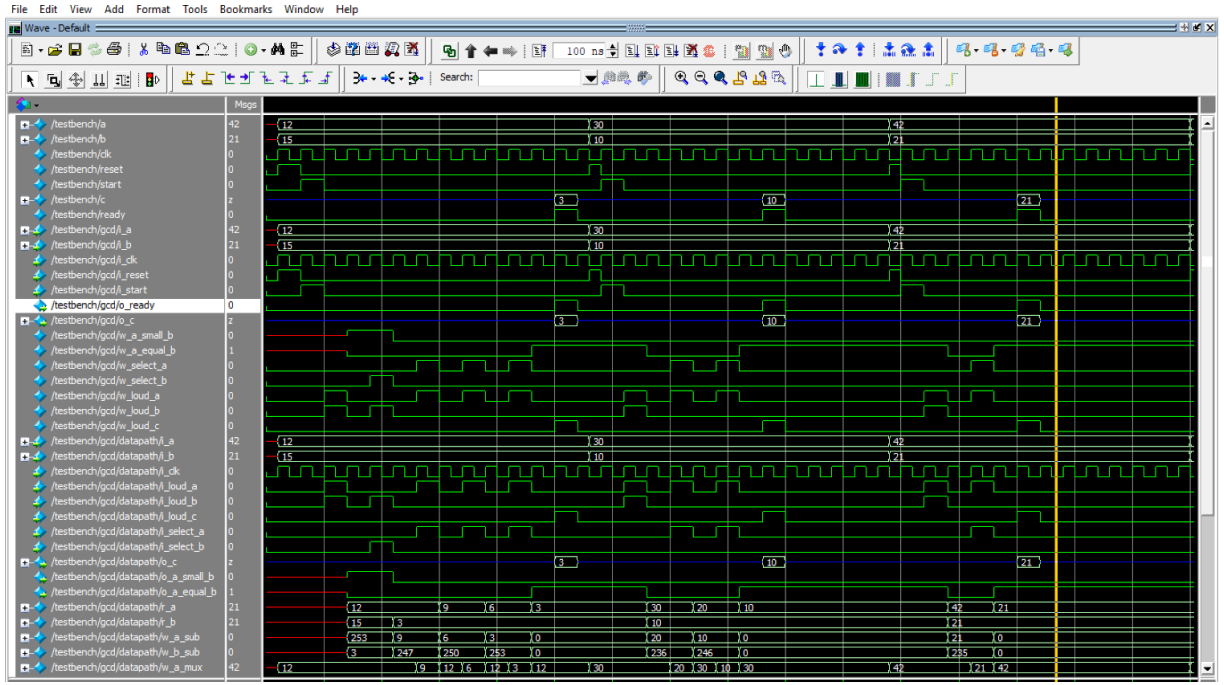
How to automatically extract parallelism from software code?

How to avoid hardware-level functional verification?

HW

Functional verification of circuits using hardware simulation
→ inefficient, limited, non-exhaustive





Bridging the Gap Between Software and Hardware

SW

How to make hardware design accessible to non-experts?

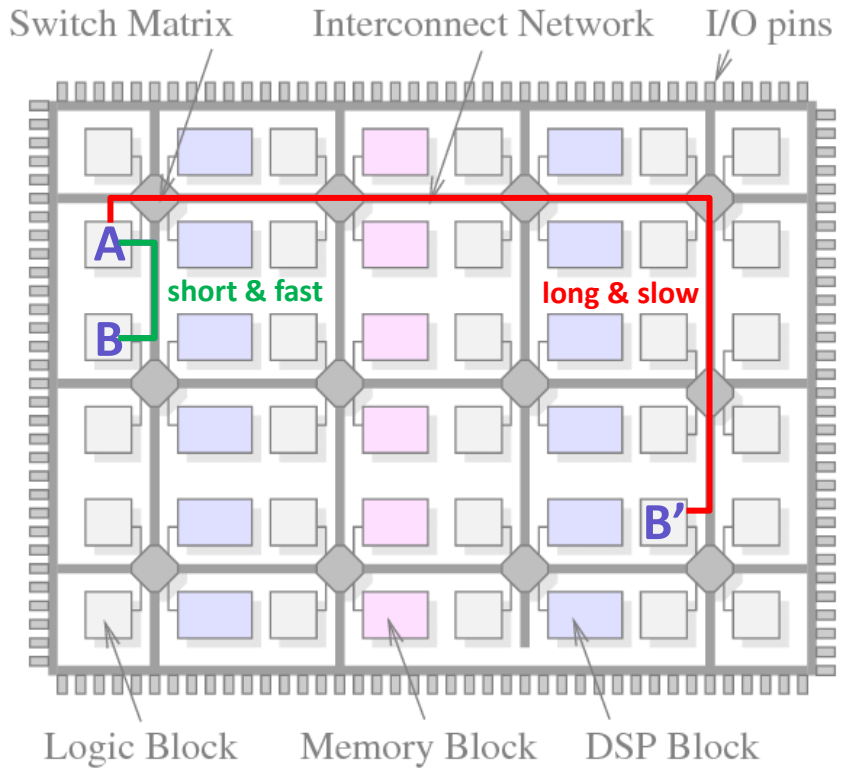
How to automatically extract parallelism from software code?

How to avoid hardware-level functional verification?

How to understand and account for hardware implementation details?

HW

FPGA technology mapping, placement, and routing
→ impact on circuit **performance and power**



A → B

Langhammer et al. ARITH 2015.

Bridging the Gap Between Software and Hardware

SW

HW

How to make hardware design accessible to non-experts?

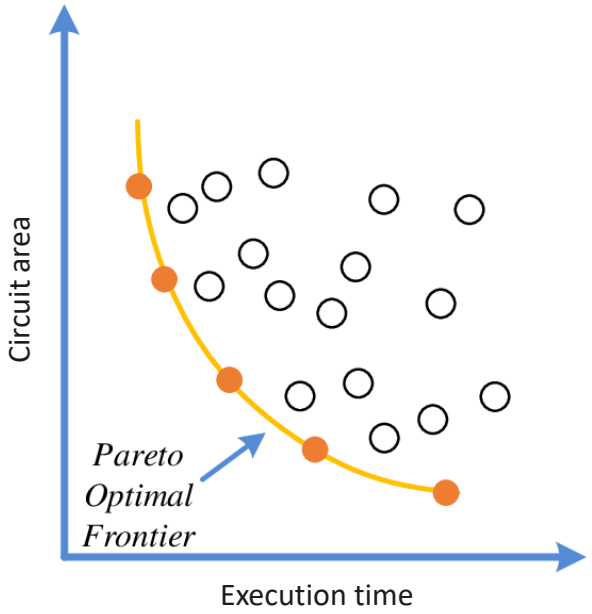
How to automatically extract parallelism from software code?

How to avoid hardware-level functional verification?

How to understand and account for hardware implementation details?

How to perform efficient HLS design space exploration?

Design Space Exploration for HLS



Large design space
(e.g., this kernel → hundreds of points)

How to identify Pareto-optimal code configurations?

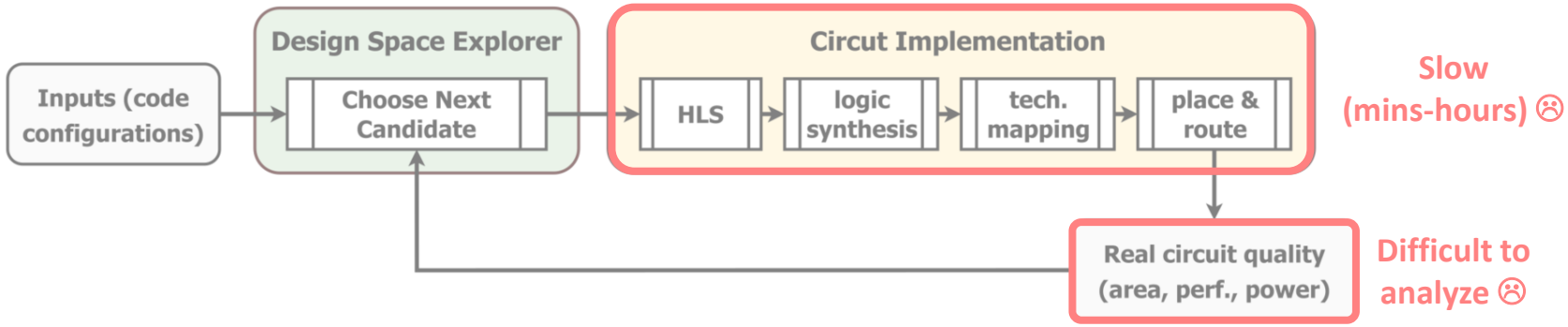
```
1  void(int* mem){
2    mem[512] = 0;
3    for(int i=0; i<512; i++)
4       mem[512] += mem[i];
5  }
```

execution time = 27,236 clock cycles

```
1  // Width of MPort = 16 * sizeof(int)
2  #define ChunkSize (sizeof(MPort)/sizeof(int))
3  #define LoopCount (512/ChunkSize)
4  // Maximize data width from memory
5  void(MPort* mem){
6     // Use a local buffer and burst access
7     MPort buff[LoopCount];
8     memcpy(buff, mem, LoopCount);
9     // Use a local variable for accumulation
10    int sum=0;
11    for(int i=1; i<LoopCount; i++){
12    // Use additional directives where useful
13    // e.g. pipeline and unroll for parallel exec.
14    #pragma PIPELINE
15       for(int j=0; j<ChunkSize; j++){
16       #pragma UNROLL
17          sum+=(int)(buff[i]>>j*sizeof(int)*8);}}
18     mem[512]=sum;
19 }
```

execution time = 302 clock cycles

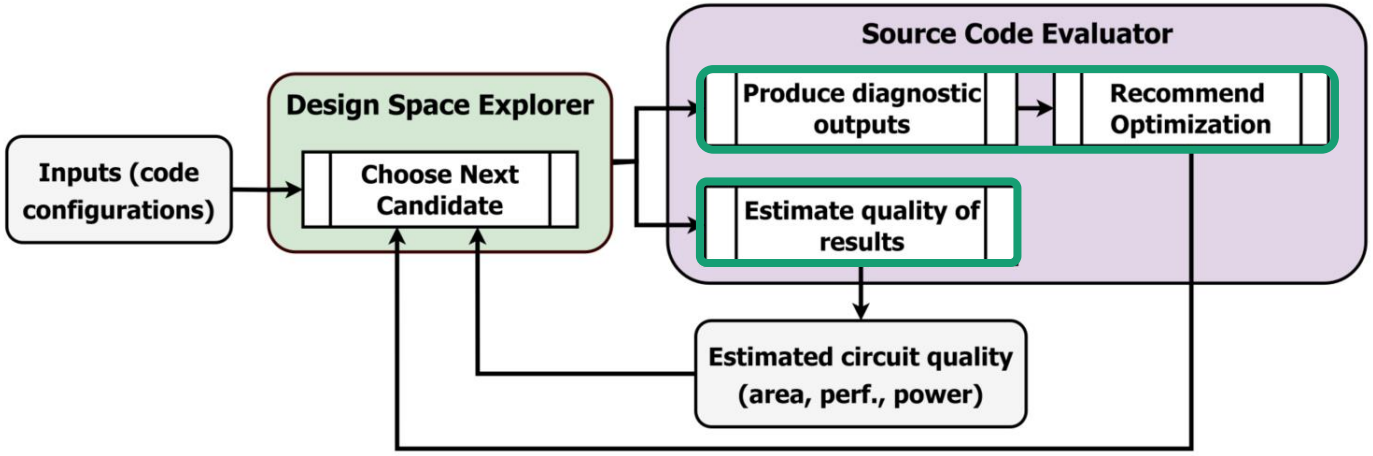
Design Space Exploration for HLS



Slow (mins-hours) ☹

Difficult to analyze ☹

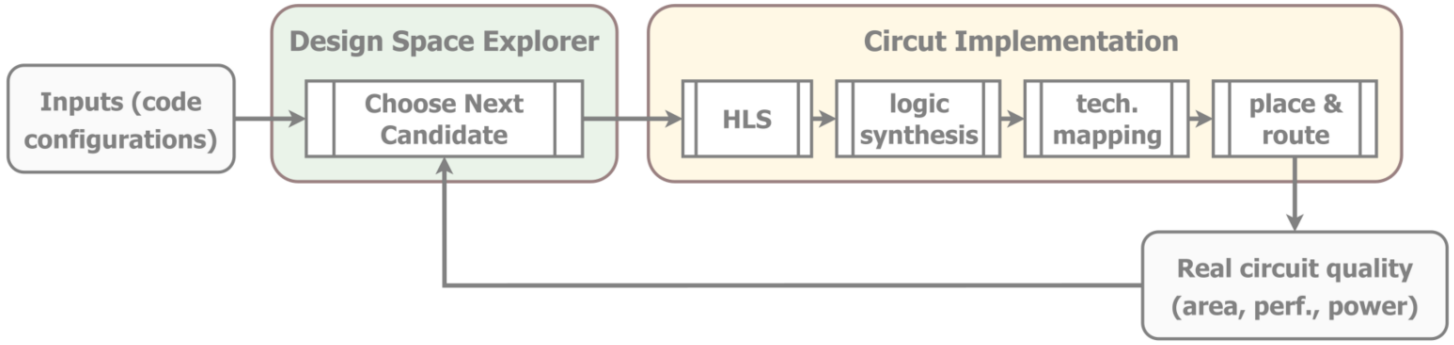
Circuit implementation is too expensive to use in DSE

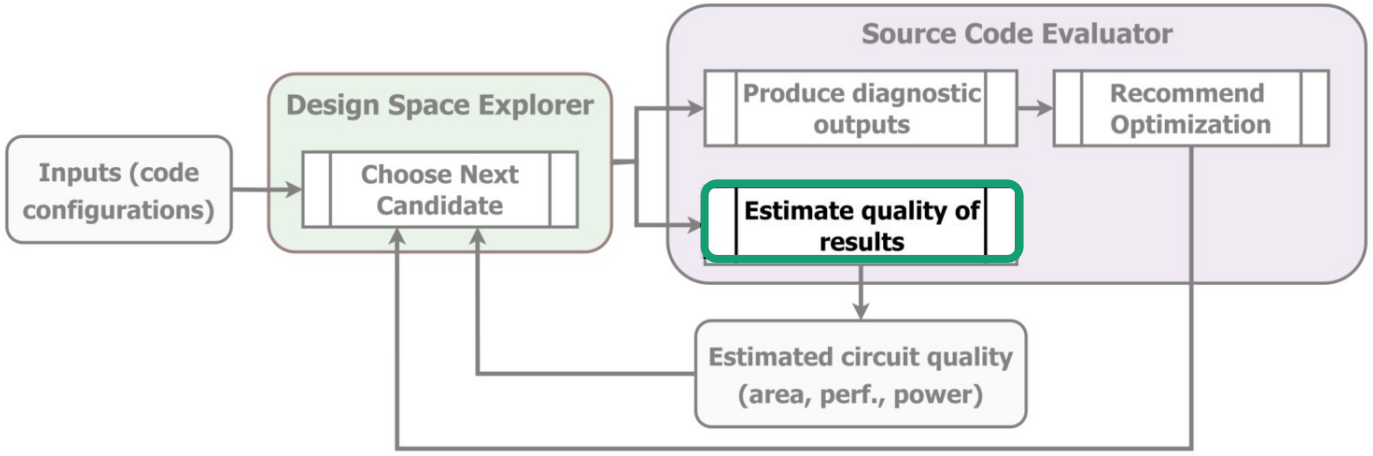


Easy to analyze ☺

Fast (ms) and accurate ☺

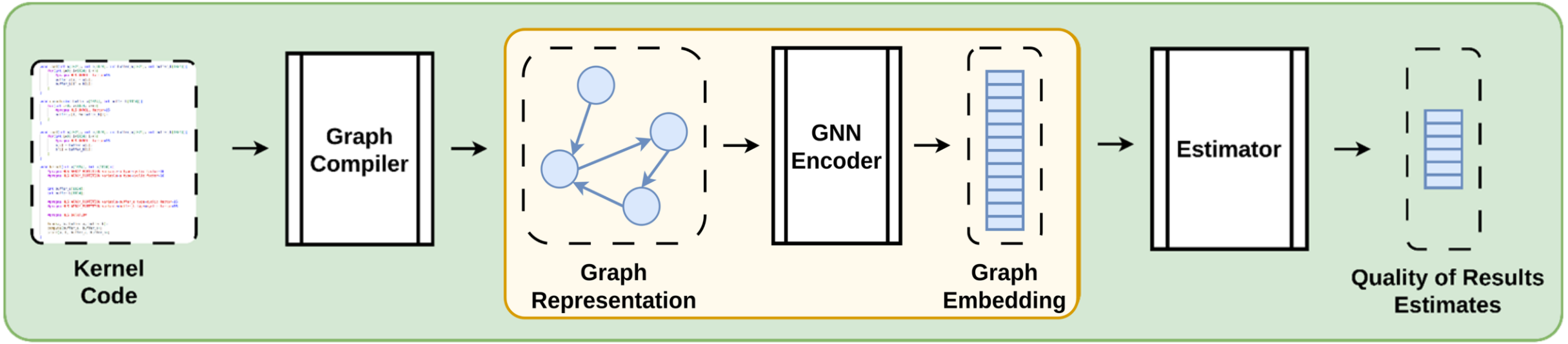
Design Space Exploration for HLS





How to quickly and accurately estimate the quality of results (QoR)?

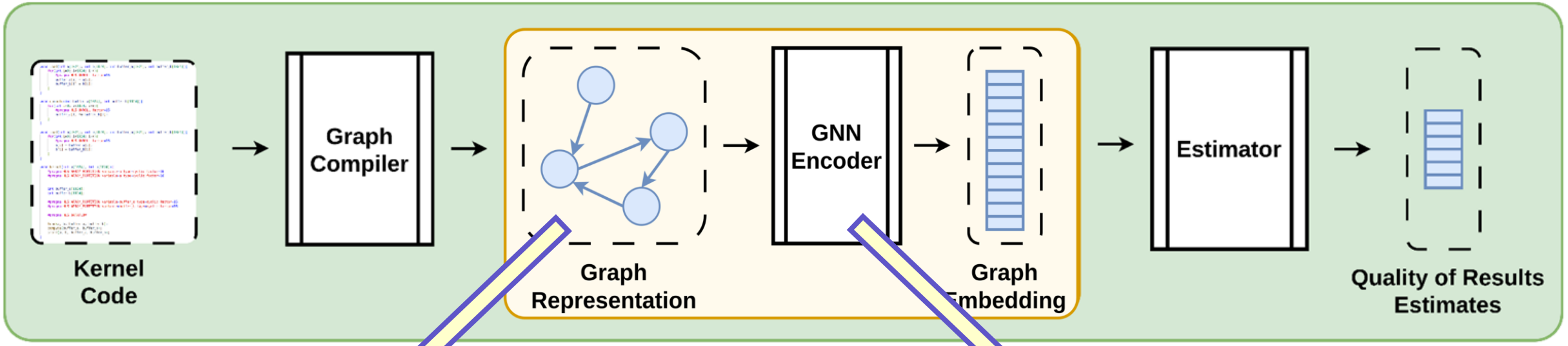
How to Go About QoR Estimation?

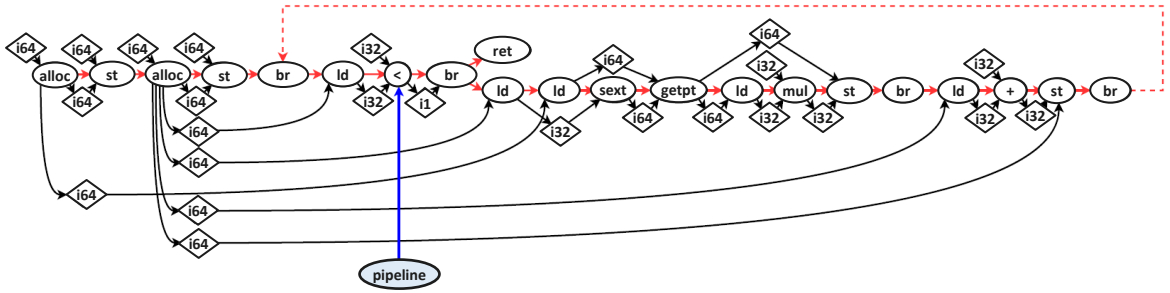


GNNs fully encode the program graph's topological information

Sohrabizadeh, Bai, Sun, and Cong. Robust GNN-Based Representation Learning for HLS. ICCAD 2023.
Gao, Zhao, Lin, and Guo. Hierarchical Source-to-Post-Route QoR Prediction in High-Level Synthesis with GNNs. DATE 2024.

How to Go About QoR Estimation?



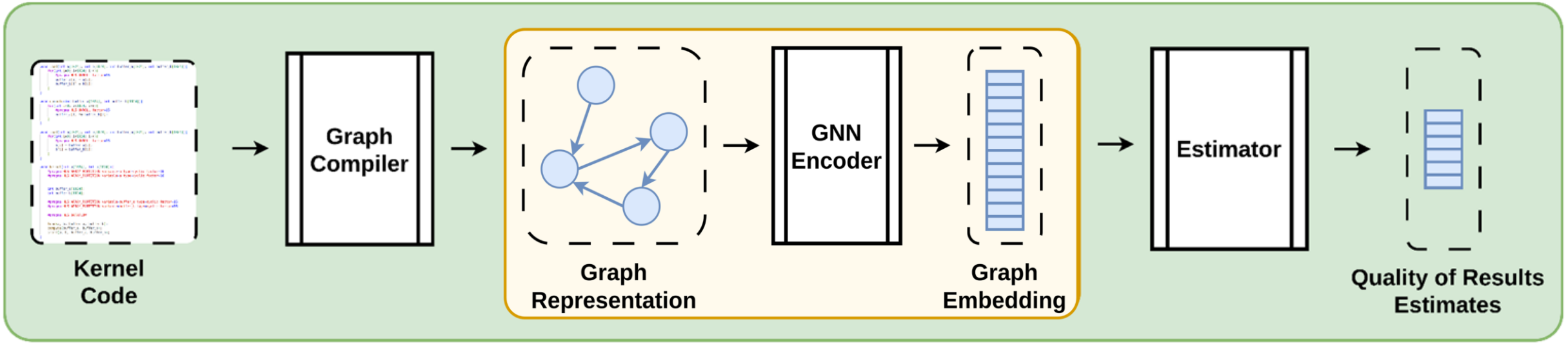


Based on a standard compiler representation
→ **complex, redundant information**

"Wide" GNN (large receptive field): poor scaling, limited cross-graph interactions, **low accuracy**

Sohrabizadeh, Bai, Sun, and Cong. Robust GNN-Based Representation Learning for HLS. ICCAD 2023.
Gao, Zhao, Lin, and Guo. Hierarchical Source-to-Post-Route QoR Prediction in High-Level Synthesis with GNNs. DATE 2024.

How to Go About QoR Estimation?



Balor, our HLS QoR estimator, increases estimation accuracy and reduces computational cost

A new **HLS-tailored graph compiler**

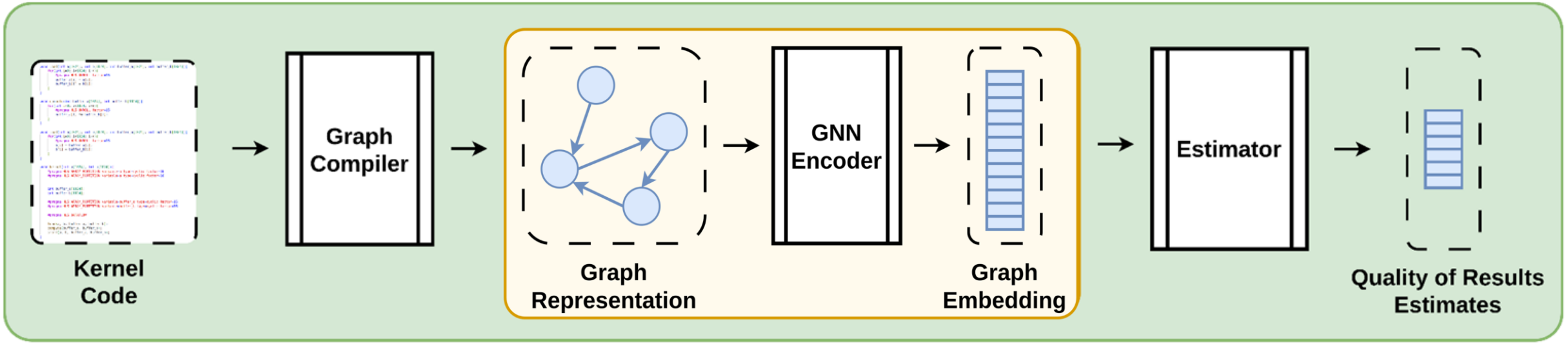
- HLS-specific, information-dense graph

Revisited **estimation stack**

- Local GNNs localize cross-graph interactions
- Hierarchical GNNs increase information density

Murphy and Josipović. Balor: HLS Source Code Evaluator Based on Custom Graphs and Hierarchical GNNs. ICCAD 2024.

How to Go About QoR Estimation?



Balor, our HLS QoR estimator, increases estimation accuracy and reduces computational cost

DB4HLS
LUT mean percentage error: 4%
Latency mean percentage error: 6%

ML Contest for Chip Design with HLS 2024:
1st place in QoR estimation

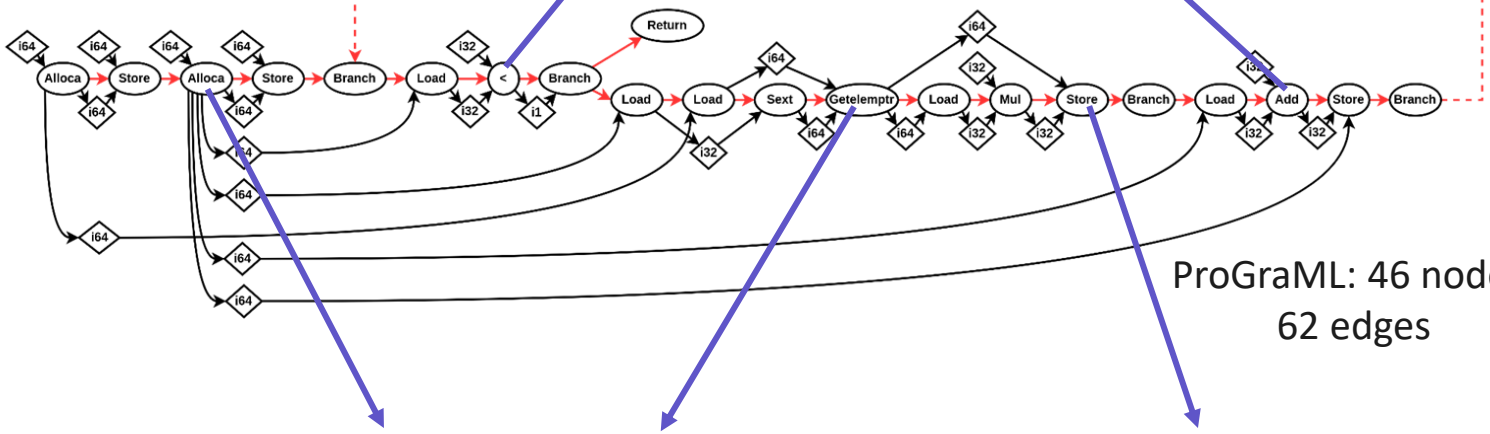
Murphy and Josipović. Balor: HLS Source Code Evaluator Based on Custom Graphs and Hierarchical GNNs. ICCAD 2024.

Graph Representation

- Graph representation directly obtained from **compiler intermediate representation (IR)**
  - **ProGraML**, used by existing estimation approaches, builds graph directly from the LLVM IR

```
void kernel(int a[1024]){
    for(int i=0; i<1024; i++){
        a[i] *= 5;
    }
}
```

Large graph with long-range interactions (e.g., loop iteration info distributed across the graph)



ProGraML: 46 nodes, 62 edges

Standard compiler representations are not intended for estimation tasks

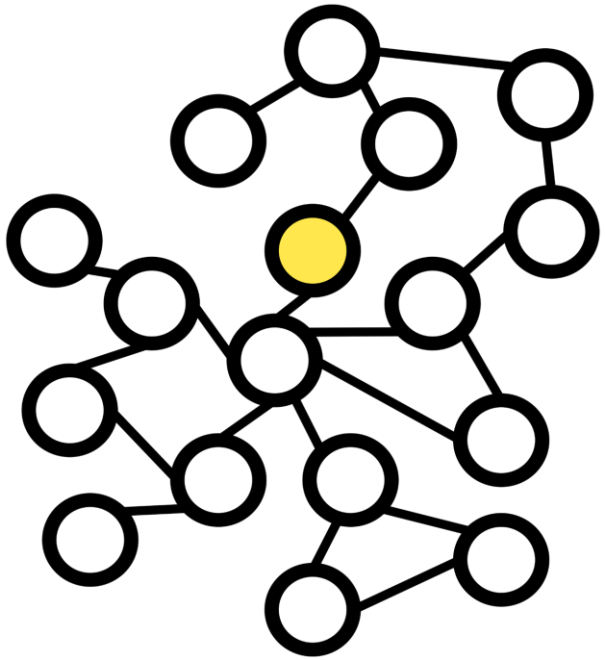
Nodes irrelevant for HLS (e.g., memory allocation)

HW-relevant info missing (Local? Global? Memory interface?)

Cummins et. al. ProGraML: A Graph-based Program Representation for Data Flow Analysis and Compiler Optimizations. ICML 2021.
Sohrabizadeh, Bai, Sun, and Cong. Robust GNN-based representation learning for HLS. ICCAD 2023.
Gao, Zhao, Lin, and Guo. Hierarchical Source-to-Post-Route QoR Prediction in High-Level Synthesis with GNNs. DATE 2024.

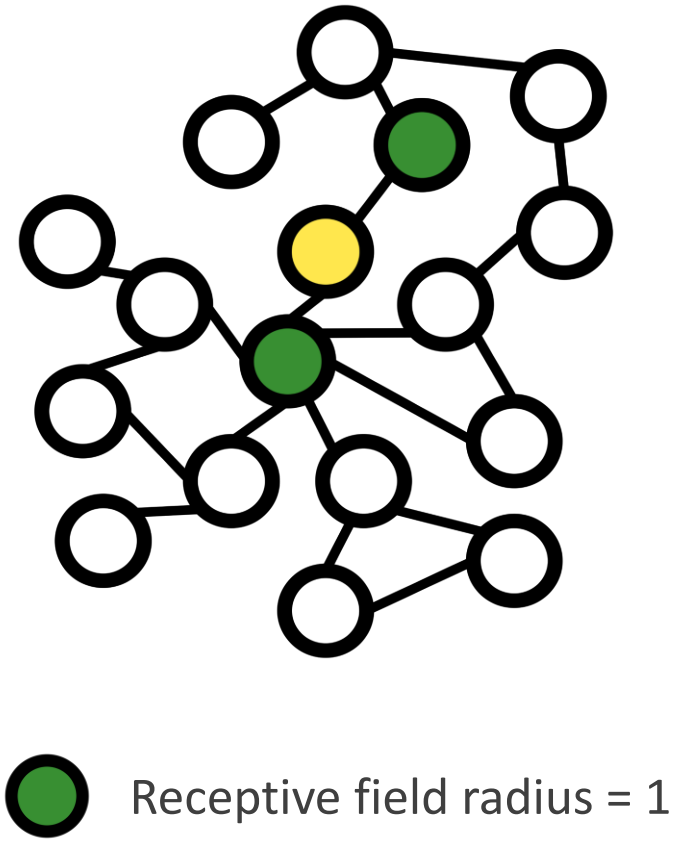
GNNs and Cross-Graph Interactions

- Previously: long-range program graph interactions forced the use of **large receptive fields**
  - **Receptive field:** a window into the graph



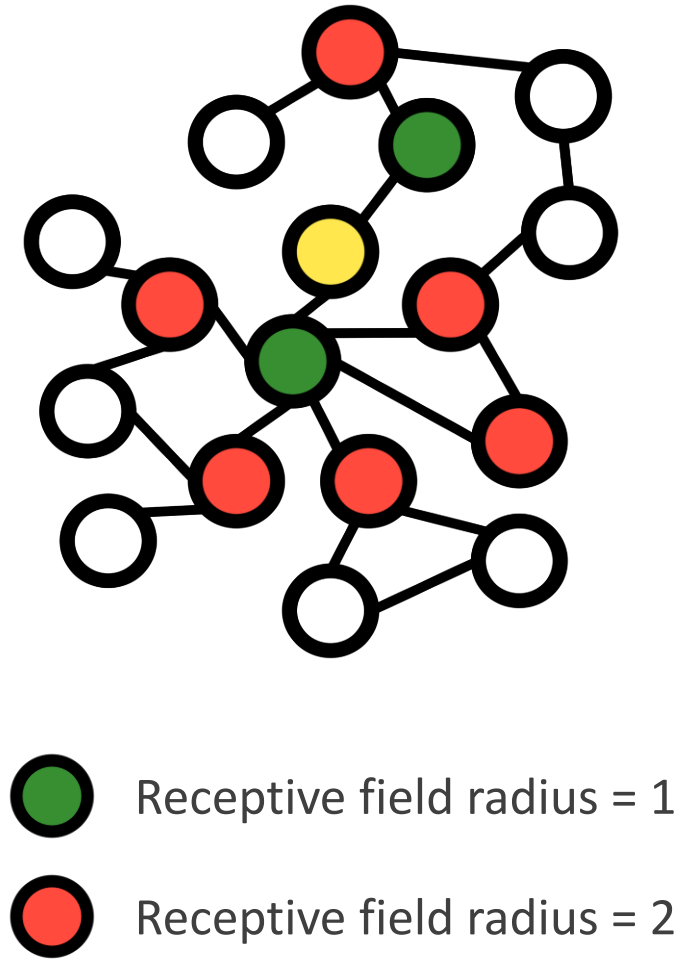
GNNs and Cross-Graph Interactions

- Previously: long-range program graph interactions forced the use of **large receptive fields**
  - **Receptive field:** a window into the graph



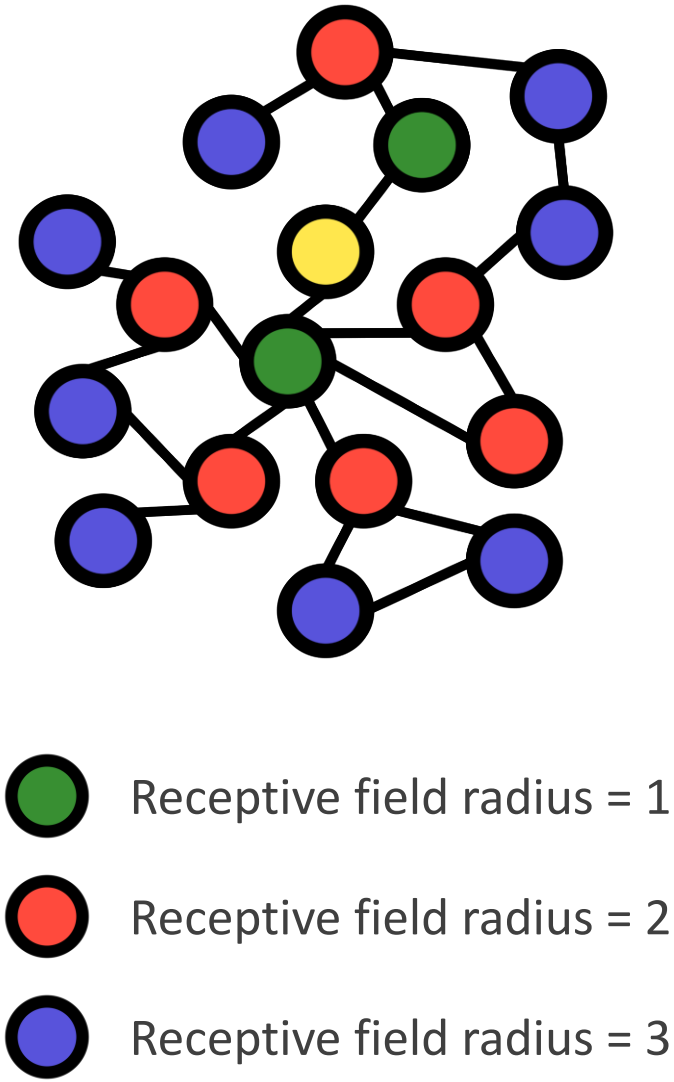
GNNs and Cross-Graph Interactions

- Previously: long-range program graph interactions forced the use of **large receptive fields**
  - **Receptive field:** a window into the graph



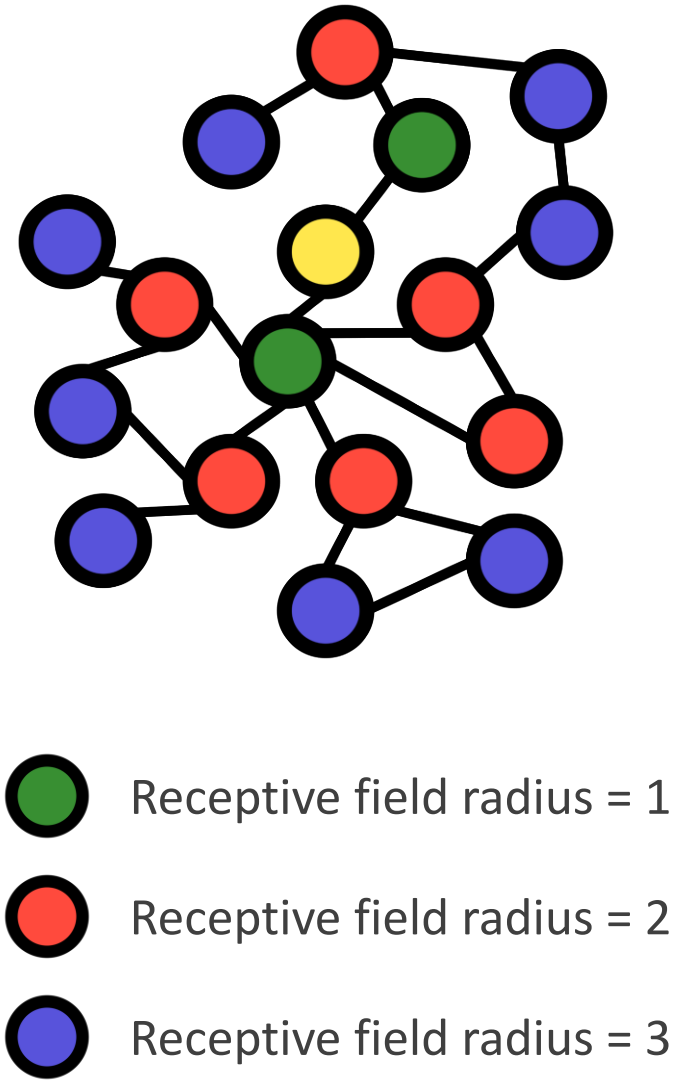
GNNs and Cross-Graph Interactions

- Previously: long-range program graph interactions forced the use of **large receptive fields**
  - **Receptive field:** a window into the graph



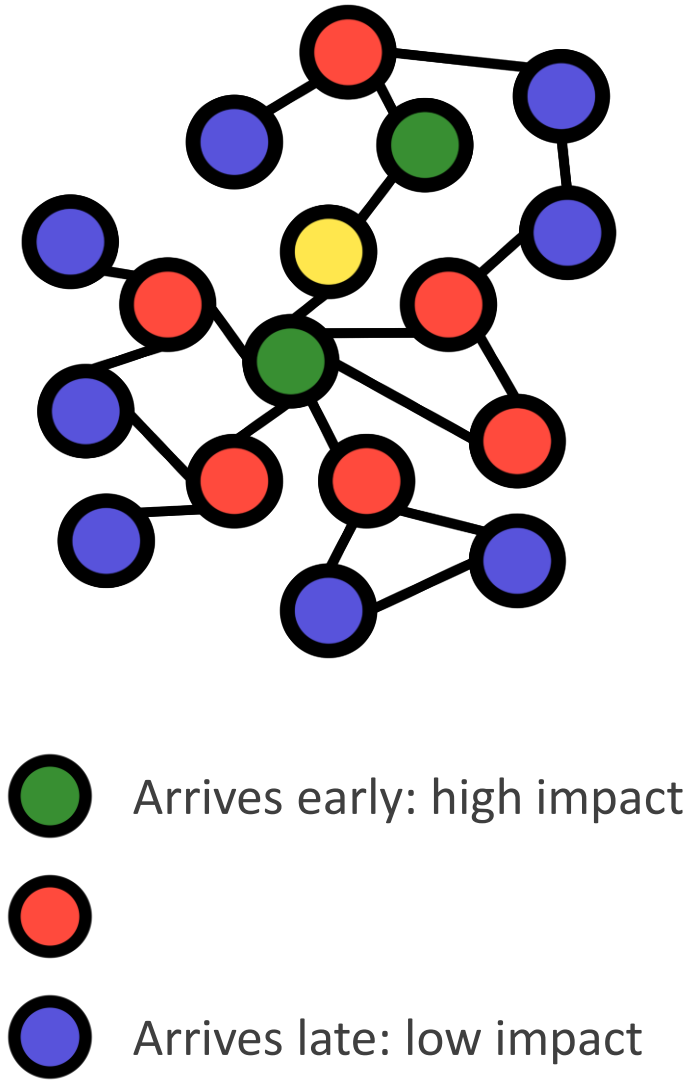
GNNs and Cross-Graph Interactions

- Previously: long-range program graph interactions forced the use of **large receptive fields**
  - **Receptive field:** a window into the graph
- Problem 1: **poor GNN scaling**
  - Large receptive fields cause redundant processing
  - Redundant processing **reduces estimation accuracy**



GNNs and Cross-Graph Interactions

- Previously: long-range program graph interactions forced the use of **large receptive fields**
  - **Receptive field:** a window into the graph
- Problem 1: **poor GNN scaling**
  - Large receptive fields cause redundant processing
  - Redundant processing **reduces estimation accuracy**
- Problem 2: the **receptive field is not even**
  - Long-range interactions **unevenly accounted for**

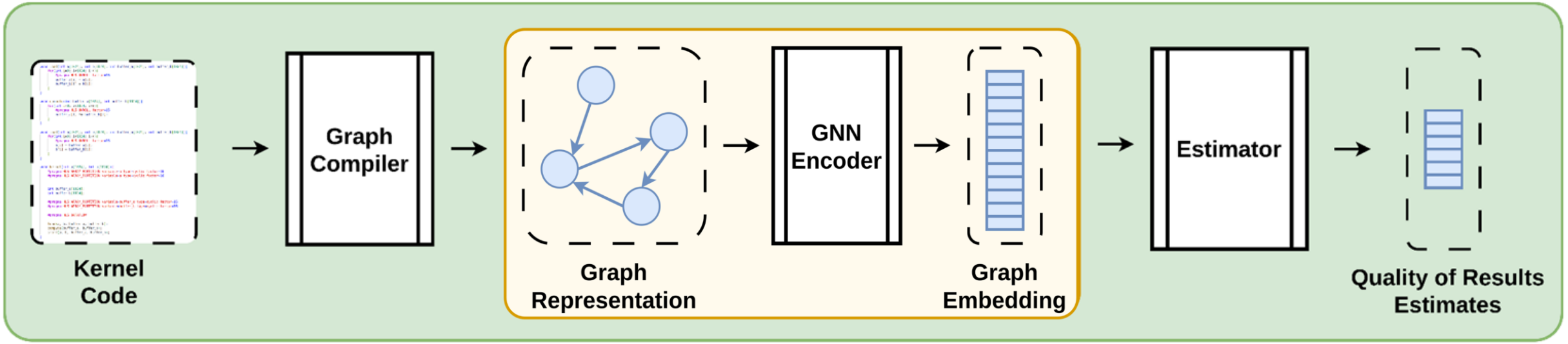


Receptive Field Size

"Wide" GNN (Large Receptive Field)	Local GNN (Small Receptive Field)
Poor GNN scaling causes **strongly redundant** computation	Low redundancy enables increased accuracy
Long-range interactions: **Unevenly accounted for** during estimation due to **late arrival**	Long-range interactions: **Not accounted for** during estimation
Information Density: No problems	Information Density: Small receptive field requires **"information-dense" graphs**

How to devise information-dense graphs suitable for local GNNs?

How to Go About QoR Estimation?



Balor, our HLS QoR estimator, increases estimation accuracy and reduces computational cost

A new **HLS-tailored graph compiler**

- HLS-specific, information-dense graph

Revisited **estimation stack**

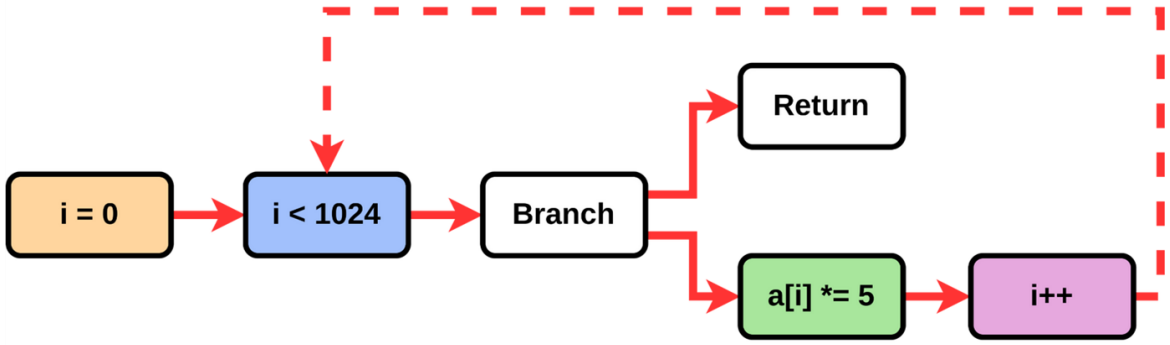
- Local GNNs localize cross-graph interactions
- Hierarchical GNNs increase information density

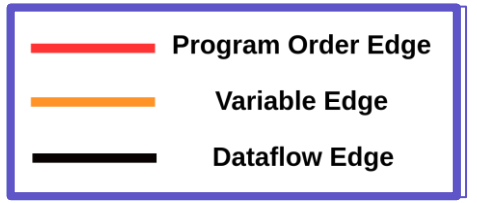
Murphy and Josipović. Balor: HLS Source Code Evaluator Based on Custom Graphs and Hierarchical GNNs. ICCAD 2024.

Balor's Graph Compiler

```
void kernel(int a[1024]){
    for(int i=0; i<1024; i++){
        a[i] *= 5;
    }
}
```

1. Create node for each code statement

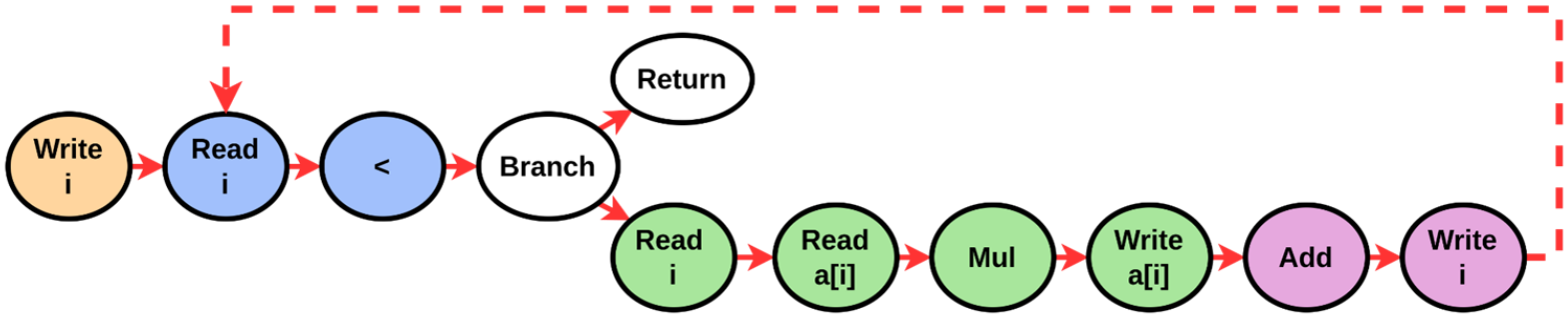


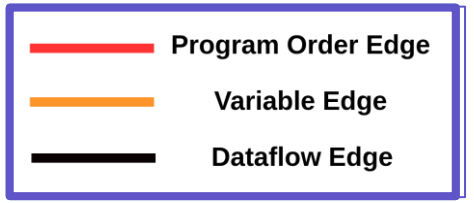


Balor's Graph Compiler

```
void kernel(int a[1024]){
    for(int i=0; i<1024; i++){
        a[i] *= 5;
    }
}
```

1. Create node for each code statement
2. Break nodes into indivisible hardware actions

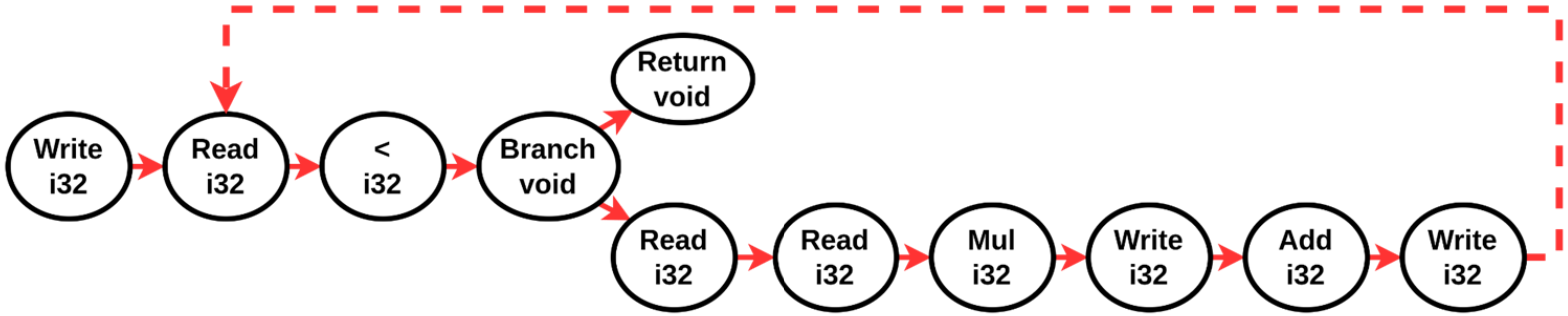


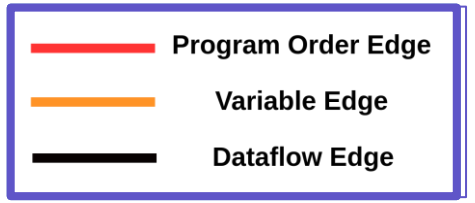


Balor's Graph Compiler

```
void kernel(int a[1024]){
    for(int i=0; i<1024; i++){
        a[i] *= 5;
    }
}
```

1. Create node for each code statement
2. Break nodes into indivisible hardware actions
3. Label each node with data type

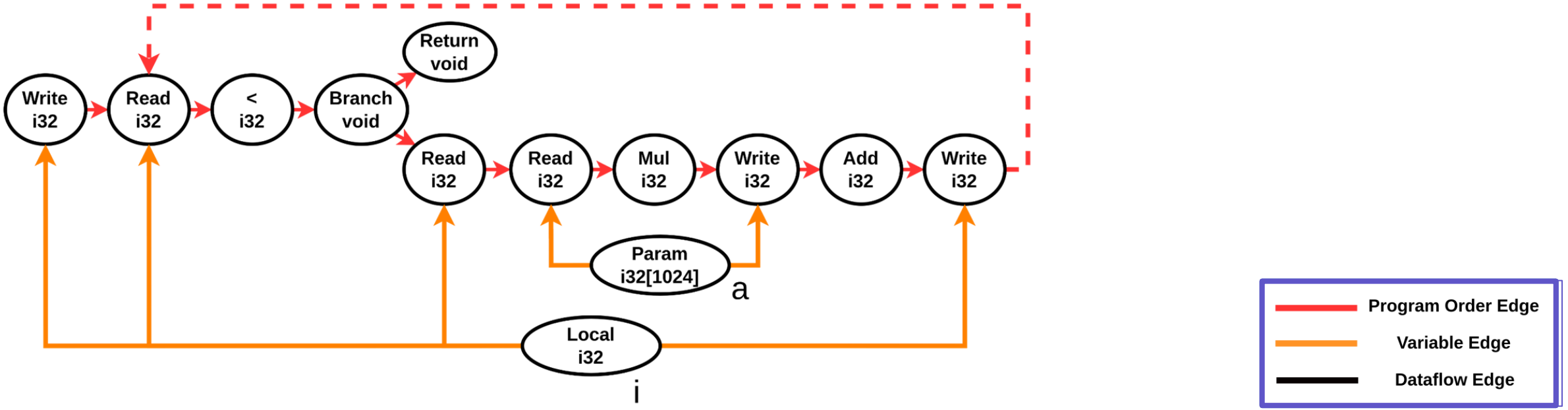




Balor's Graph Compiler

```
void kernel(int a[1024]){
    for(int i=0; i<1024; i++){
        a[i] *= 5;
    }
}
```

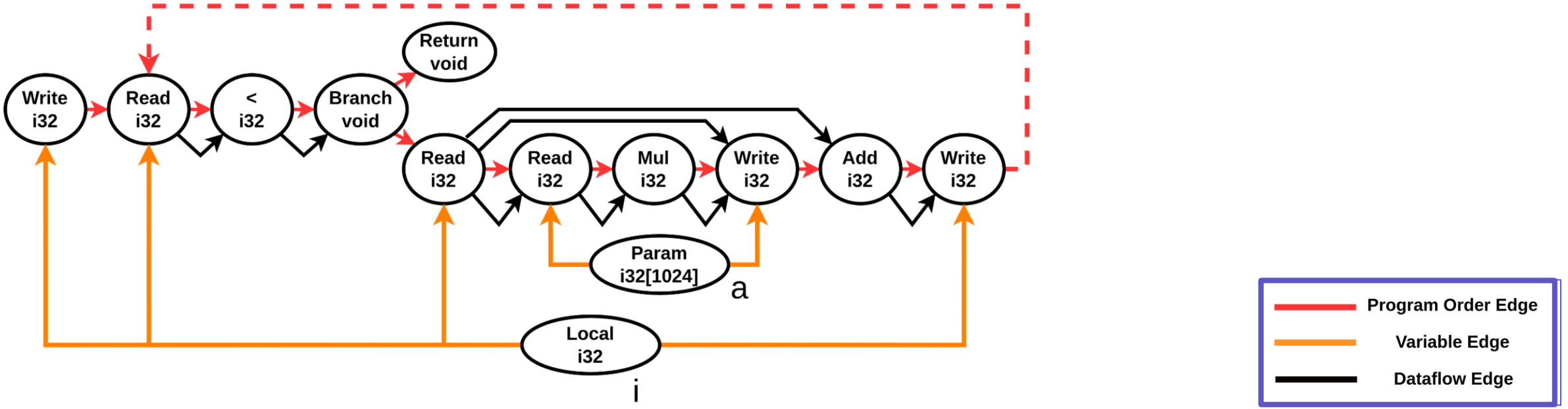
1. Create node for each code statement
2. Break nodes into indivisible hardware actions
3. Label each node with data type
4. Add variable declaration nodes & connect them to reads/writes



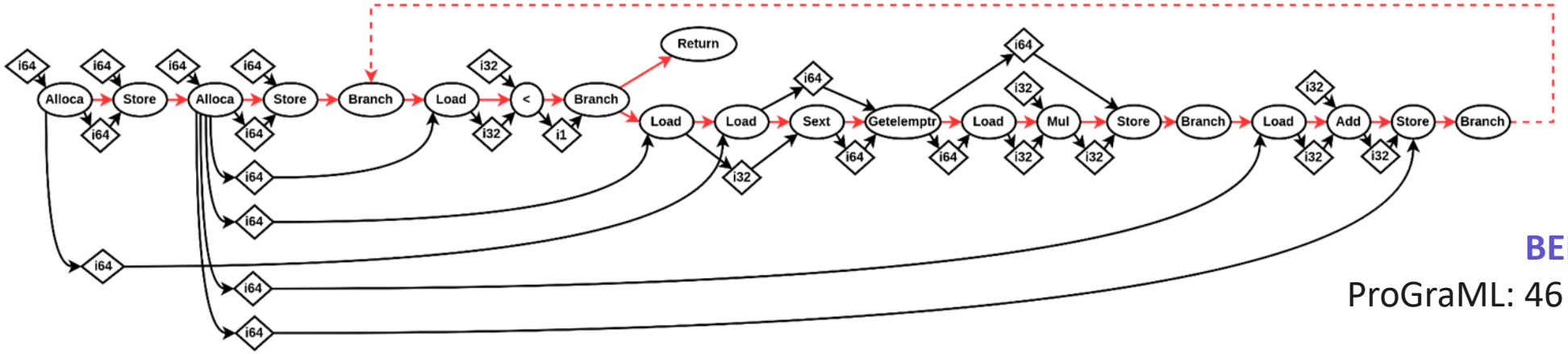
Balor's Graph Compiler

```
void kernel(int a[1024]){
    for(int i=0; i<1024; i++){
        a[i] *= 5;
    }
}
```

1. Create node for each code statement
2. Break nodes into indivisible hardware actions
3. Label each node with data type
4. Add variable declaration nodes & connect them to reads/writes
5. Add dataflow edges

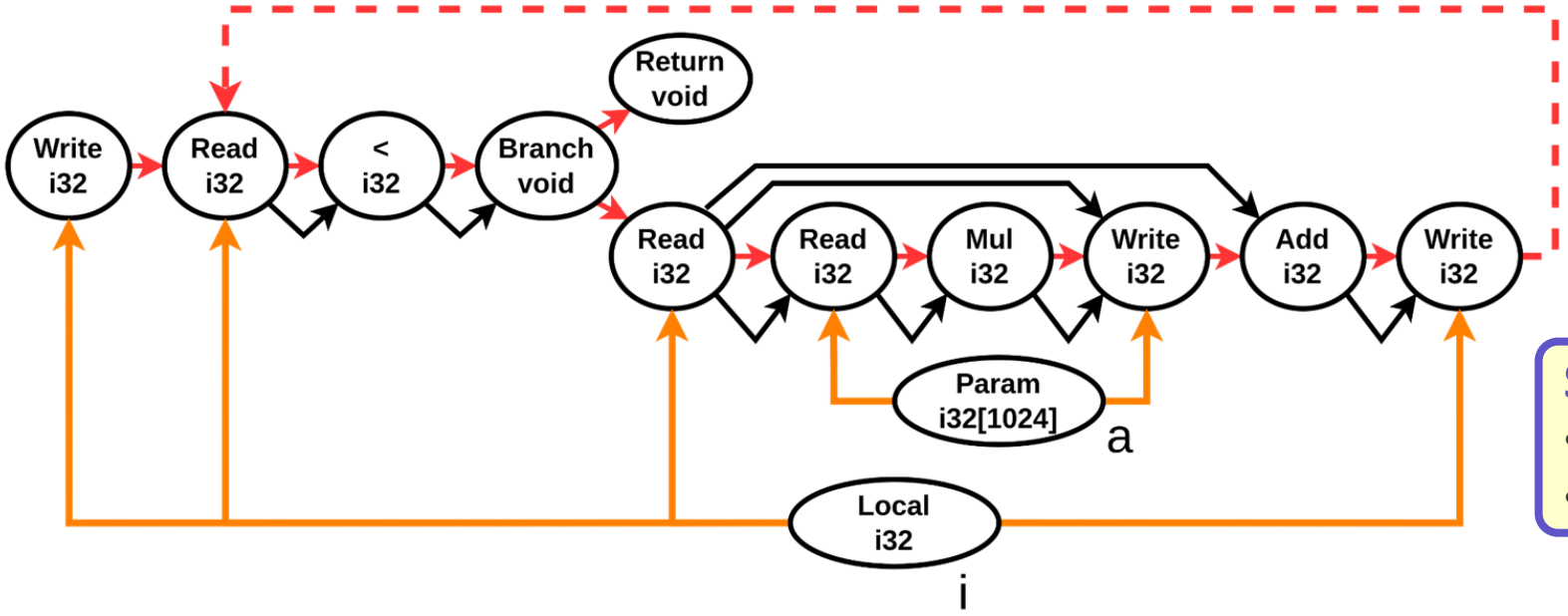


Balor's Graph Compiler



BEFORE

ProGraML: 46 nodes, 62 edges



NOW

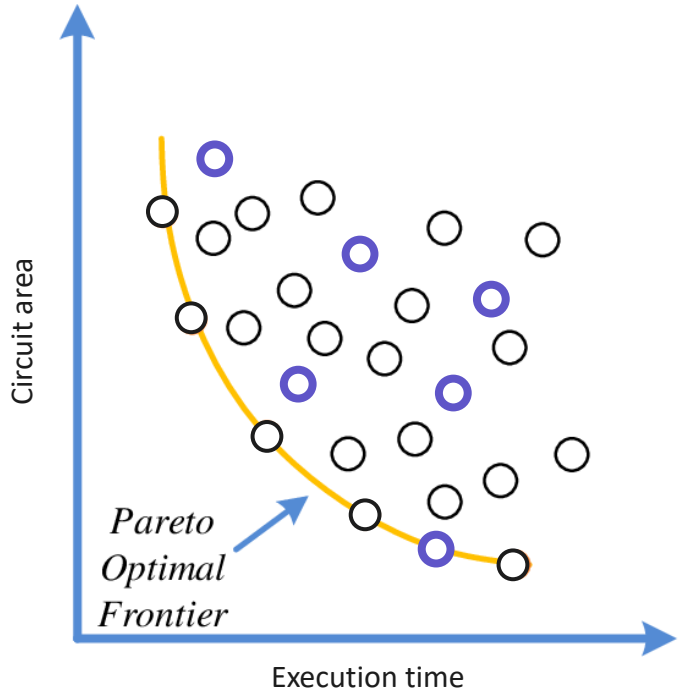
Balor: 13 nodes, 25 edges

Simplified, HLS-tailored graph

- Irrelevant **SW information** is omitted
- Relevant **HW information** is explicit

Evaluation

- **Architecture:** GNN DSE
  - The core of state-of-the-art architectures
- **Dataset:** DB4HLS
  - 36,296 data points, synthesized with Vitis HLS
  - 29 Machsuite kernels
- **Estimate:**
  - Latency (clock cycles)
  - Clock period (ns)
  - Resource usage (LUTs/FFs/BRAMs/DSPs)



Training & validation

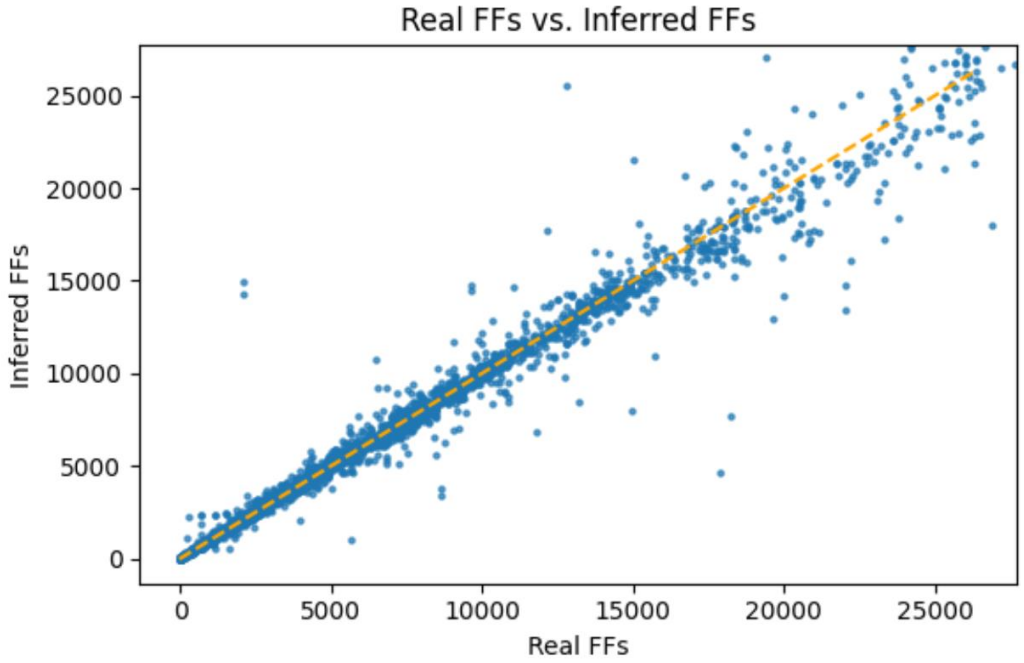
Testing (estimation)

Sohrabizadeh, Bai, Sun, and Cong. Automated Accelerator Optimization Aided by Graph Neural Networks. DAC 2022.
Ferretti, Kwon, Ansaloni, Di Guglielmo, Carloni, and Pozzi. DB4HLS: A Database of High-Level Synthesis. ESL 2021

Evaluation

- **Architecture:** GNN DSE
  - The core of state-of-the-art architectures
- **Dataset:** DB4HLS
  - 36,296 data points, synthesized with Vitis HLS
  - 29 Machsuite kernels
- **Estimate:**
  - Latency (clock cycles)
  - Clock period (ns)
  - Resource usage (LUTs/FFs/BRAMs/DSPs)
- **Percentage error:**

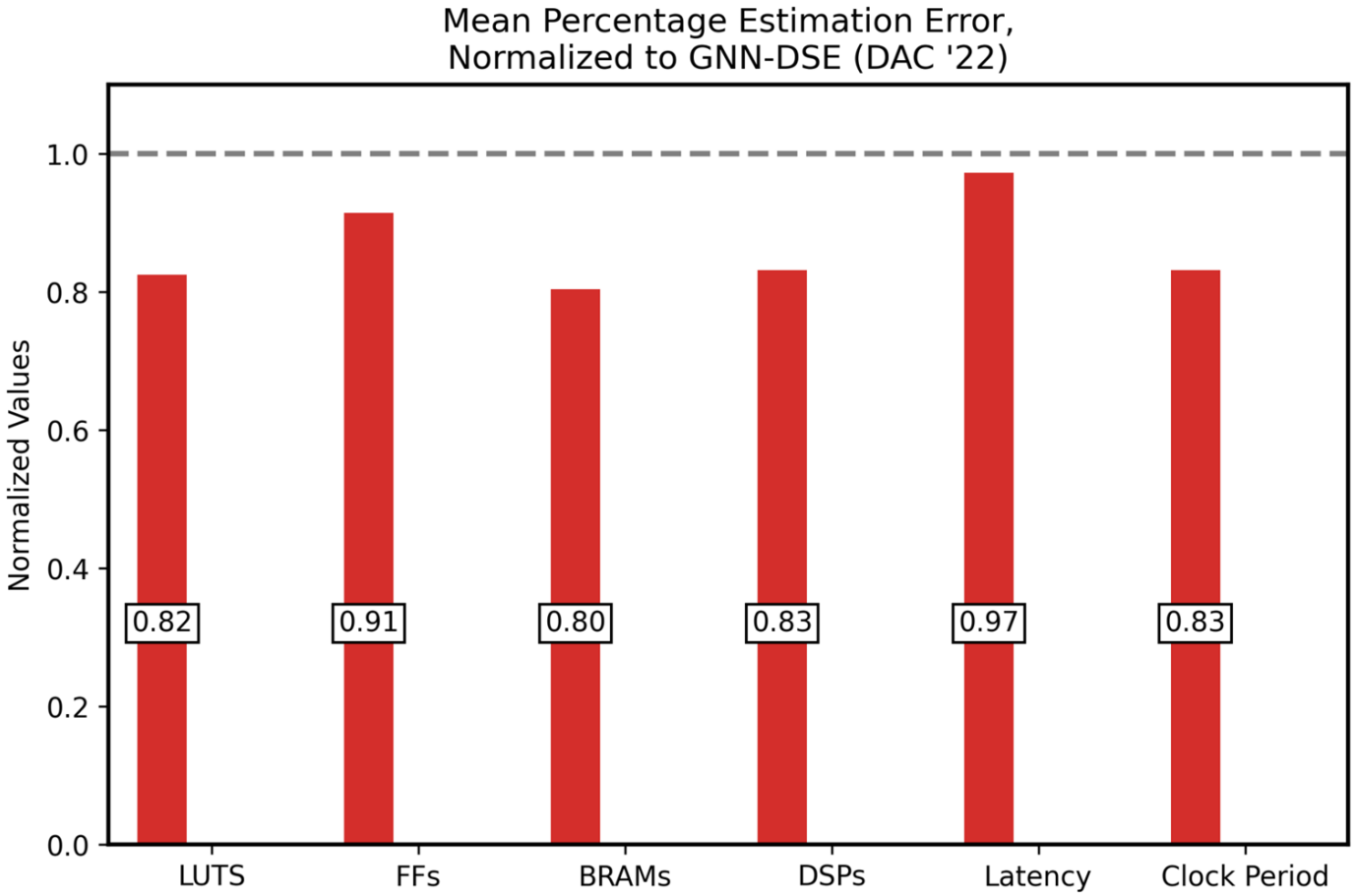
$$\frac{\text{abs}(\text{Real Value} - \text{Inferred Value})}{\text{Real Value}} * 100$$

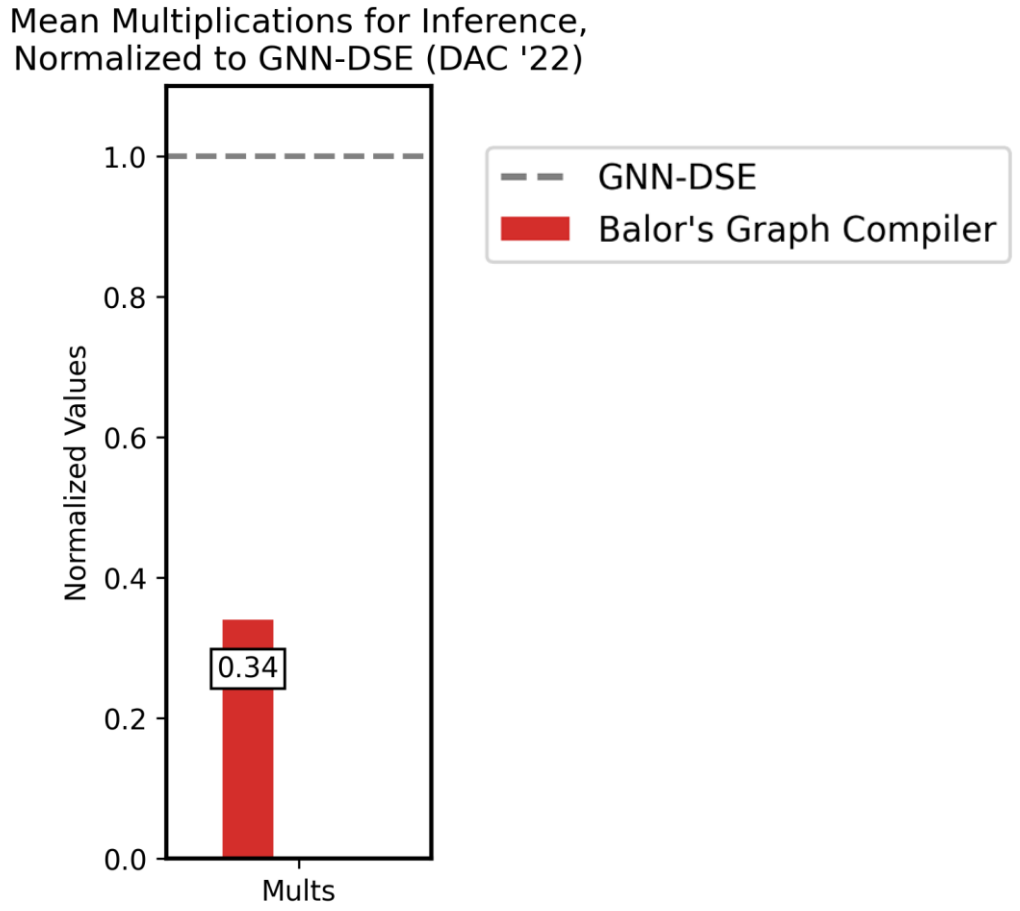


Example scatter plot, 5,673 test points

Sohrabizadeh, Bai, Sun, and Cong. Automated Accelerator Optimization Aided by Graph Neural Networks. DAC 2022.
Ferretti, Kwon, Ansaloni, Di Guglielmo, Carloni, and Pozzi. DB4HLS: A Database of High-Level Synthesis. ESL 2021

Evaluation

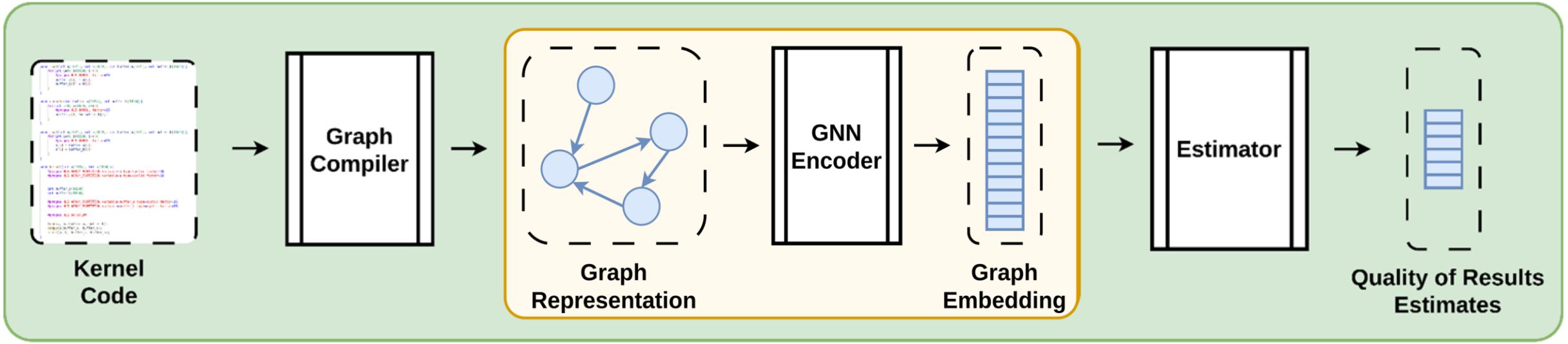




Information-dense, HLS-tailored graphs reduce both error and cost

Sohrabizadeh, Bai, Sun, and Cong. Automated Accelerator Optimization Aided by Graph Neural Networks. DAC 2022.

How to Go About QoR Estimation?



Balor, our QoR estimator, increases estimation accuracy and reduces computational cost

A new **HLS-tailored graph compiler**

- HLS-specific, information-dense graph

Revisited **estimation stack**

- Local GNNs localize cross-graph interactions
- Hierarchical GNNs increase information density

Murphy and Josipović. Balor: HLS Source Code Evaluator Based on Custom Graphs and Hierarchical GNNs. ICCAD 2024.

Receptive Field Size

"Wide" GNN (Large Receptive Field)	Local GNN (Small Receptive Field)
Poor GNN scaling causes **strongly redundant** computation	Low redundancy enables increased accuracy
Long-range interactions: **Unevenly accounted for** during estimation due to **late arrival**	Long-range interactions: **Not accounted for** during estimation
Information Density: No problems	Information Density: Small receptive field requires **"information-dense" graphs**

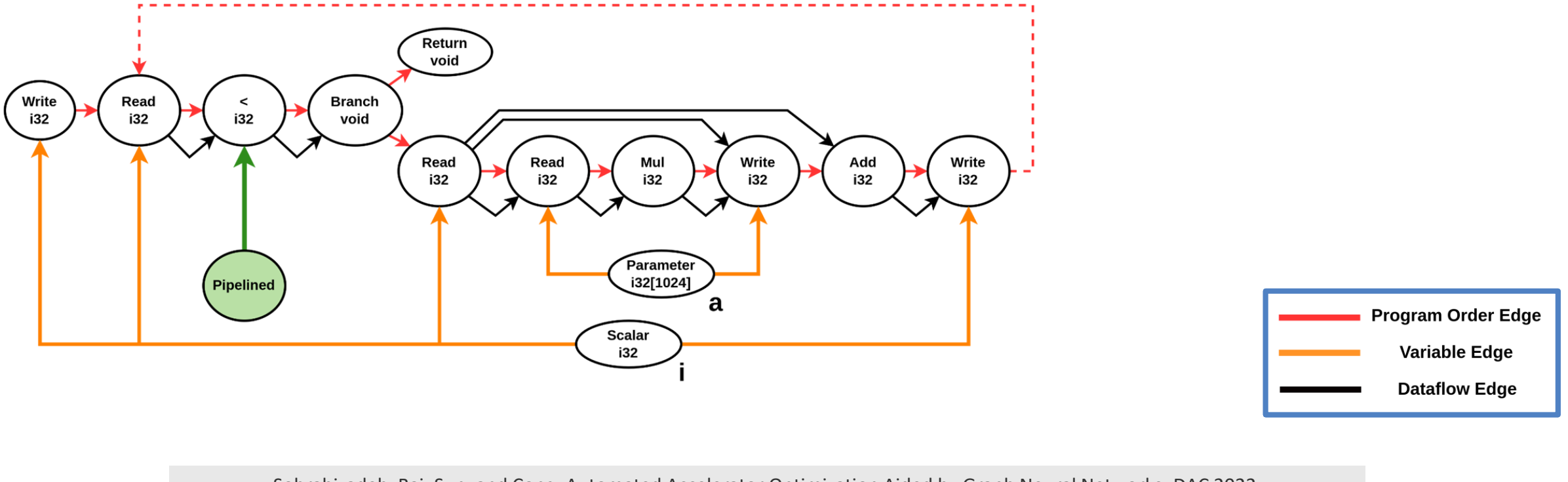
Are our custom graphs suitable for local GNNs?

Cross-Graph Interactions: The Pragma Challenge

```
void kernel(int a[1024]){
    for(int i=0; i<1024; i++){
        #pragma HLS pipeline
        a[i] *= 5;
    }
}
```

Prior philosophy (e.g., GNN-DSE and HARP): **information has location**
→ pragma information isolated to single node

Pragma information propagates unevenly to affected nodes



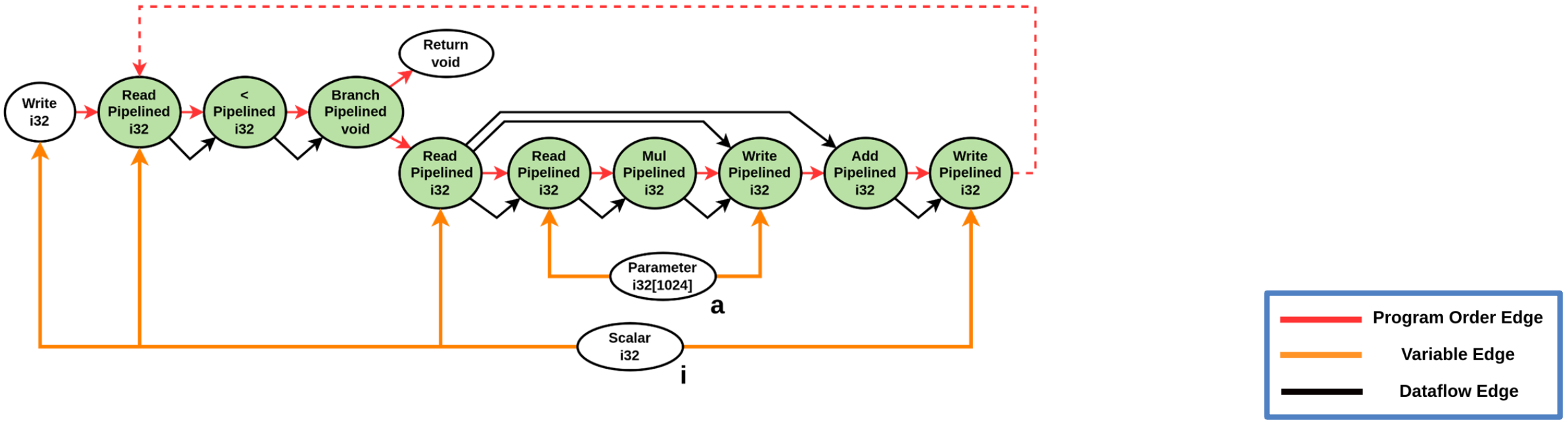
Sohrabizadeh, Bai, Sun, and Cong. Automated Accelerator Optimization Aided by Graph Neural Networks. DAC 2022.
Sohrabizadeh, Bai, Sun, and Cong. Robust GNN-Based Representation Learning for HLS. ICCAD 2023.

Cross-Graph Interactions: The Pragma Challenge

```
void kernel(int a[1024]){
    for(int i=0; i<1024; i++){
        #pragma HLS pipeline
        a[i] *= 5;
    }
}
```

Our philosophy: **locations have information**
→ pragma information distributed to all affected nodes

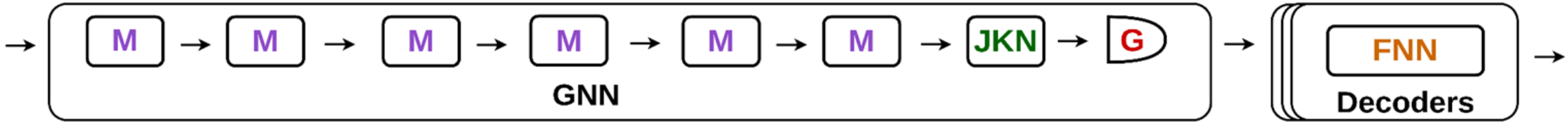
Pragma information available everywhere where it matters



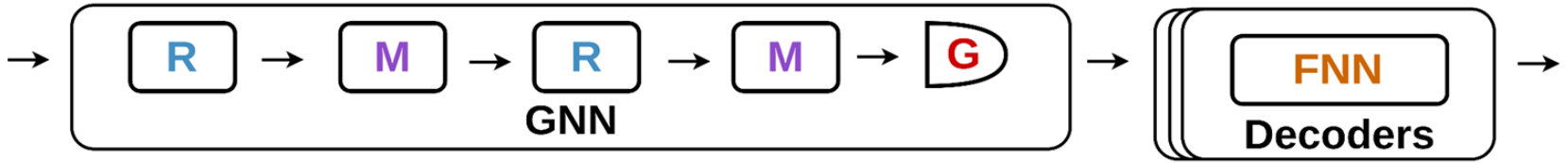
Sohrabizadeh, Bai, Sun, and Cong. Automated Accelerator Optimization Aided by Graph Neural Networks. DAC 2022.
Sohrabizadeh, Bai, Sun, and Cong. Robust GNN-Based Representation Learning for HLS. ICCAD 2023.

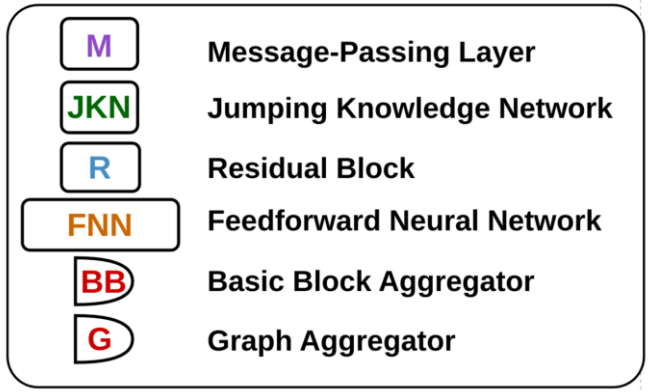
Move to a Local Architecture

GNN-DSE (DAC '22)



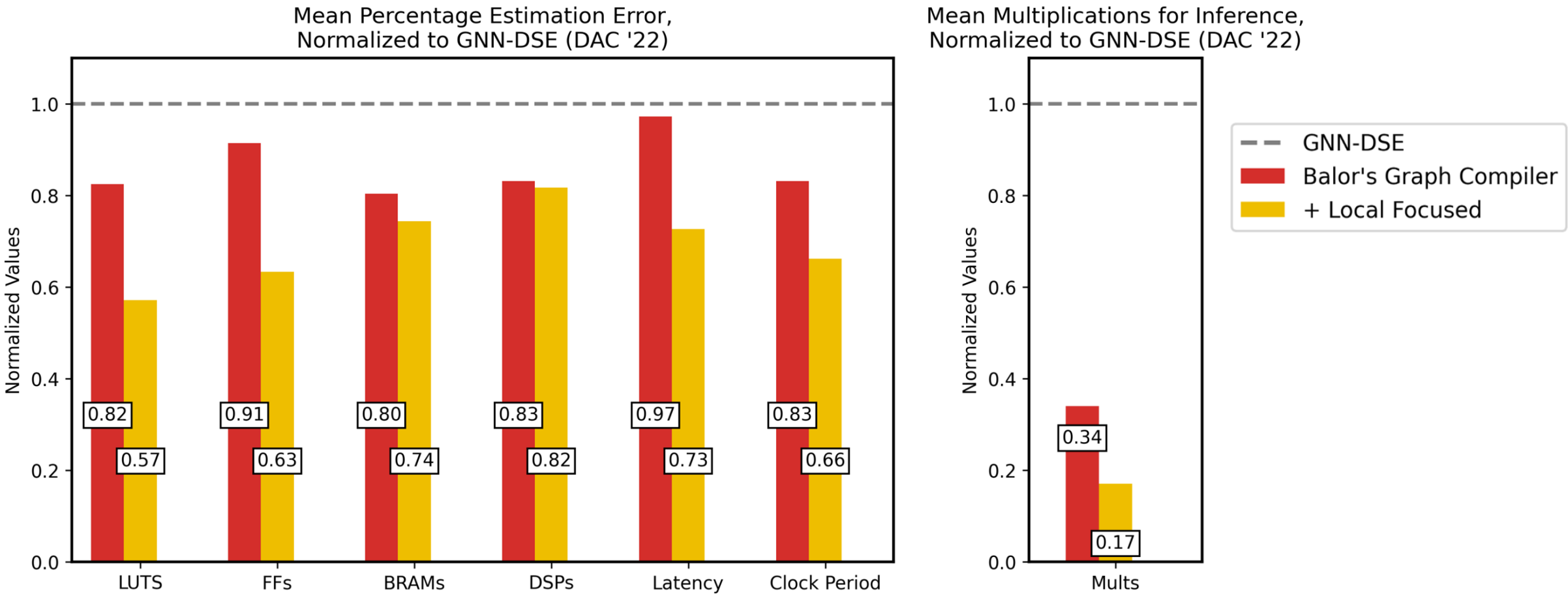
Move to a **strongly local** architecture





Sohrabizadeh, Bai, Sun, and Cong. Automated Accelerator Optimization Aided by Graph Neural Networks. DAC 2022.

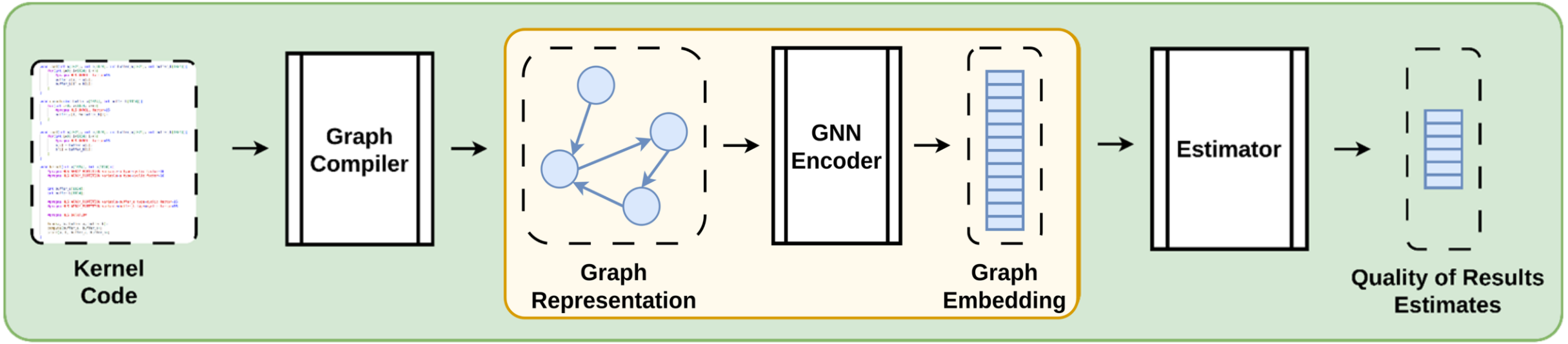
Evaluation



Local-focused approaches reduce redundant processing

Sohrabizadeh, Bai, Sun, and Cong. Automated Accelerator Optimization Aided by Graph Neural Networks. DAC 2022.

How to Go About QoR Estimation?



Balor, our QoR estimator, increases estimation accuracy and reduces computational cost

A new **HLS-tailored graph compiler**

- HLS-specific, information-dense graph

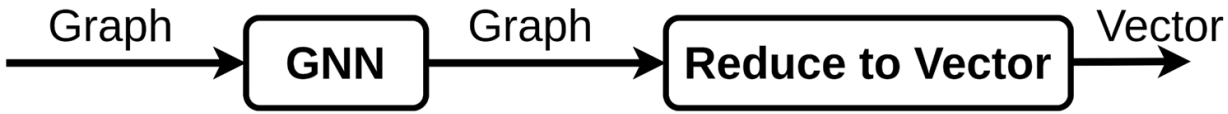
Revisited **estimation stack**

- Local GNNs localize cross-graph interactions
- Hierarchical GNNs increase information density

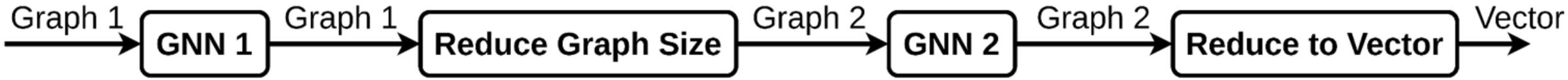
Murphy and Josipović. Balor: HLS Source Code Evaluator Based on Custom Graphs and Hierarchical GNNs. ICCAD 2024.

What are Hierarchical GNNs?

Flat GNNs use 1 network and 1 reduction:



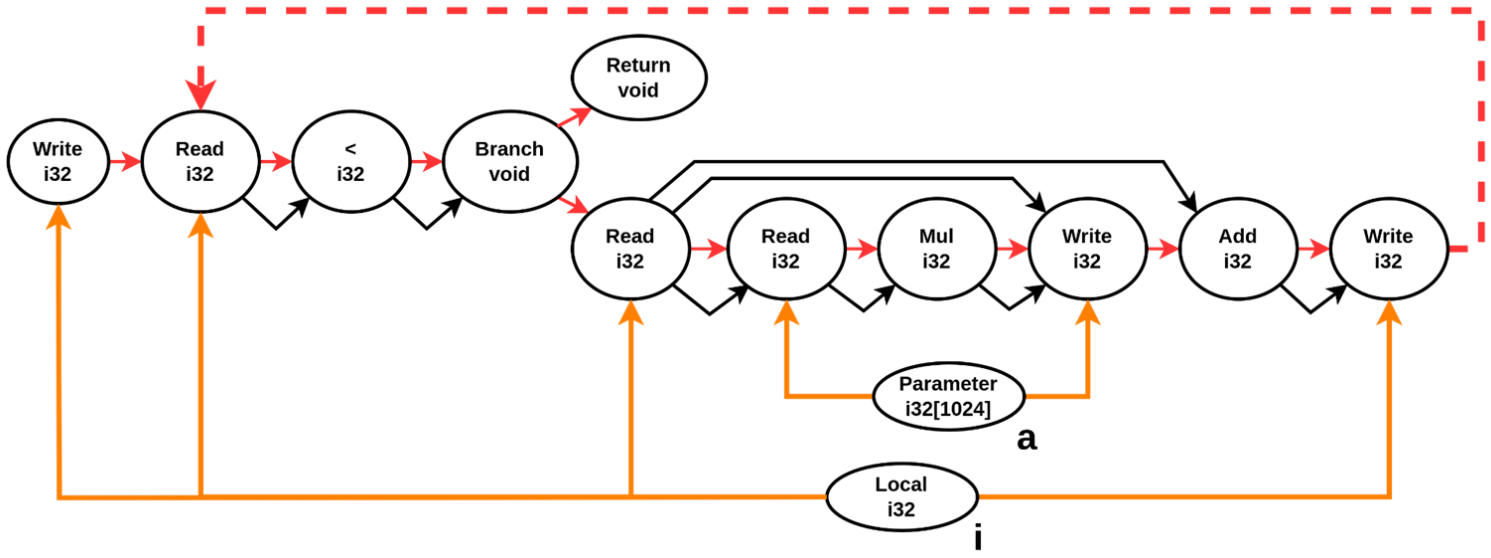
Hierarchical GNNs use multiple networks and multiple reductions to **increase information density**:



How to cluster the graph to reduce its size?

How to Cluster?

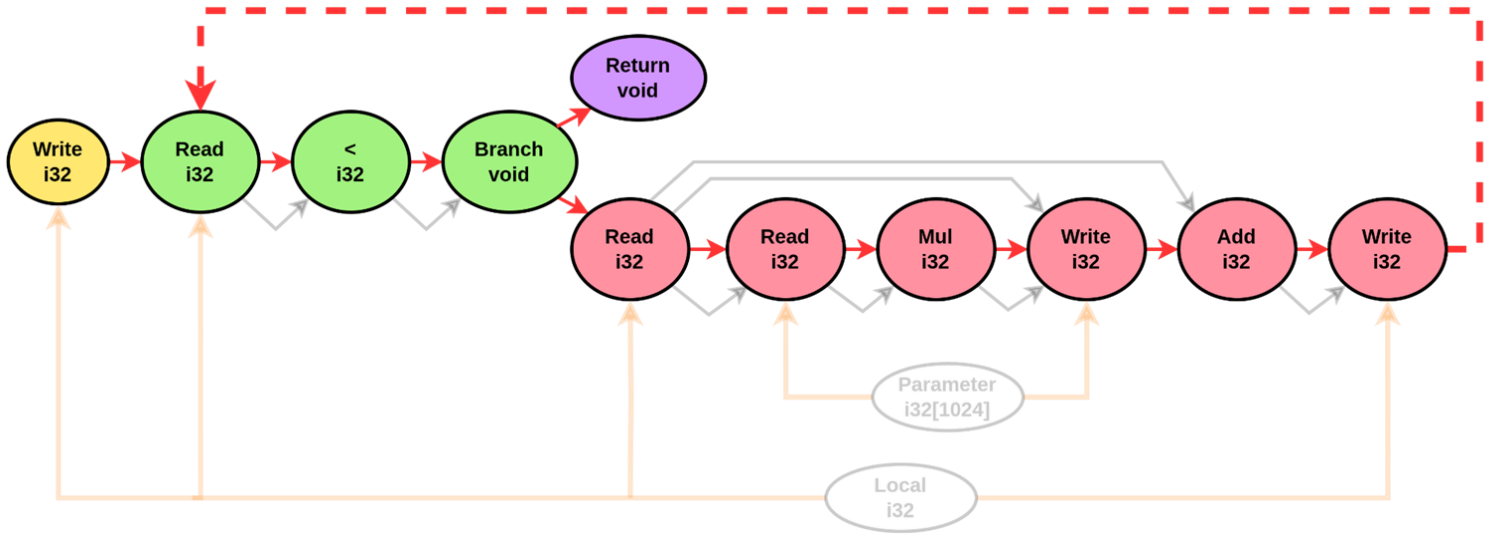
```
void kernel(int a[1024]){
    for(int i=0; i<1024; i++){
        a[i] *= 5;
    }
}
```

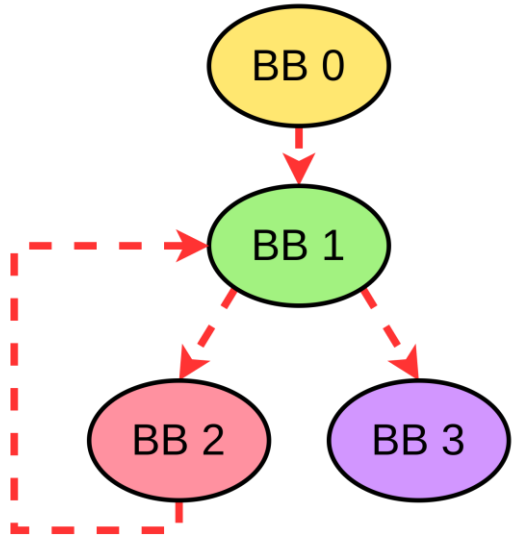


How to Cluster?

```
void kernel(int a[1024]){
    for(int i=0; i<1024; i++){
        a[i] *= 5;
    }
}
```

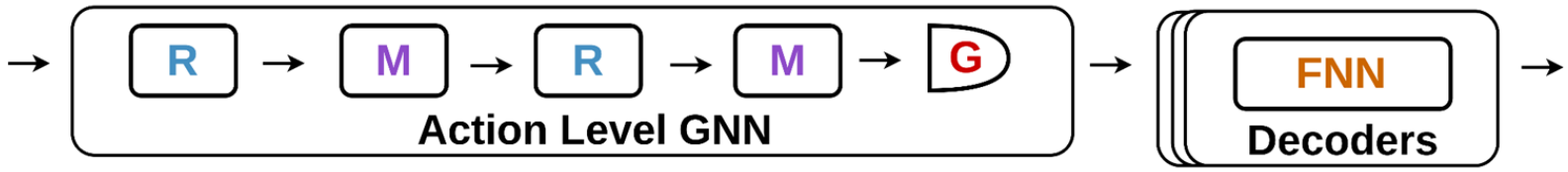
Good clusters already exist:
basic blocks in the control-flow graph



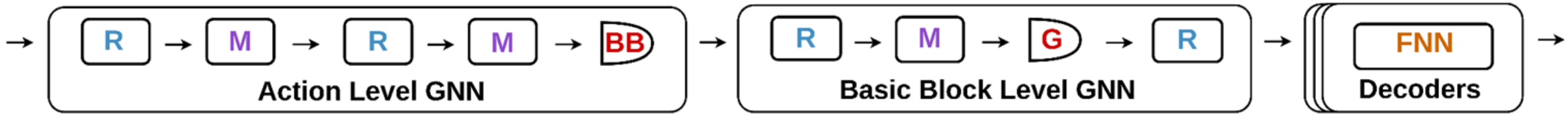


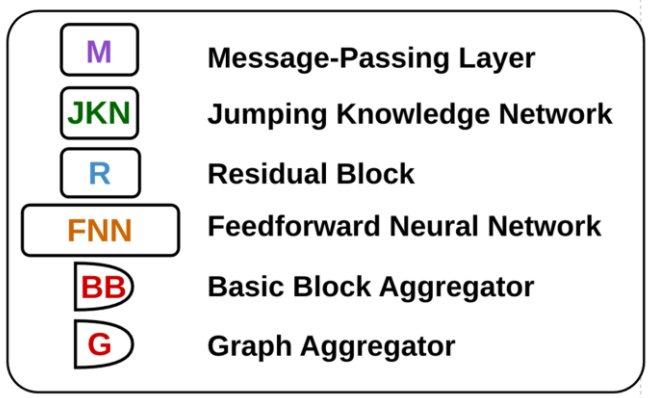
Hierarchical Architecture

Local-Focused Architecture:

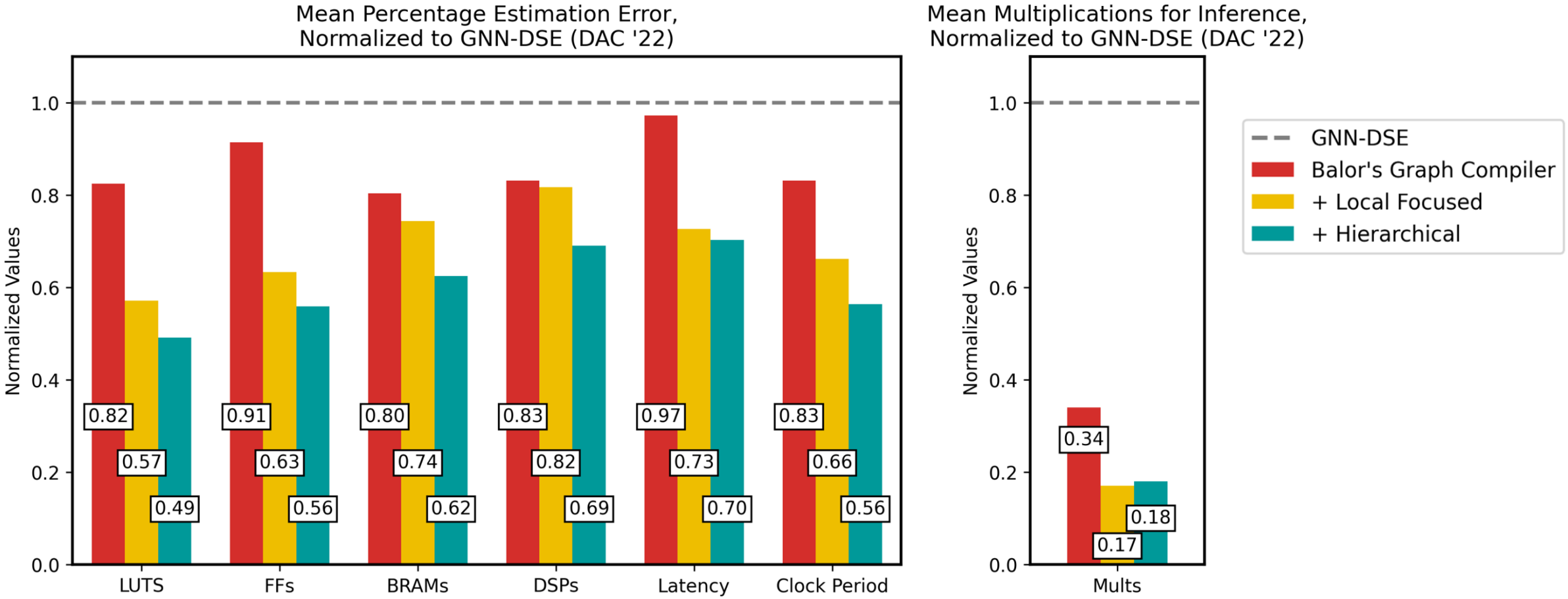


Move to a **hierarchical** architecture:





Evaluation



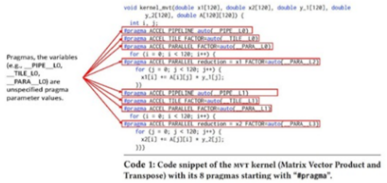
Hierarchical GNNs
further reduce error at low cost

Sohrabizadeh, Bai, Sun, and Cong. Automated Accelerator Optimization Aided by Graph Neural Networks. DAC 2022.

1st Place at ML Contest for Chip Design with HLS

ML Contest for Chip Design with HLS

Machine Learning Contest for Chip Design with High-Level Synthesis



Code 1: Code snippet of the SYR kernel (Matrix Vector Product and Transpose) with its 8 pragmas starting with "#pragma".

Overview Data Discussion Leaderboard Rules

Overview

This competition is hosted by UCLA Vastlab, UCLA DM lab, and AMD.

Please sign-up on our official sign-up sheet to be eligible to receive prizes

High-level synthesis (HLS) aims to raise the abstraction level in hardware design, enabling the design of domain-specific accelerators (DSAs) like field-programmable gate arrays (FPGAs) using C/C++ instead of hardware description languages (HDLs). The figure on the right shows an example of HLS designs, where the C++ code defines the functionality and compiler directives in the form of pragmas play a crucial role in modifying the microarchitecture within the HLS framework which determine the performance (e.g., latency and resources utilization rate) of the hardware.

Competition Host
Zongyue Qin

Prizes & Awards
Kudos
Does not award Points or Medals

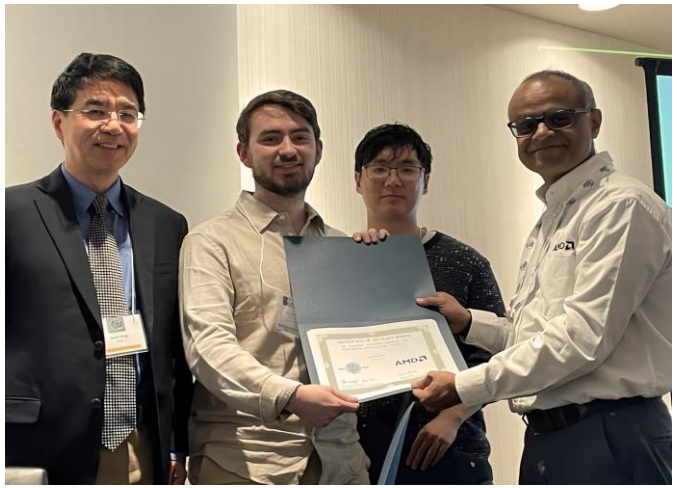
Participation
170 Entrants
51 Participants
27 Teams
608 Submissions

Overview Data Discussion Leaderboard Rules

#	△	Team	Members	Score
1	—	Emmet Murphy		1.26033

$$\text{Minimize} \rightarrow \frac{\sum_t \text{RMSE}(t)}{\text{F1} + 10^{-6}}$$

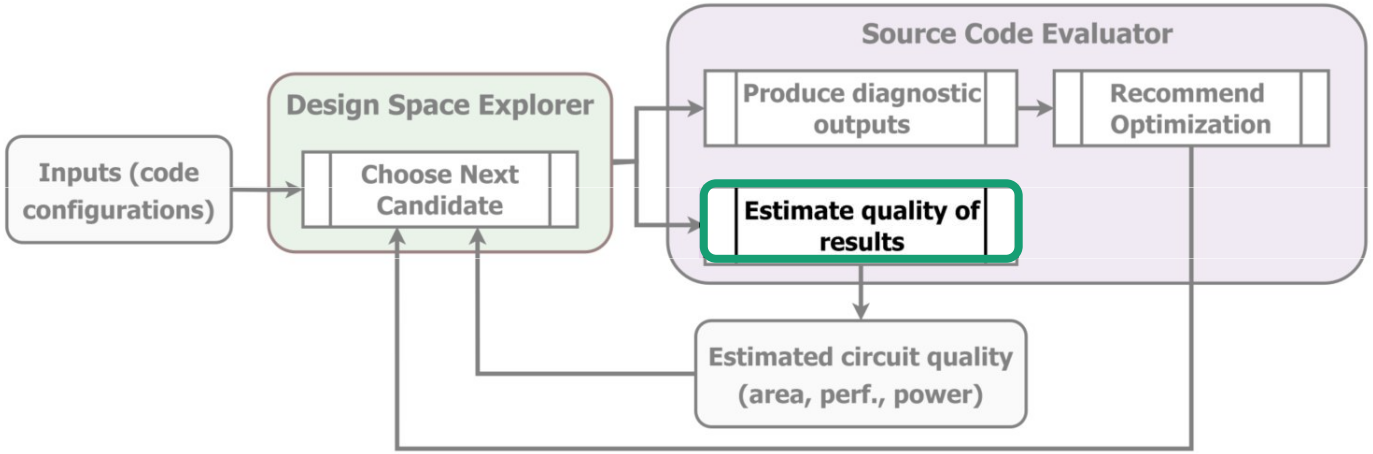
1. **1.26 → Balor**
2. 1.72 (+37%)
3. 1.99 (+58%)



Murphy and Josipović. Balor: HLS Source Code Evaluator Based on Custom Graphs and Hierarchical GNNs. ICCAD 2024.

Where Do We Go from Here?

- Balor: an **HLS source code evaluator**
  - Information-dense, **HLS-tailored** graphs
  - Local and hierarchical **GNNs**
  - **Increased estimation accuracy** and reduced computational cost



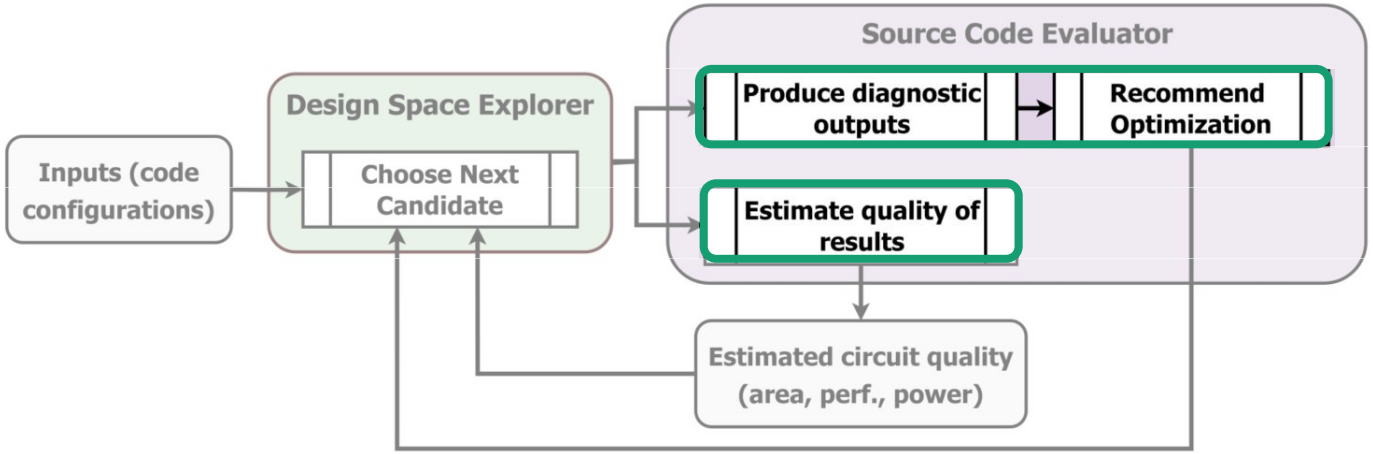
GNNs for accurate quality of results estimation directly from the source code

How to **generalize** to different HLS tools, FPGAs, unseen kernels,…?

Murphy and Josipović. Balor: HLS Source Code Evaluator Based on Custom Graphs and Hierarchical GNNs. ICCAD 2024.

Where Do We Go from Here?

- Balor: an **HLS source code evaluator**
  - Information-dense, **HLS-tailored** graphs
  - Local and hierarchical **GNNs**
  - **Increased estimation accuracy** and reduced computational cost



How to **provide insights** that guide design space exploration?

GNNs for accurate quality of results estimation directly from the source code

How to **generalize** to different HLS tools, FPGAs, unseen kernels,…?

Murphy and Josipović. Balor: HLS Source Code Evaluator Based on Custom Graphs and Hierarchical GNNs. ICCAD 2024.

Bridging the Gap Between Software and Hardware

SW

How to make hardware design accessible to non-experts?

How to automatically extract parallelism from software code?

How to avoid hardware-level functional verification?

How to understand and account for hardware implementation details?

HW

GNNs for accurate quality of results estimation directly from the source code

Murphy and Josipović. Balor: HLS Source Code Evaluator Based on Custom Graphs and Hierarchical GNNs. ICCAD 2024.

Bridging the Gap Between Software and Hardware

SW

How to make hardware design accessible to non-experts?

How to automatically extract parallelism from software code?

How to avoid hardware-level functional verification?

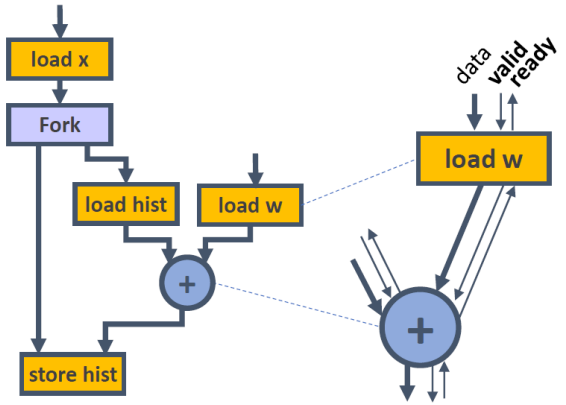
How to understand and account for hardware implementation details?

HW

Typical software features prevent **hardware parallelism**

```
for (i=0; i<N; i++) {
   hist[x[i]] = hist[x[i]] + weight[i];
}
```

Irregular memory access patterns



Features of **OoO superscalar processors** (e.g., speculation) → up to **14.9X faster** than Vivado HLS circuits

Dynamatic: an open-source HLS compiler that generates dynamically scheduled circuits from C/C++

Josipović, Ghosal, and Ienne. Dynamically Scheduled High-Level Synthesis. FPGA 2018 **Best Paper Award Nominee**
Josipović, Sheikhha, Guerrieri, Ienne, and Cortadella. Buffer Placement and Sizing for High-Performance Dataflow Circuits. FPGA 2020 **Best paper award**

Bridging the Gap Between Software and Hardware

SW

How to make hardware design accessible to non-experts?

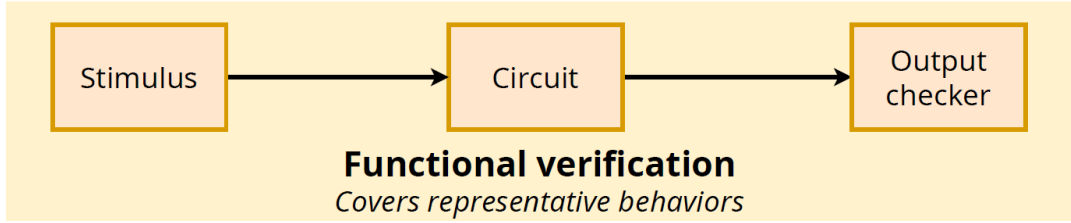
How to automatically extract parallelism from software code?

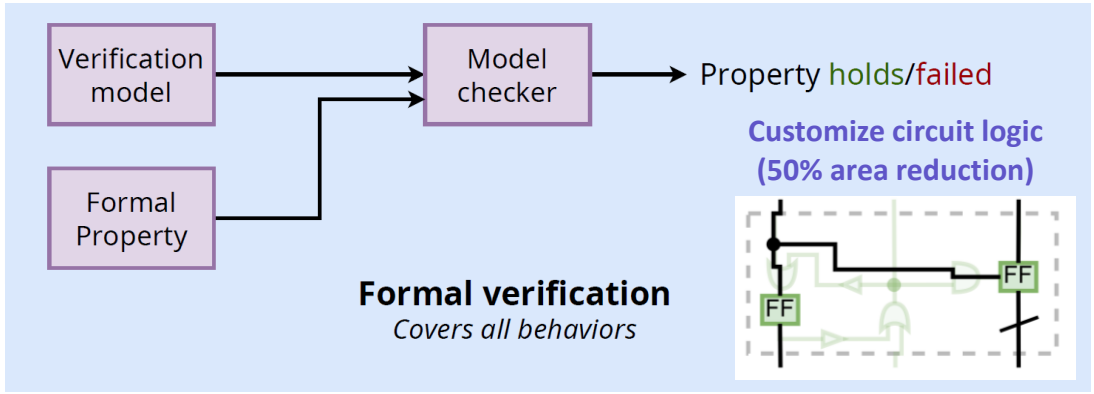
How to avoid hardware-level functional verification?

How to understand and account for hardware implementation details?

HW

Functional verification of circuits using hardware simulation
→ inefficient, limited, non-exhaustive





Formal verification for HLS: prove HLS correctness and improve HLS-generated circuits

Xu, Murphy, Cortadella, and Josipović. Eliminating excessive dynamism of dataflow circuits using model checking. FPGA 2023.
Elakhras, Guerrieri, Erhart, Ienne, and Josipović. ElasticMiter: Formally Verified Dataflow Circuit Rewrites. ASPLOS 2025

Bridging the Gap Between Software and Hardware

SW

How to make hardware design accessible to non-experts?

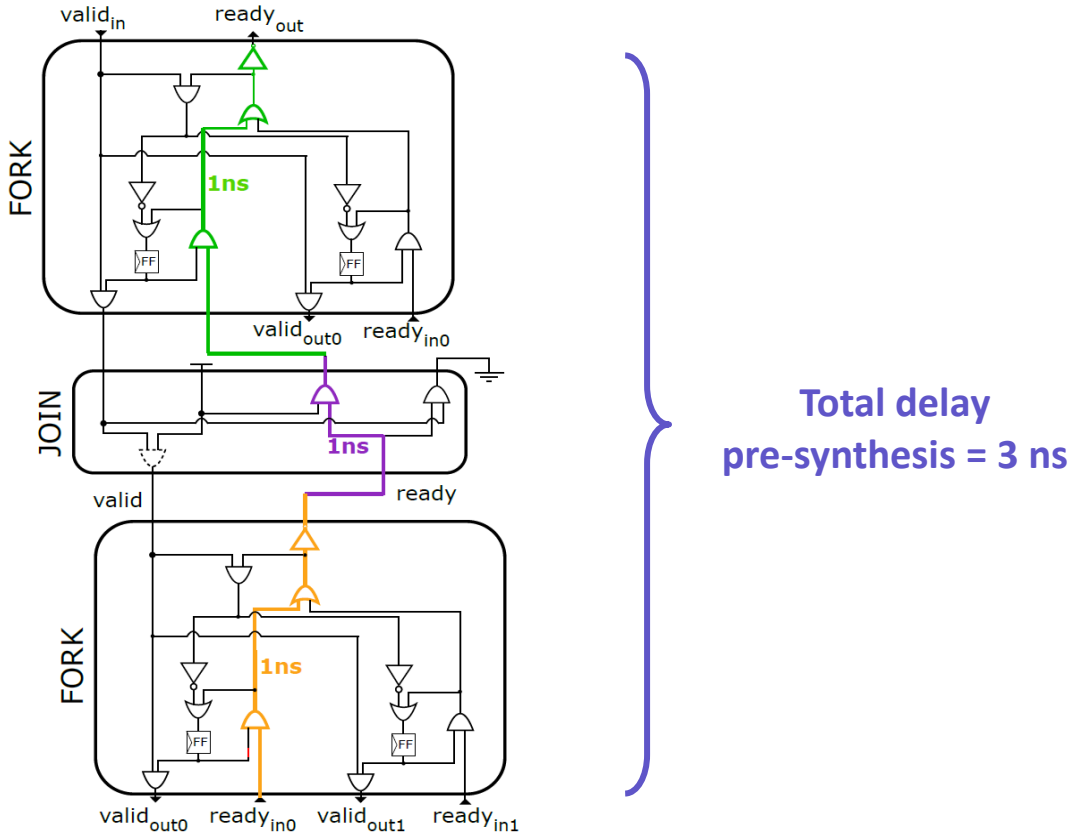
How to automatically extract parallelism from software code?

How to avoid hardware-level functional verification?

How to understand and account for hardware implementation details?

HW

Standard pipelining (register placement) is **unaware of circuit transformations** during logic synthesis and technology mapping



Rizzi, Guerrieri, and Josipović. An Iterative Method for Mapping-Aware Frequency Regulation in Dataflow Circuits. DAC 2023

Wang, Rizzi, and Josipović. MapBuf: Simultaneous Technology Mapping and Buffer Insertion for HLS Performance Optimization. ICCAD 2023 **Best Paper Award Nominee**

Bridging the Gap Between Software and Hardware

SW

How to make hardware design accessible to non-experts?

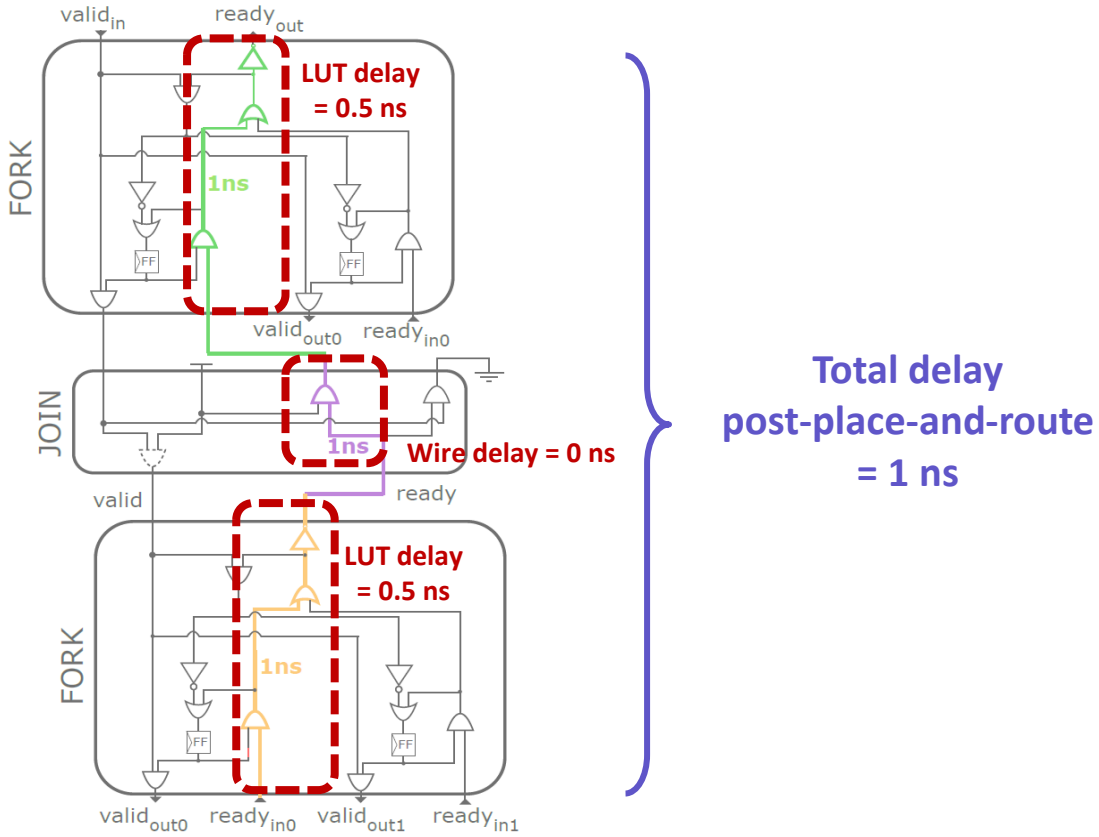
How to automatically extract parallelism from software code?

How to avoid hardware-level functional verification?

How to understand and account for hardware implementation details?

HW

Standard pipelining (register placement) is **unaware of circuit transformations** during logic synthesis and technology mapping



Implementation-aware compiler optimizations for fast and small circuits

Rizzi, Guerrieri, and Josipović. An Iterative Method for Mapping-Aware Frequency Regulation in Dataflow Circuits. DAC 2023

Wang, Rizzi, and Josipović. MapBuf: Simultaneous Technology Mapping and Buffer Insertion for HLS Performance Optimization. ICCAD 2023 **Best Paper Award Nominee**

Bridging the Gap Between Software and Hardware

FPGA implementation (synthesis, placement, and routing) is **orders of magnitude slower** than software compilation

SW

How to make hardware design accessible to non-experts?

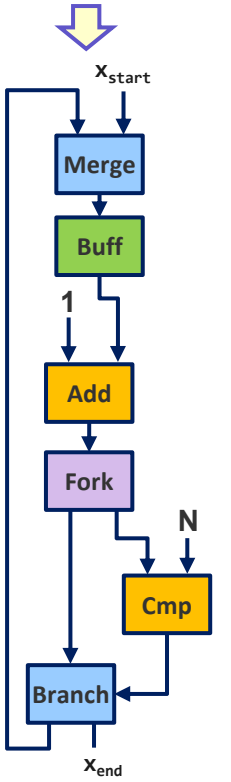
How to automatically extract parallelism from software code?

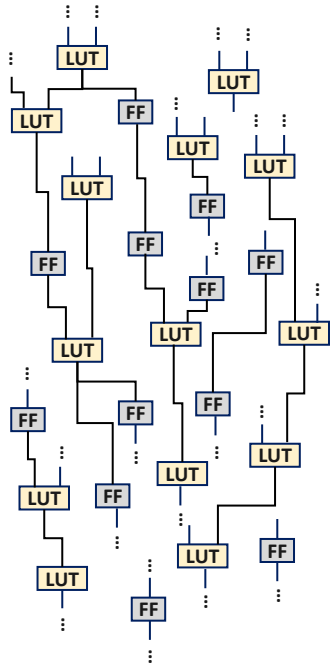
How to avoid hardware-level functional verification?

How to understand and account for hardware implementation details?

HW

```
do x++
    while (x<N)
return x
```





To place: 151 LUTs, 124 FFs
To route: 1299 wires (bits)

Slow (mins-hours) ☹

Guerrieri, Guha, Lavin, Hung, Josipović, and Ienne. DynaRapid: Fast-Tracking From C to Routed Circuits. FPL'24. **Best Paper Award**

Bridging the Gap Between Software and Hardware

SW

How to make hardware design accessible to non-experts?

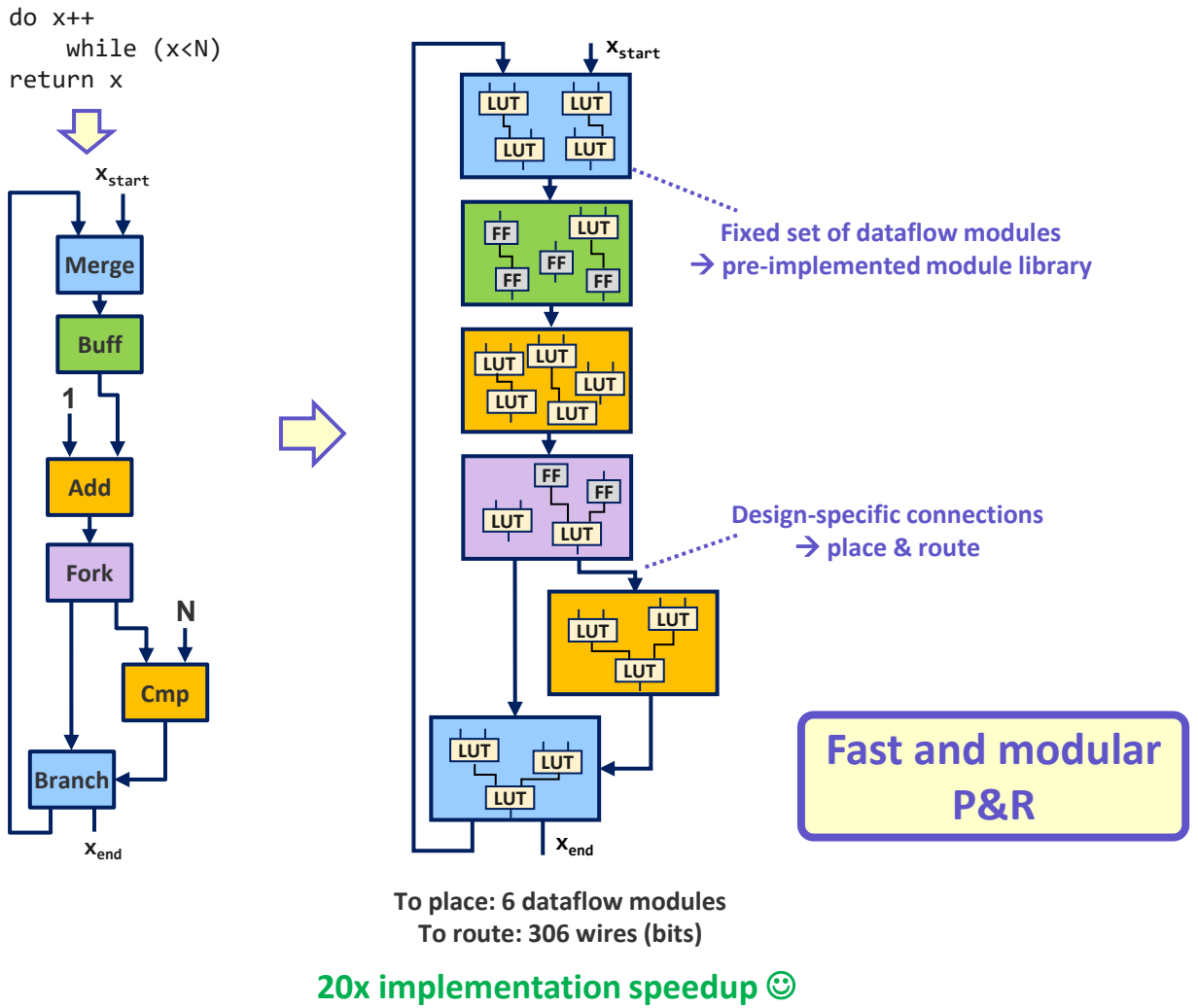
How to automatically extract parallelism from software code?

How to avoid hardware-level functional verification?

How to understand and account for hardware implementation details?

HW

FPGA implementation (synthesis, placement, and routing) is **orders of magnitude slower** than software compilation



Guerrieri, Guha, Lavin, Hung, Josipović, and Ienne. DynaRapid: Fast-Tracking From C to Routed Circuits. FPL'24. **Best Paper Award**

Bridging the Gap Between Software and Hardware

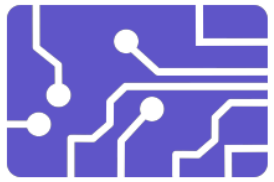
High-level abstractions

programming languages,
software applications

Hardware compilers

formal methods,
electronic design automation

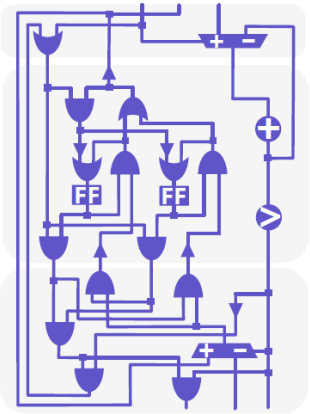
Hardware design



systems, digital design,
computer architecture

```
for (j = 0; j < 10; j++) {
   float x = 0.0;
   for (i = 0; i < 10; i++)
       x += data[i][j];
    mean[j] = x / float_n;
}

for (j = 0; j < 10; j++) {
   float x = 0.0;
   for (i = 0; i < 10; i++)
       x += (data[i][j] - mean[j]) *
(data[i][j] - mean[j]);
   x /= float_n;
   x = x*x;
   stdev[j] = x;
}
```



Make hardware design
broadly accessible, fast, and reliable

DYNAMO: Digital Systems and Design Automation Group

Research group:

dynamo.ethz.ch

Balor QoR estimator

github.com/ETHZ-DYNAMO/balor

Thanks! ☺