Loom: A Compiler and Hardware Generator for Heterogeneous Reconfigurable Systems

When Graphs and Streams Are First-Class Citizens

Sihao Liu, Tony Nowatzki

UCLA | CDSC Annual Review | February 2026

The Representation Problem

Today: Fragmented Representations

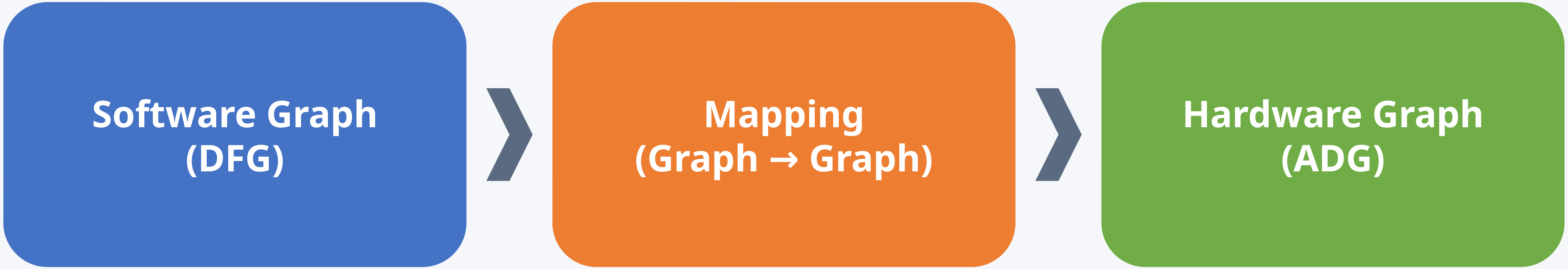
HLS	C → opaque RTL
CGRA	ad-hoc DFG → fixed mapper
Verification	separate testbench
DSE	manual re-integration

Key Observation

SW kernels	= dataflow graphs
HW architectures	= resource graphs
Mapping	= graph → graph association

Make graphs first-class citizens → accelerator compilation = graph transformations

The Graph-First Thesis



Software Graph (DFG)

Mapping (Graph → Graph)

Hardware Graph (ADG)

Handshake + Dataflow
MLIR dialect

Placement + routing +
resource assignment

Fabric MLIR dialect
PEs, switches, memories

SW Optimization

Graph mutation preserving execution semantics

HW Optimization

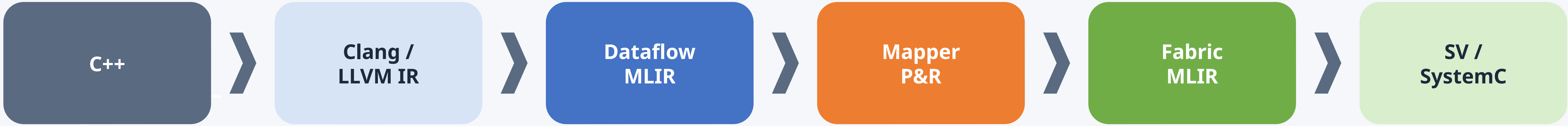
Topology transformation (tile replication, reshaping)

SW-HW Mapping

Constrained graph-to-graph association

No prior framework treats both SW and HW as native graph IRs in one compiler

Loom Full-Stack Pipeline



C++ ❯ Clang / LLVM IR ❯ Dataflow MLIR ❯ Mapper P&R ❯ Fabric MLIR ❯ SV / SystemC

Software Frontend

C++ with pragmas → Clang

SCF → Handshake → Dataflow MLIR

DFG: streaming ops + loop deps

Mapper P&R

Placement on ADG tiles

Routing through switches

config_mem bitstream output

Hardware Backend

ADGBuilder C++ API

Fabric MLIR → dual backend

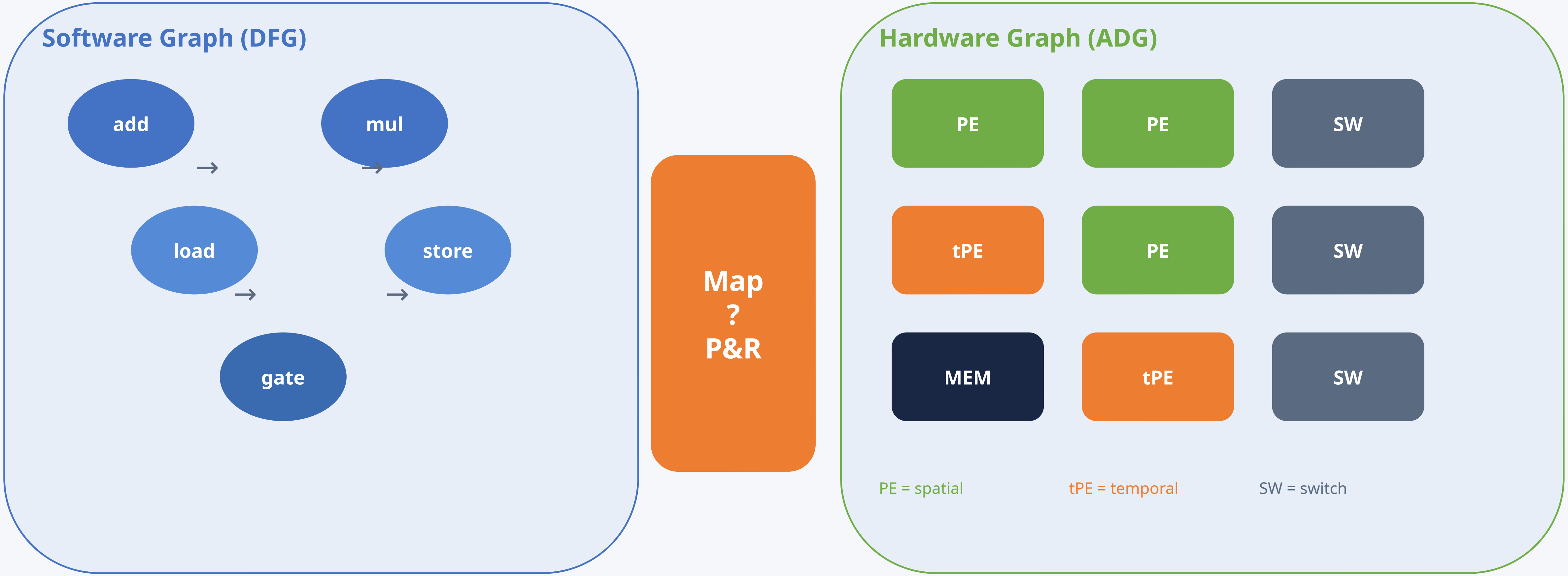
SystemC + SystemVerilog

Verification: ESI cosim + gem5 full-system simulation

2 MLIR dialects | Dual backend | 135+ kernels validated end-to-end

From C++ source to verified multi-core hardware in one framework

Heterogeneous Mapping Problem



Software Graph (DFG)

Hardware Graph (ADG)

PE	PE	SW
tPE	PE	SW
MEM	tPE	SW

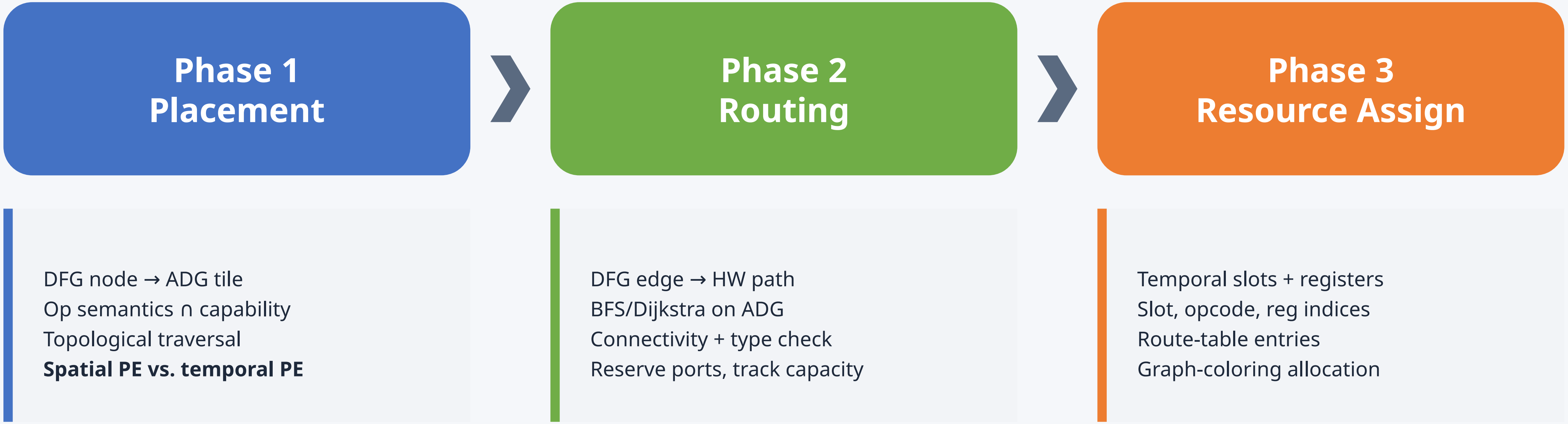
PE = spatial

tPE = temporal

SW = switch

Find constrained association from DFG onto ADG: placement + routing + resource assignment

Constraint-Driven Mapping



Phase 1 Placement

- DFG node → ADG tile
- Op semantics ∩ capability
- Topological traversal
- **Spatial PE vs. temporal PE**

Phase 2 Routing

- DFG edge → HW path
- BFS/Dijkstra on ADG
- Connectivity + type check
- Reserve ports, track capacity

Phase 3 Resource Assign

- Temporal slots + registers
- Slot, opcode, reg indices
- Route-table entries
- Graph-coloring allocation

Repair: conflict → reroute → reassign → restart

Cost: placement + routing + utilization

6 hard constraint classes (C1-C6) — never relaxed

Three-phase place-and-route with iterative repair and 6 hard constraints

Tagged Types for Heterogeneity

How do multiple operations share one physical tile and interconnect?

Port Multiplexing

Multiple streams share one port, demuxed by tag

Instruction Dispatch

Tag matches → selects operation + routing

Memory Disambiguation

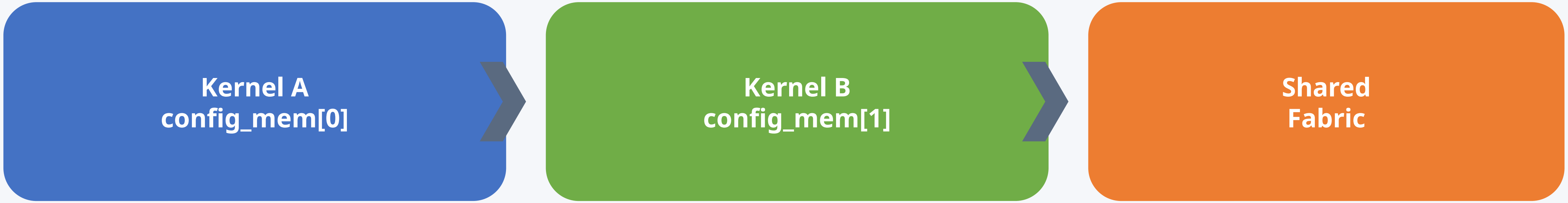
LSQ matches load/store transactions per tag

Route Multiplexing

Per-tag route-table entries on shared links

Not a separate dimension — tag assignment follows from placement/routing decisions

Multi-Kernel Scheduling



Kernel A
config_mem[0]

Kernel B
config_mem[1]

Shared
Fabric

Config-Mem Switching

Each kernel → independent config image
Host writes config between invocations
Delta update: overwrite only changed modules

Cross-Kernel Optimization

Share temporal PE slots across kernels
Disjoint spatial regions run concurrently
Complementary kernels maximize utilization

gem5-loom: full-system CGRA simulation in gem5 memory hierarchy
CPU dispatch | config loading | DMA | cache effects | workload-level evaluation

From single-kernel mapping to workload-level scheduling on shared fabric

Experimental Setup

Benchmark Suite: 135+ Kernels

- **Signal Proc**
  conv, FFT, FIR
- **Linear Alg**
  axpy, gemv, matmul
- **Stencil**
  Jacobi, Laplacian
- **ML**
  batchnorm, softmax
- **Sparse**
  scatter, gather

Architecture Configs

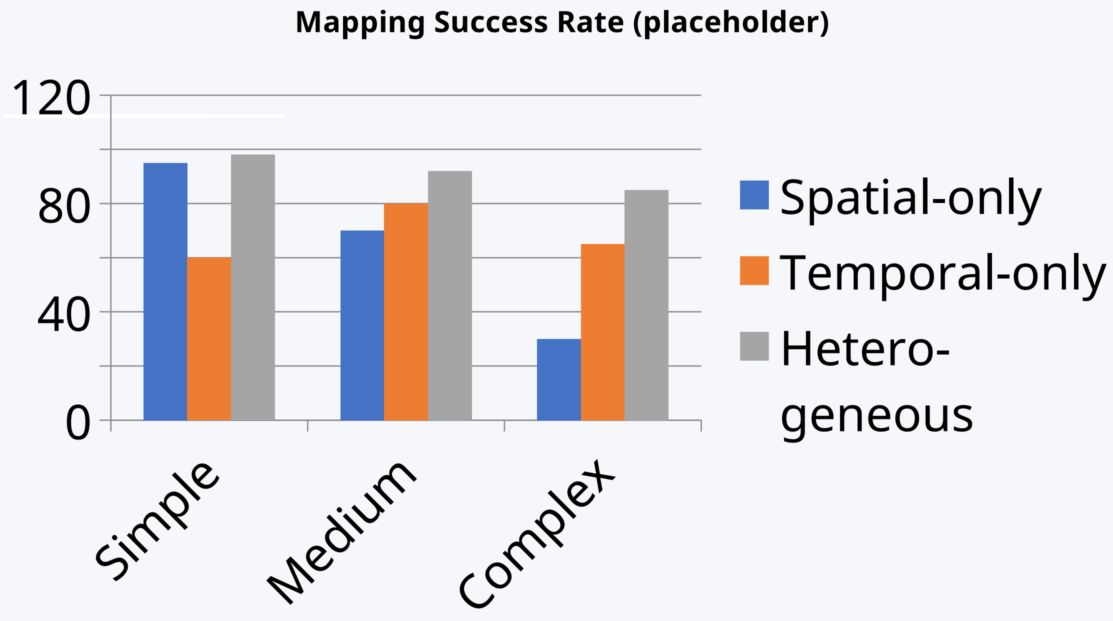
Spatial-only	4×4 mesh, all spatial PEs
Temporal-only	2×2 mesh, all temporal
Heterogeneous	4×4 spatial + temporal

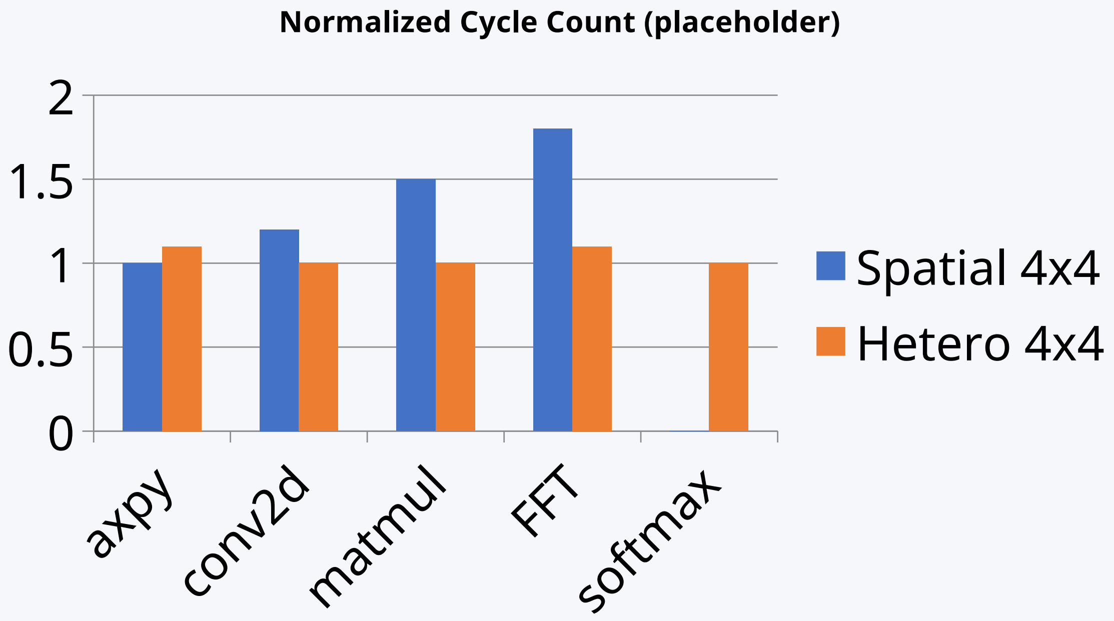
ESI cosim (bit-exact) | Metrics: success rate, PE utilization, routing congestion, cycle count, config footprint

Topology sweep: Mesh | Torus | DiagonalMesh | Custom

[Preliminary results — detailed data at the poster session]

Mapping Quality and Performance





Success: XX%

Utilization: +XX%

Config: XX words avg

Heterogeneous mapping wins: best of spatial throughput + temporal flexibility

Positioning in the Landscape

	CGRA-ME	OpenCGRA	Calyx	Loom
Input	Restricted C	DFG	Custom IR	**C/C++**
HW desc	Fixed XML	Fixed RTL	Custom	**ADGBuilder API**
IR	Custom	Custom	Custom	**MLIR native**
Scheduling	Modulo (ILP)	Modulo (SA)	Static	**Constraint P&R**
Heterogeneous	No	No	No	**Spatial + Temporal**
Retargetable	Limited	No	Partial	**Any topology**
Verification	External	External	External	**ESI + gem5**

Graph-native IR

Heterogeneous sched

Full-system verif

vs. HLS: Loom targets reconfigurable fabrics with transparent config; HLS produces opaque monolithic RTL

First framework with graph-native IR + heterogeneous scheduling + full verification

Summary and Future Directions

What we built

Dataflow + Fabric	Two MLIR dialects
Mapper	Constraint-driven P&R
Backends	SystemC + SystemVerilog
gem5-loom	Full-system simulation

The bigger thesis

- Graphs and streams as first-class citizens
- Complexity emerges from connectivity
- No fine-tuning HW or SW individually
- Different philosophy from HLS + classical CGRA

Open Research Directions

- ILP/SAT placement
- Auto-tiling
- Physical design closure
- Cross-layer DSE

135+ kernels | Dual backend | Retargetable to any topology

Thank you | sihao@cs.ucla.edu

Backup: Dialect Details

Dataflow Dialect

dataflow.stream	index generator + predicate
dataflow.carry	loop-carried dependency
dataflow.invariant	loop-invariant broadcast
dataflow.gate	stream alignment

Fabric Dialect

fabric.pe	compute / const / load-store
fabric.temporal_pe	time-multiplexed PE
fabric.switch	static / temporal routing
fabric.memory	storage + LSQ
fabric.fifo	buffering (bypassable)

Structure vs. Configuration

HW params (fixed at build) | Runtime config via config_mem (tags, predicates, routes)

Two MLIR dialects capture both software dataflow and hardware fabric structure

Backup: ADGBuilder API

Programmatic Hardware Construction in C++

```
ADGBuilder builder("my_cgra");
auto pe  = builder.newPE("alu").setLatency(1).addOp(...);
auto tpe = builder.newTemporalPE("shared").setNumInstruction(8);
auto sw  = builder.newSwitch("router").setPortCount(4, 4);
builder.buildMesh(4, 4, pe, sw, Topology::Mesh);
builder.exportSV("output/sv");   // SystemVerilog
builder.exportSysC("output/sysc"); // SystemC
```

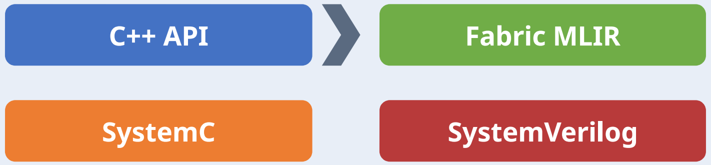
Features

- Single-source C++ API
- Mesh/Torus/Diagonal/Custom
- Clone-with-deduplication

Multi-Backend

- Fabric MLIR → backends
- SystemC: fast iteration
- SystemVerilog: synthesis

Pipeline



Single C++ API → any topology → dual-backend export

Backup: Constraints C1-C6

C1	Node	SW op semantics match HW tile capability
C2	Port/Type	Category match, value type, tag-width
C3	Route	Physical connectivity + directionality
C4	Capacity	Fan-in/out, temporal slots, memory queues
C5	Temporal	Tag uniqueness, slot validity, register bounds
C6	Config	Emitted config matches Fabric op spec

All 6 are hard constraints — never traded for cost optimization