AutoSA: A Polyhedral Compiler for High-Performance Systolic Arrays on FPGAs

Jie Wang, Licheng Guo, Jason Cong

University of California, Los Angeles

Outline

- **A Brief Background**
- Details about AutoSA Compilation Flow
- Demos
    - Matrix Multiplication
    - Convolutional Neural Network
- Auto-Tuning in AutoSA
- Using AutoBridge to Boost the Design Frequency

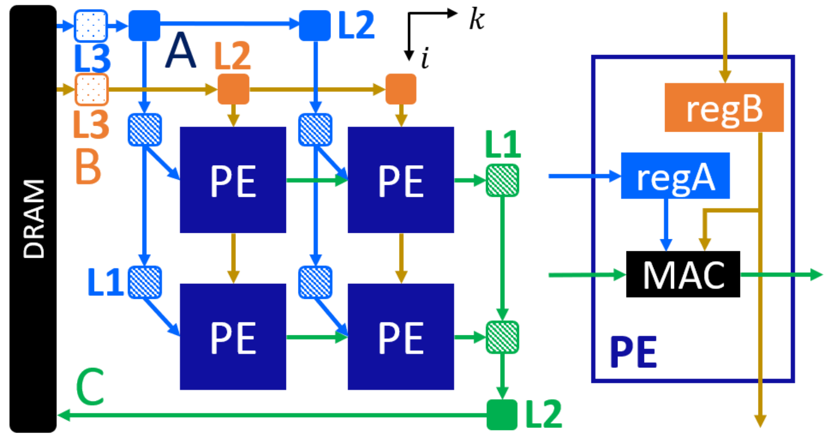
AutoSA: A Polyhedral Compiler for High-Performance Systolic Arrays on FPGAs

Wang, Jie, Licheng Guo, and Jason Cong. "AutoSA: A Polyhedral Compiler for High-Performance Systolic Arrays on FPGA." Proceedings of the 2021 ACM/SIGDA international symposium on Field-programmable gate arrays. 2021.

```
#pragma scop
for (int i = 0; i < M; ++i)
    for (int j = 0; j < N; ++j) {
S0:     C[i][j] = 0;
        for (int k = 0; k < K; ++k)
S1:         C[i][j] += A[i][k] * B[k][j];
    }
#pragma endscop
```

Input: C code

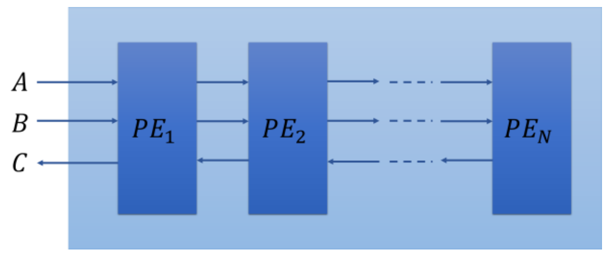
AutoSA

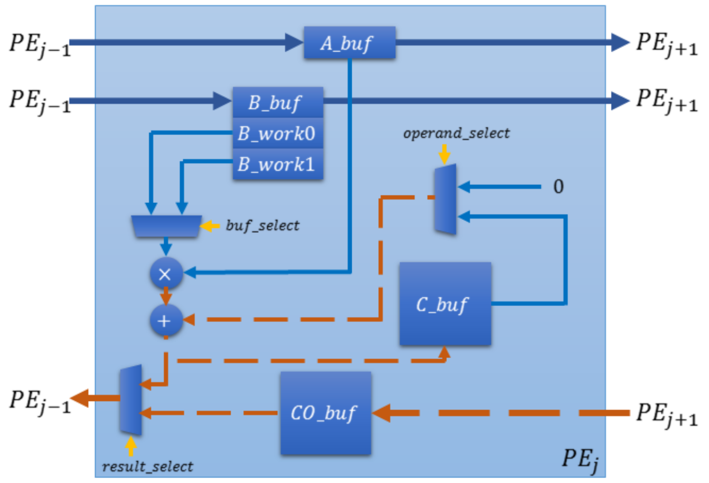


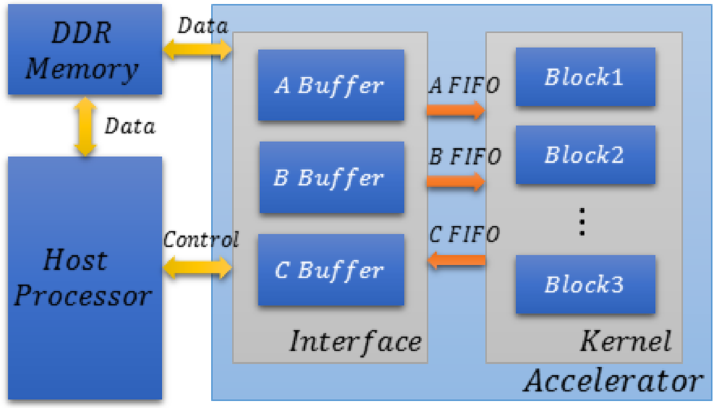
Output: Systolic array design in HLS C

Starting Point

- 2014 Summer Internship at UCLA
- Time: ~5 months
- A manual-designed RTL-based 1D systolic array for MM
- 198.1 GFLOPs on Xilinx VC709







Previous Work

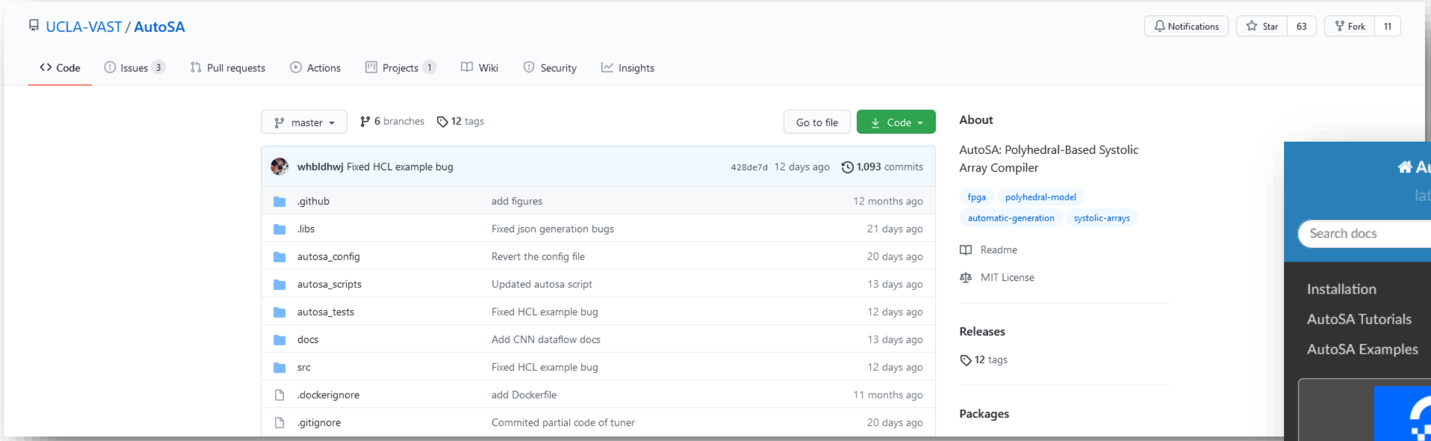
	MMAlpha '01	Bondhugula et al. '07	PolySA '18	SuSy '20	AutoSA '21
Generality Support applications as general as possible.					
Imperfectly Nested Loops	No	No	No	No	Yes
Multi-Statement	Yes	No	No	Yes	Yes
Performance Generate designs with performance as high as possible.					
Latency Hiding	No	No	Auto	Semi-Auto	Auto
Double Buffering	No	No	Auto	Semi-Auto	Auto
Productivity Shorten the development time as much as possible.					
Auto-Tuning	No	No	Yes	No	Yes

Outline

- A Brief Background
- **Details about AutoSA Compilation Flow**
- Demos
  - Matrix Multiplication
  - Convolutional Neural Network
- Auto-Tuning in AutoSA
- Using AutoBridge to Boost the Design Frequency

AutoSA is Open-Sourced

- Github: https://github.com/UCLA-VAST/AutoSA
- Document: https://autosa.readthedocs.io/en/latest/



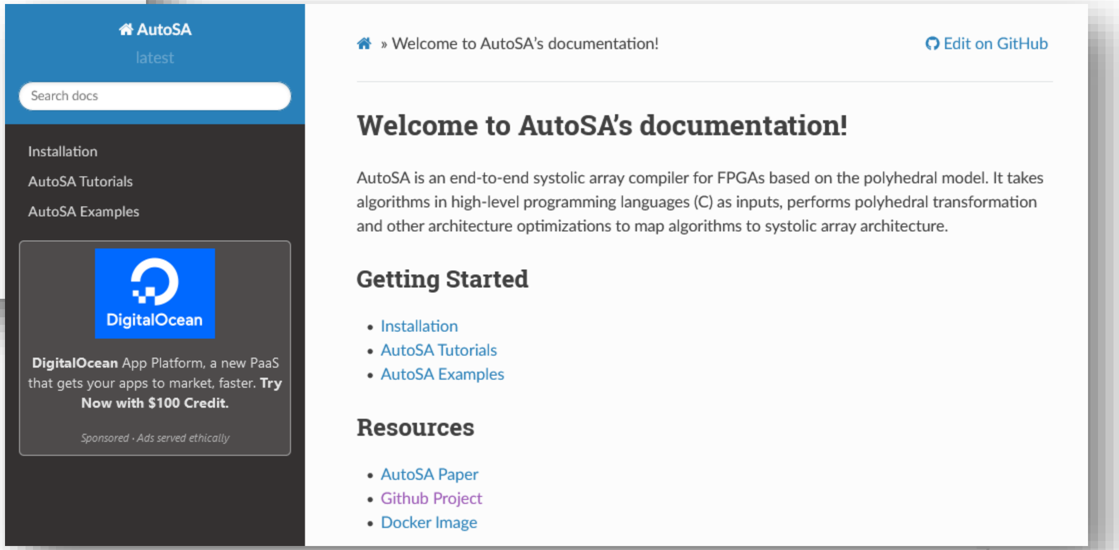
UCLA-VAST / AutoSA

whbldhwj Fixed HCL example bug 428de7d 12 days ago 1,093 commits

.github	add figures	12 months ago
.libs	Fixed json generation bugs	21 days ago
autosa_config	Revert the config file	20 days ago
autosa_scripts	Updated autosa script	13 days ago
autosa_tests	Fixed HCL example bug	12 days ago
docs	Add CNN dataflow docs	13 days ago
src	Fixed HCL example bug	12 days ago
.dockerignore	add Dockerfile	11 months ago
.gitignore	Commited partial code of tuner	20 days ago

About

AutoSA: Polyhedral-Based Systolic Array Compiler



DigitalOcean App Platform, a new PaaS that gets your apps to market, faster. **Try Now with $100 Credit.**

Sponsored · Ads served ethically

» Welcome to AutoSA's documentation! Edit on GitHub

Welcome to AutoSA's documentation!

AutoSA is an end-to-end systolic array compiler for FPGAs based on the polyhedral model. It takes algorithms in high-level programming languages (C) as inputs, performs polyhedral transformation and other architecture optimizations to map algorithms to systolic array architecture.

Getting Started

- Installation
- AutoSA Tutorials
- AutoSA Examples

Resources

- AutoSA Paper
- Github Project
- Docker Image

Current Capabilities

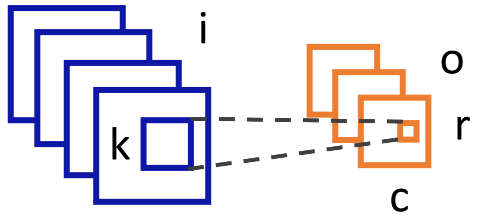
- Back-end
  - Xilinx HLS C (Mature, recommended)
  - Intel OpenCL (Experimental)
  - Catapult HLS C (Experimental)
- High-Performance designs on Xilinx platforms

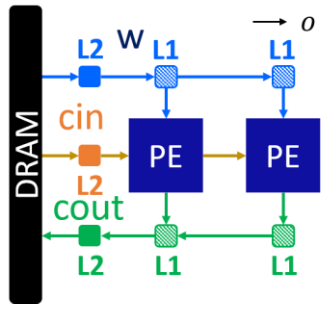
Matrix Multiplication	FP32	int16	Int8
MHz	300	250	300
TFLOPs	0.93	3.41	6.95

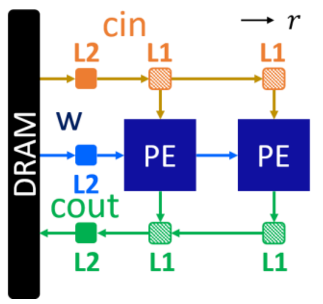
Board: Xilinx Alveo U250

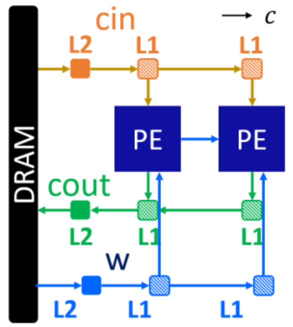
Current Capabilities

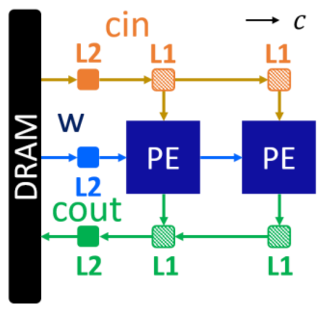
- **Architectural exploration** of different dataflows
- Example: 10 dataflows of CNN (https://autosa.readthedocs.io/en/latest/examples/cnn.html)

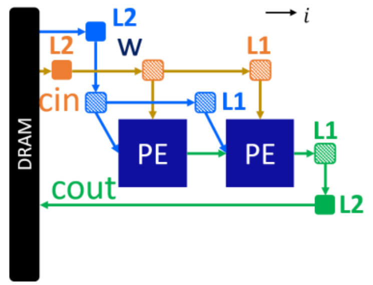


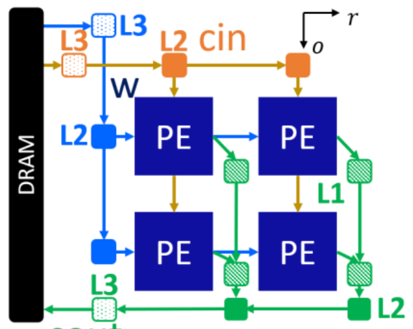


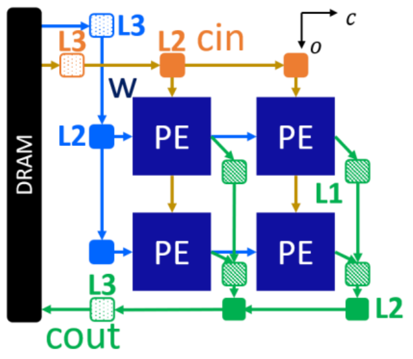


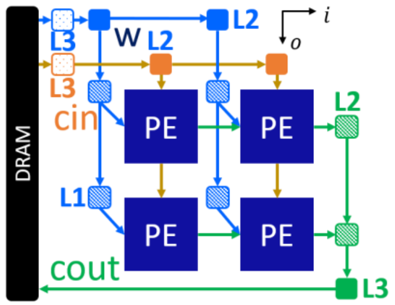


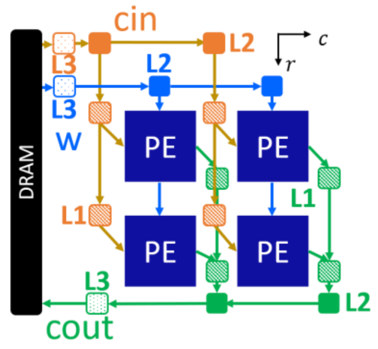


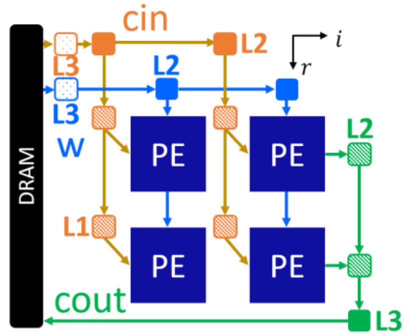






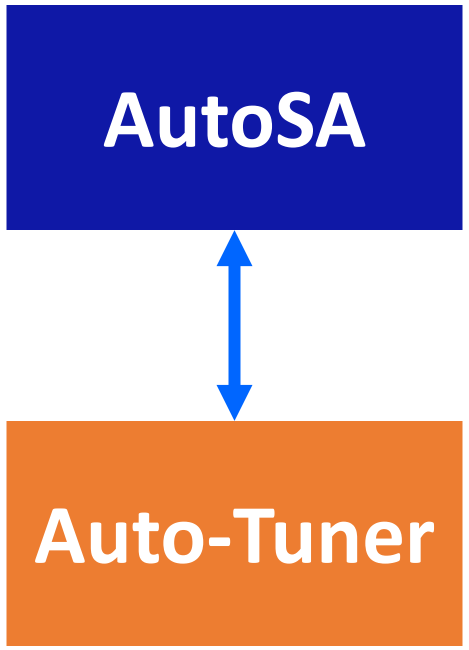






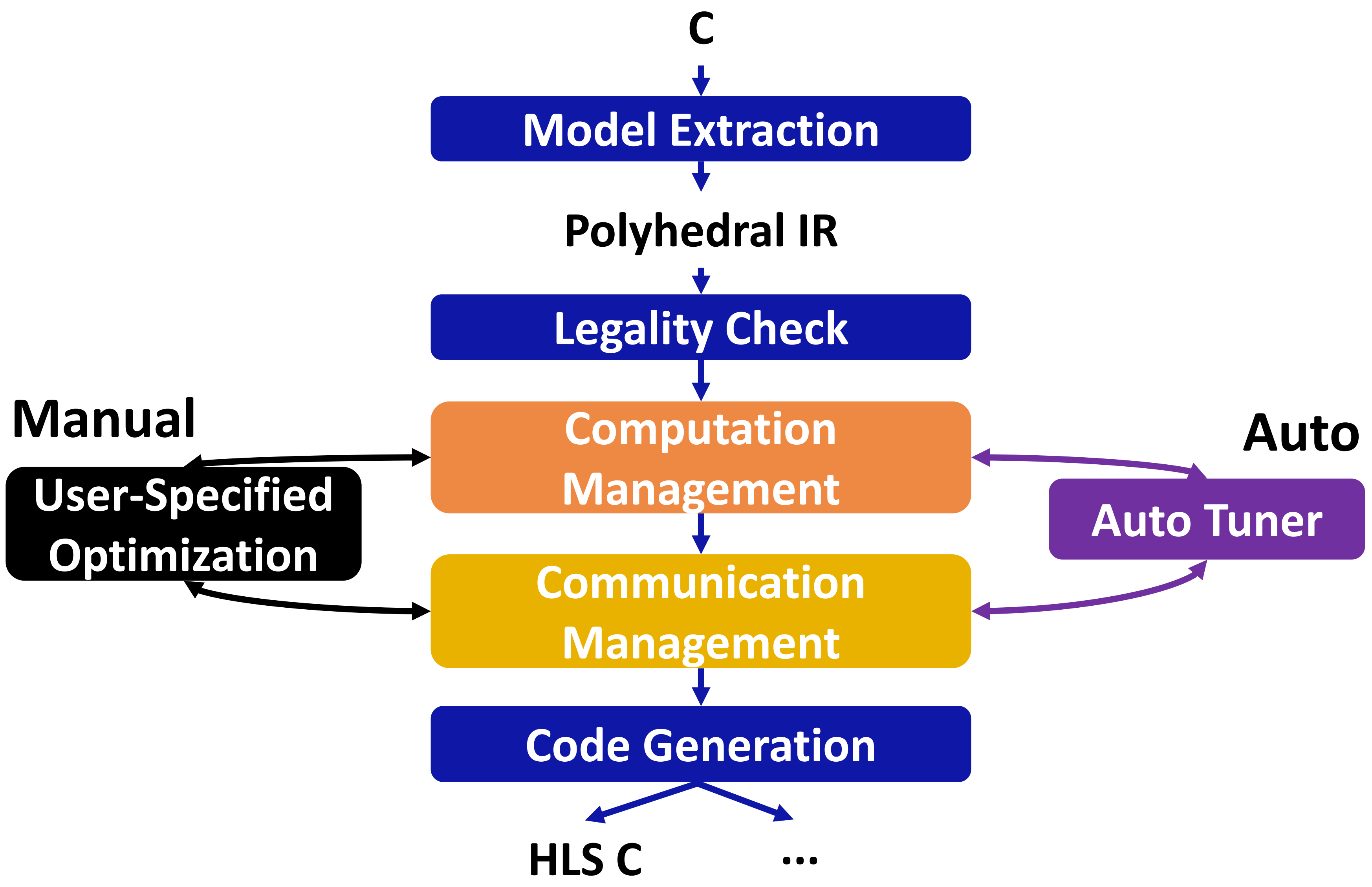
Current Capabilities

- Auto-tuning support



- Exhaustive search with pruning
  - mins to hours
- Genetic search
  - < 1 minute

Compilation Flow



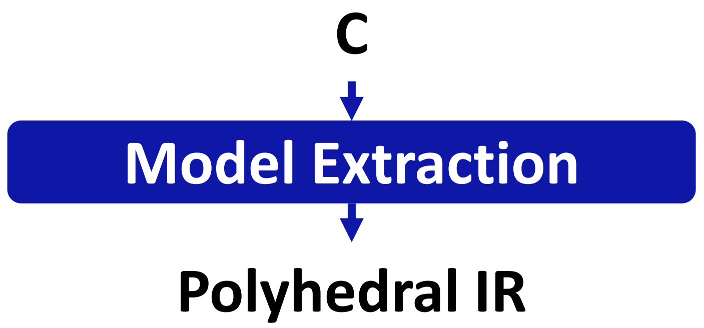
Compilation Flow

C

```
#pragma scop
for (int i = 0; i < M; ++i)
    for (int j = 0; j < N; ++j) {
S0:     C[i][j] = 0;
        for (int k = 0; k < K; ++k)
S1:         C[i][j] += A[i][k] * B[k][j];
    }
#pragma endscop
```

The input program of AutoSA

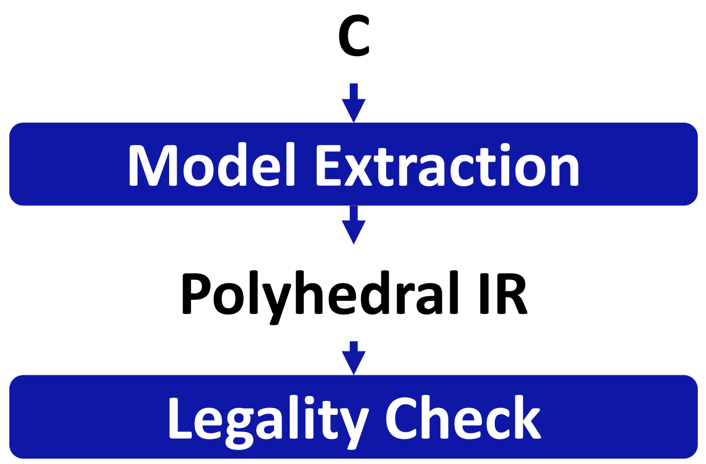
Compilation Flow



Polyhedral Model

- A **mathematical** compilation framework for **loop nest optimization**.
- Applicable to a **subset** of general loop programs (static control of parts (SCoP)).
- **Complete** and **robust** toolchains.

Compilation Flow



Legality Check

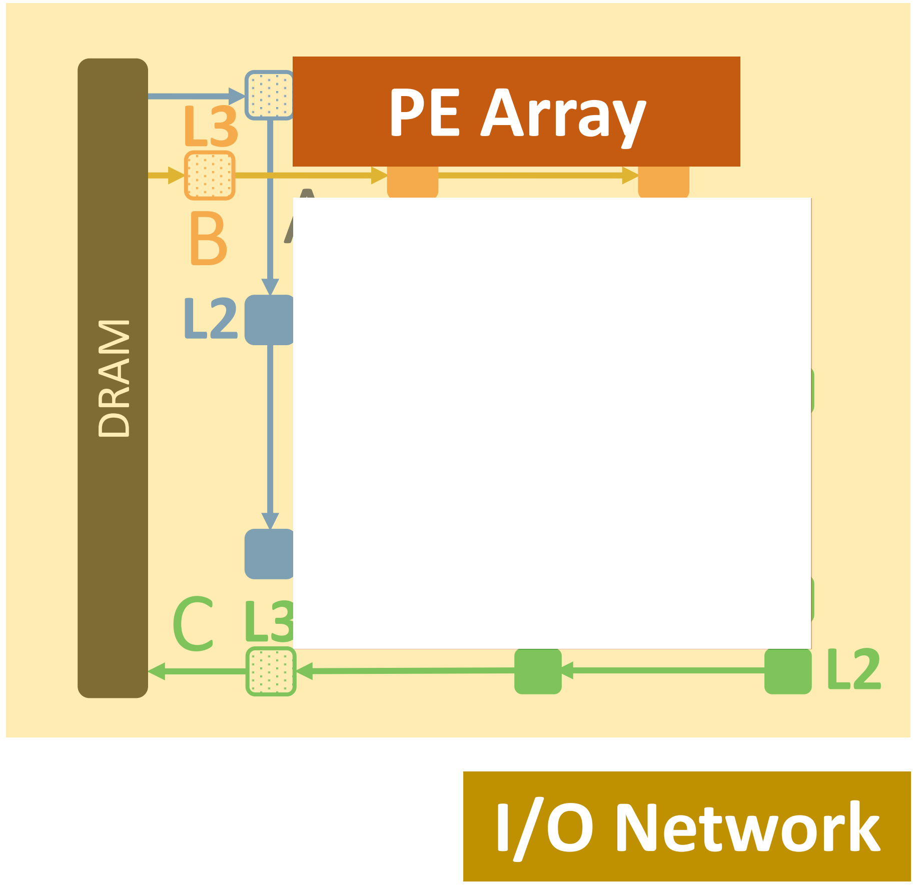
- All dependences are **uniform** dependences (with constant dependence distance)

  $\longrightarrow$ **Statically-scheduled** design

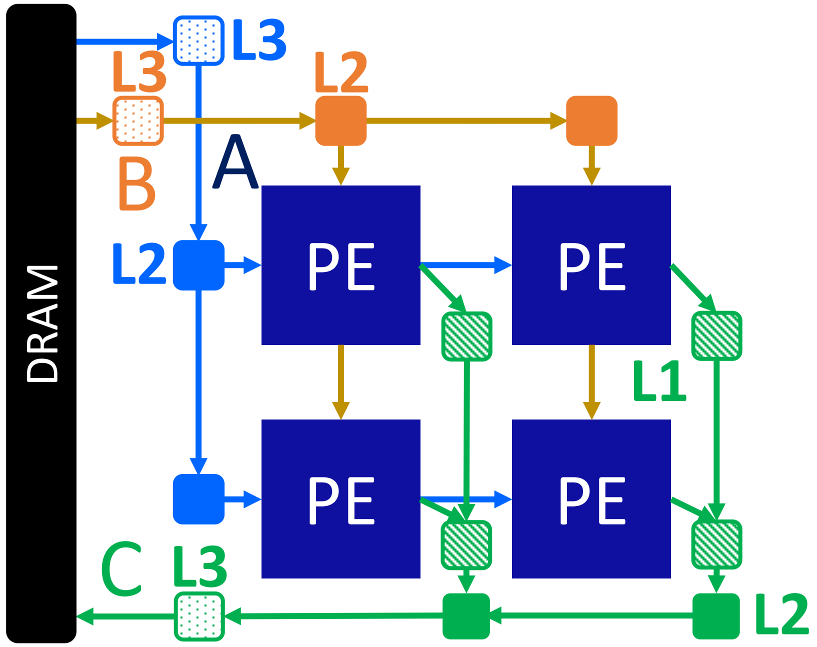
- Dependence distances on space loops (loops to be mapped to PE dimensions) to be **no greater than one**.

  $\longrightarrow$ **Local** communication

Compilation Flow



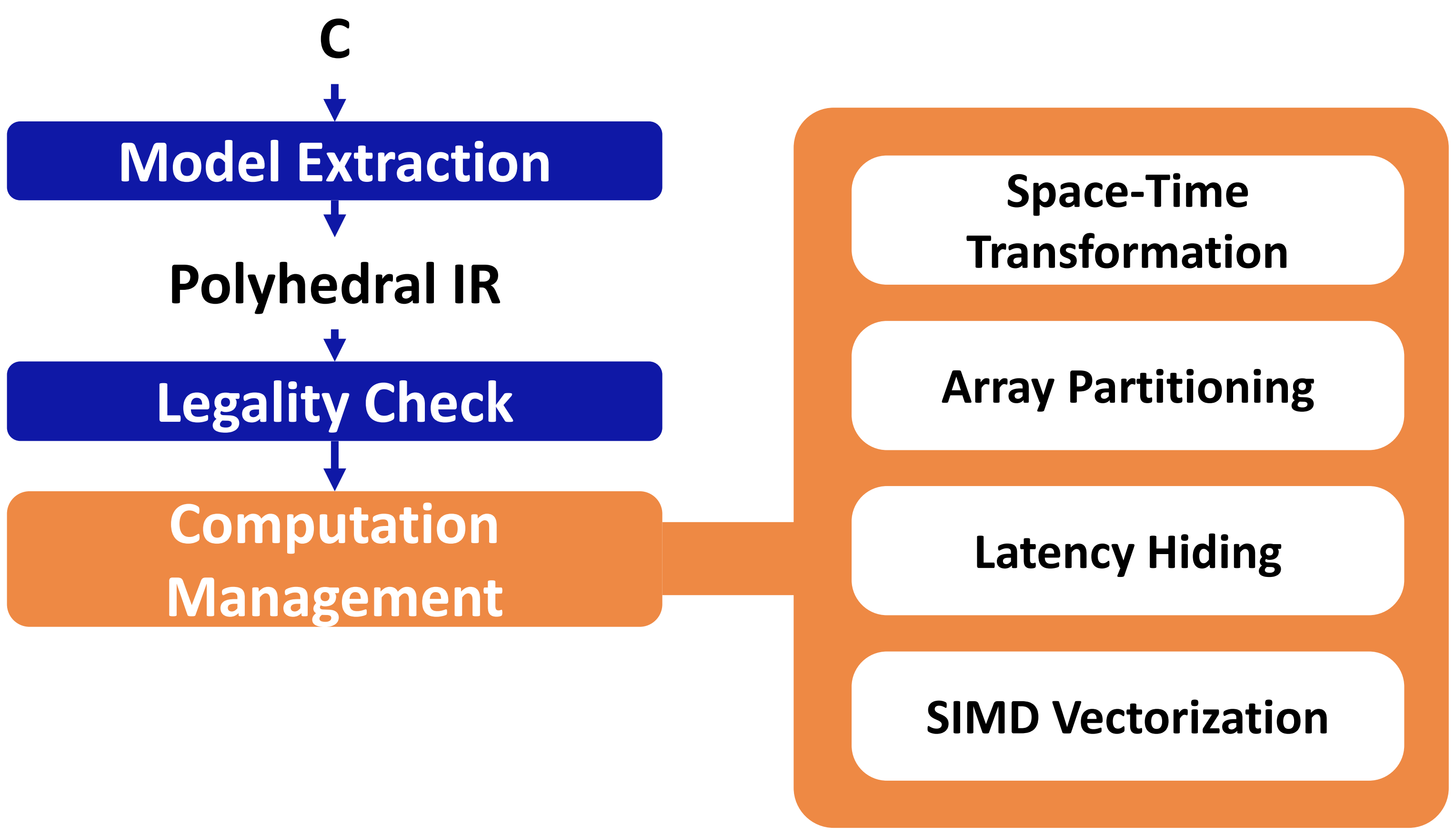
Compilation Flow



Computation Management

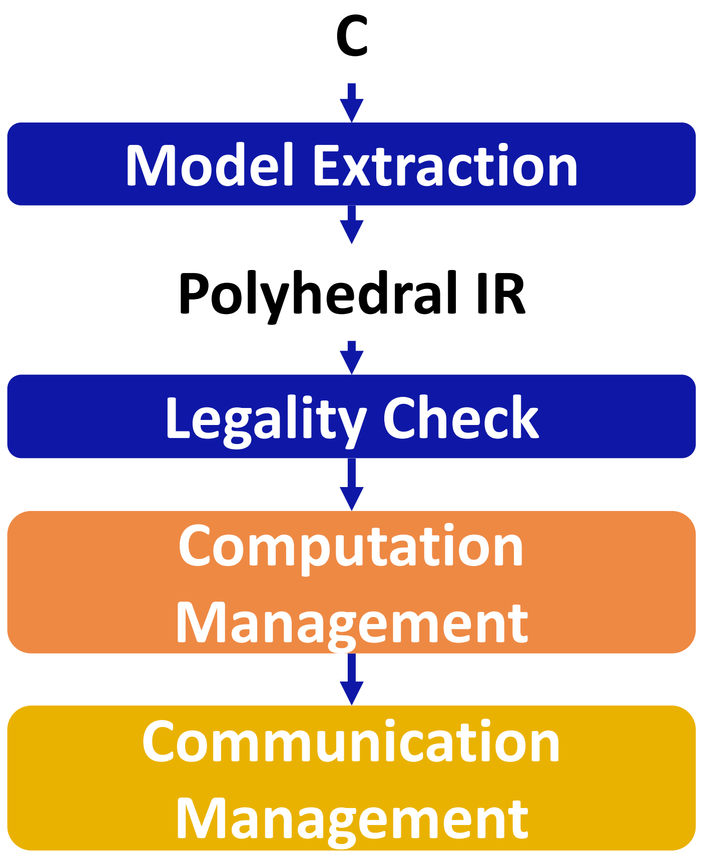
Communication Management

Compilation Flow



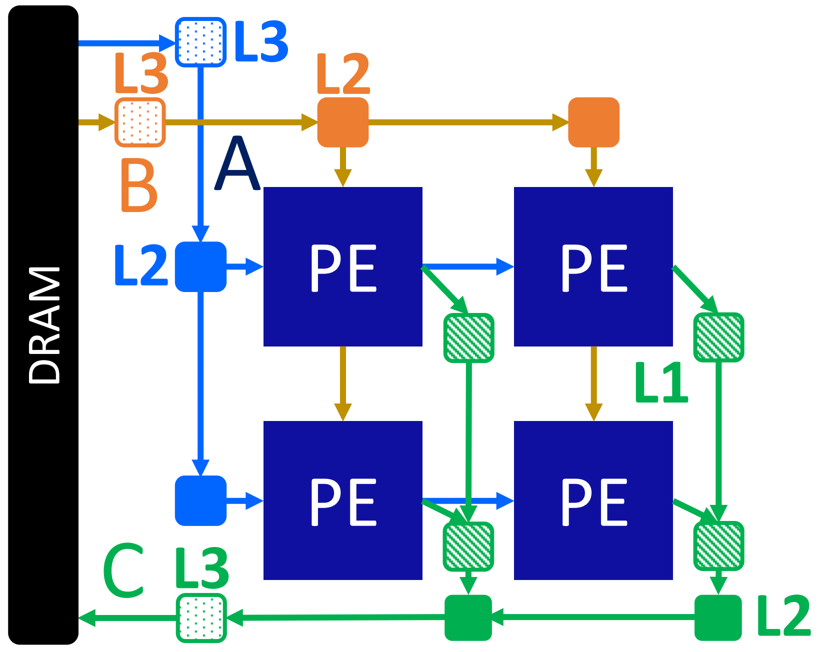
Construct and optimize the PE array

Compilation Flow



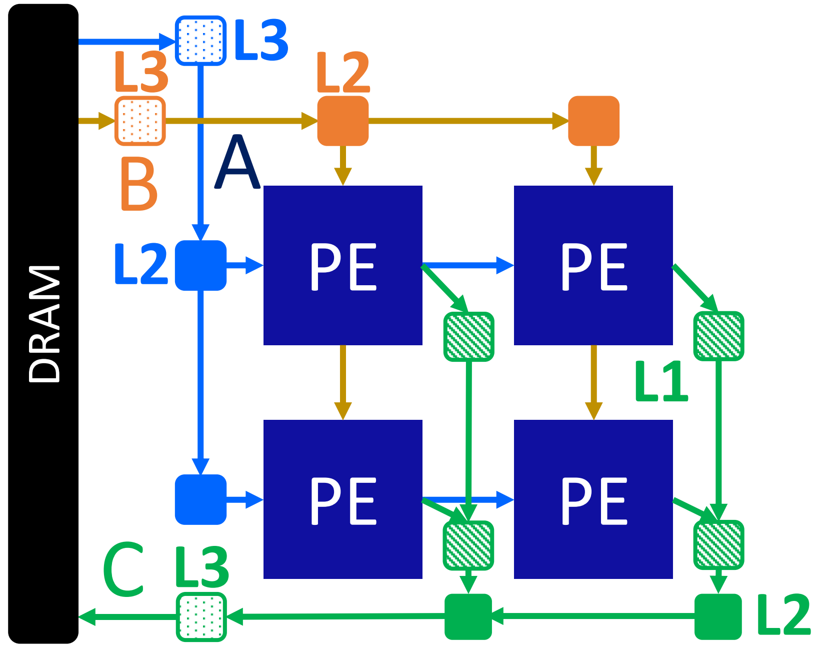
Construct and optimize the I/O network

Communication Management



- **What** hardware modules to be generated for the I/O network?
- **How** to generate the I/O network?
- **How** to **optimize** the I/O network?

Communication Management



What hardware modules to be generated for the I/O network?

- Inter-array communication modules
  - Data transfer logic inside PEs
  - PE interconnects
- Outer-array communication modules
  - I/O modules

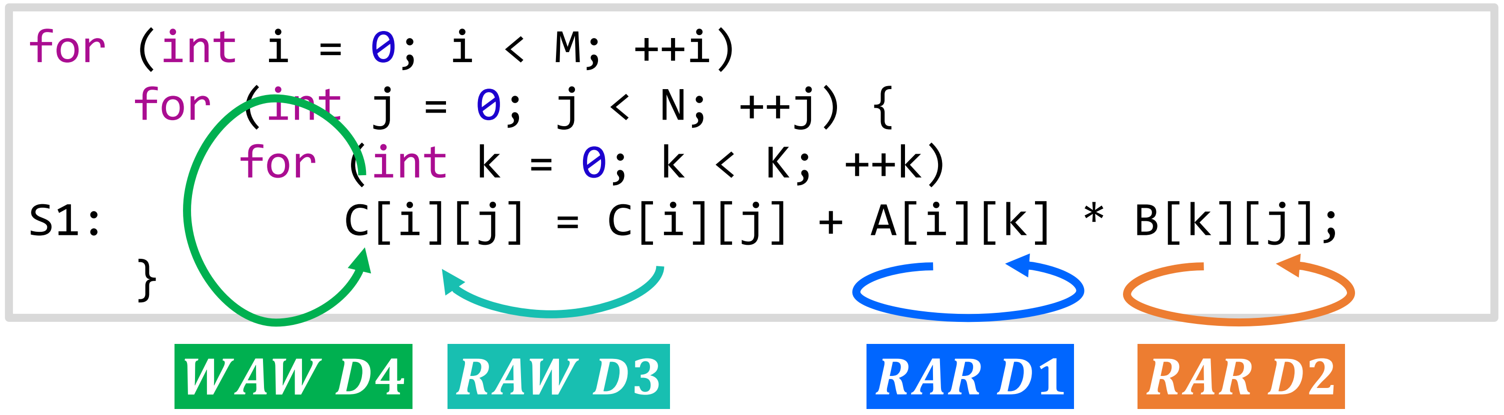
Communication Management

How to generate the I/O network?

- **Key idea**: **Data communication** is determined by the data dependences.

Read dependences (Read-After-Read, RAR)	$\longrightarrow$	Read-only data
Flow dependences (Read-After-Write, RAW)	$\longrightarrow$	Intermediate data
Output dependences (Write-After-Write, WAW)	$\longrightarrow$	Computation results

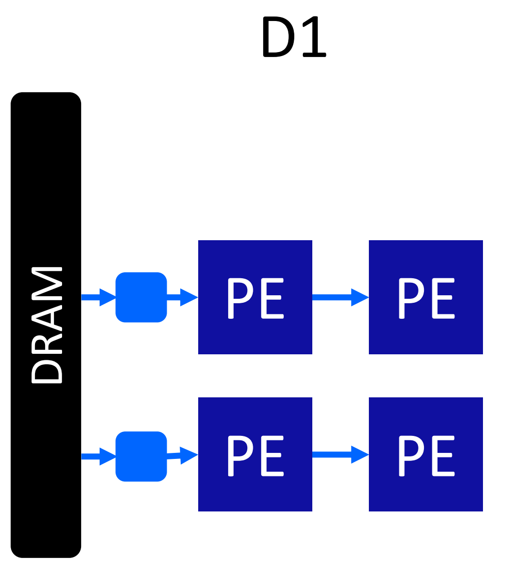
Communication Management

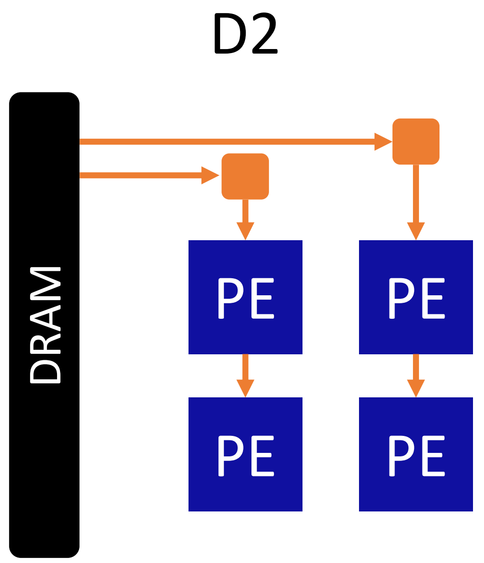


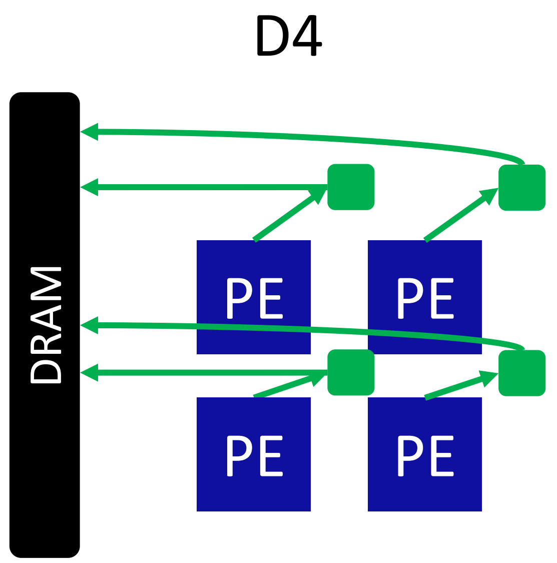
```
for (int i = 0; i < M; ++i)
    for (int j = 0; j < N; ++j) {
        for (int k = 0; k < K; ++k)
S1:         C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

Dependence	Dependence Type	Array Access	Dependence Distance

We omit the statement of array initialization for the sake of brevity.







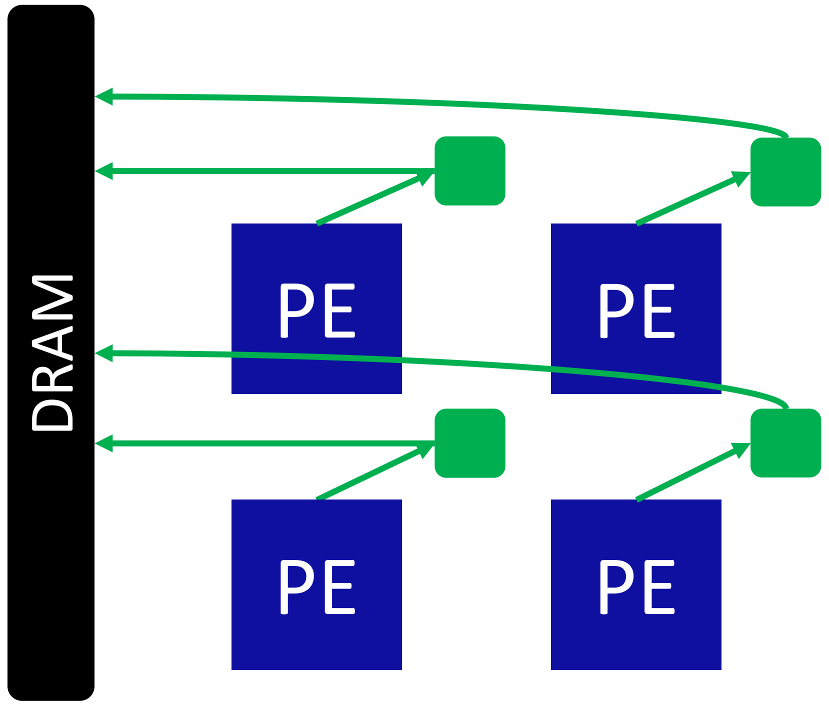
Dependence	Dependence Type	Array Access	Dependence Distance
D1	Read (RAR)	`A[i][k]`	(0,1,0)
D2	Read (RAR)	`B[k][j]`	(1,0,0)
D3	Flow (RAW)	`C[i][j]`	(0,0,1)
D4	Output (WAW)	`C[i][j]`	(0,0,1)

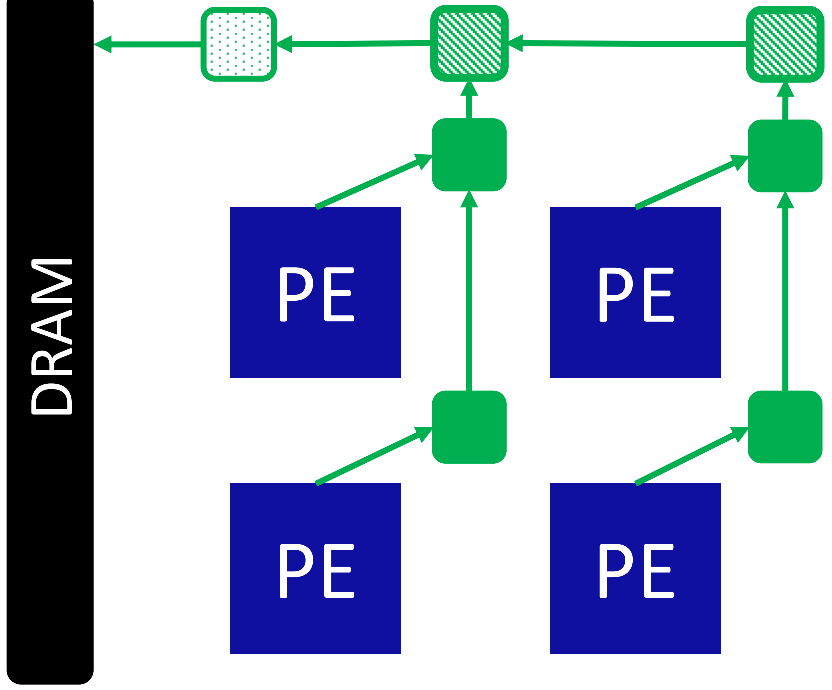
We omit the statement of array initialization for the sake of brevity.

Communication Management

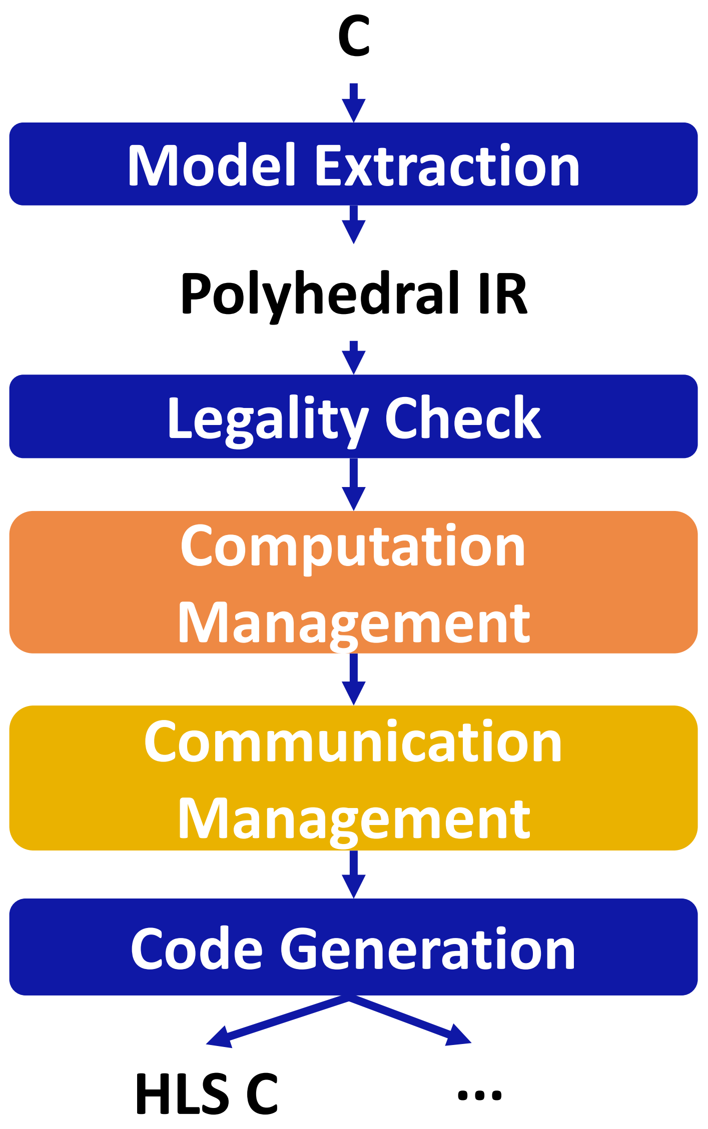
How to **optimize** the I/O network?

- I/O network localization
- Double buffering, data packing, etc.

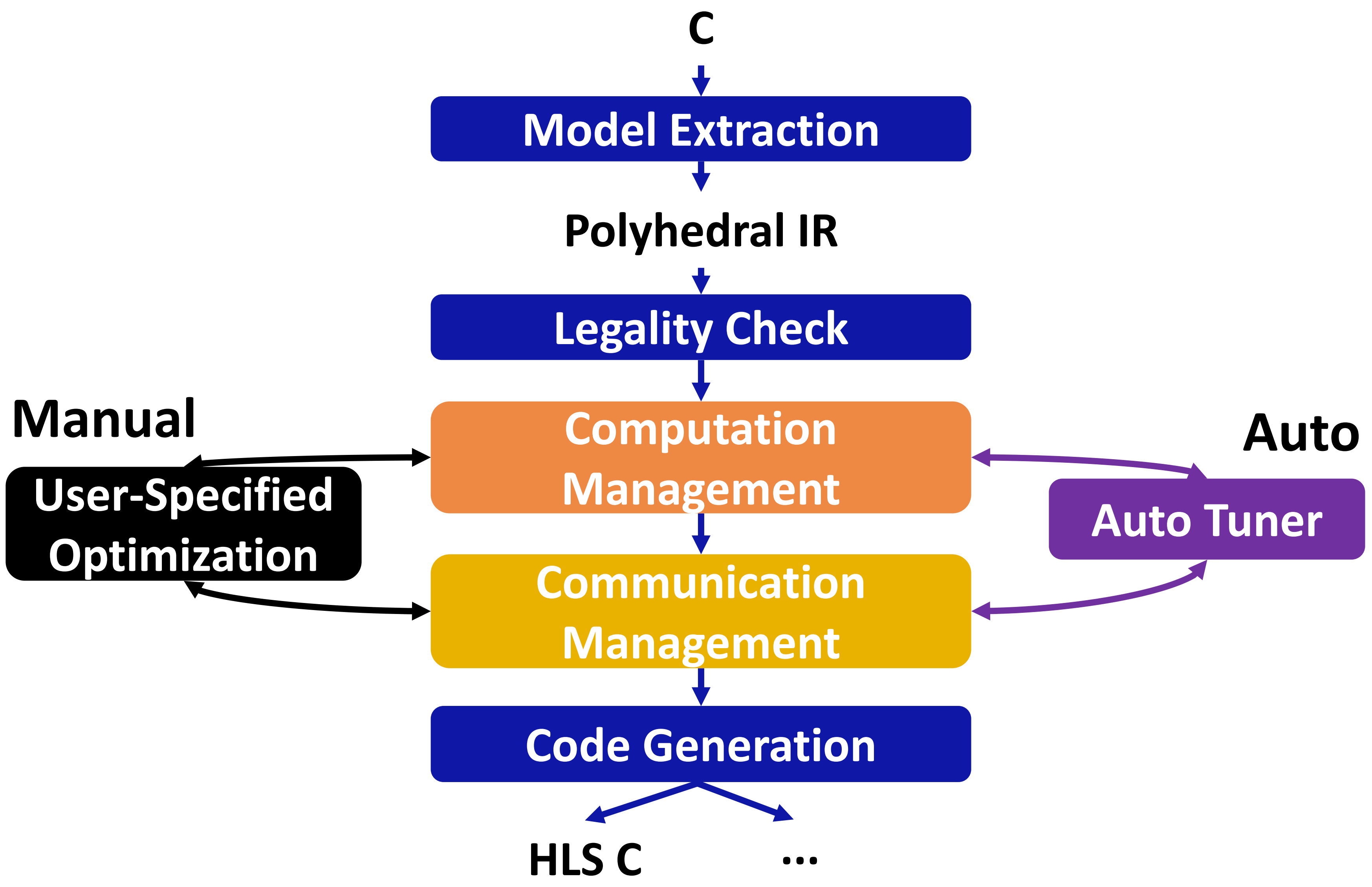




Compilation Flow



Compilation Flow



Outline

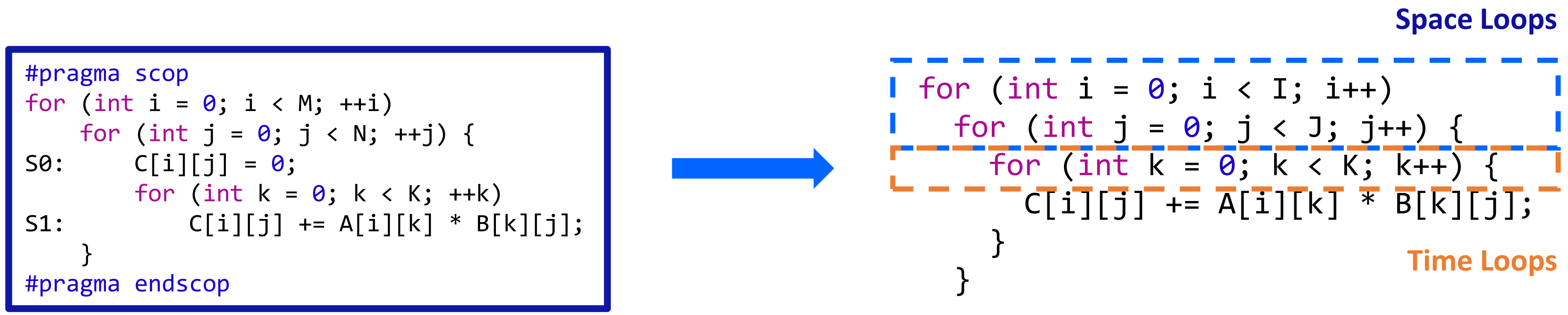
- A Brief Background
- Details about AutoSA Compilation Flow
- **Demos**
  - Matrix Multiplication
  - Convolutional Neural Network
- Auto-Tuning in AutoSA
- Using AutoBridge to Boost the Design Frequency

Demo 1: Matrix Multiplication

- Document: https://autosa.readthedocs.io/en/latest/examples/mm.html
- Example directory: ${AUTOSA_ROOT}/autosa_tests/mm

A Closer Look at Computation Management

- **Space-time mapping**: transforming the program to a systolic array with space-time mapping
- **Array partitioning**: partitioning the array into smaller sub-arrays to fit limited on-chip resource
- **Latency hiding**: permuting parallel loops inside to hide computation latency
- **SIMD vectorization**: vectorizing computation to amortize the PE control overheads



```
#pragma scop
for (int i = 0; i < M; ++i)
    for (int j = 0; j < N; ++j) {
S0:     C[i][j] = 0;
        for (int k = 0; k < K; ++k)
S1:         C[i][j] += A[i][k] * B[k][j];
    }
#pragma endscop
```

```
for (int i = 0; i < I; i++)
  for (int j = 0; j < J; j++) {
    for (int k = 0; k < K; k++) {
      C[i][j] += A[i][k] * B[k][j];
    }
  }
```

A Closer Look at Computation Management

- **Space-time mapping**: transforming the program to a systolic array with space-time mapping
- **Array partitioning**: partitioning the array into smaller sub-arrays to fit limited on-chip resource
- **Latency hiding**: permuting parallel loops inside to hide computation latency
- **SIMD vectorization**: vectorizing computation to amortize the PE control overheads

Space Loops

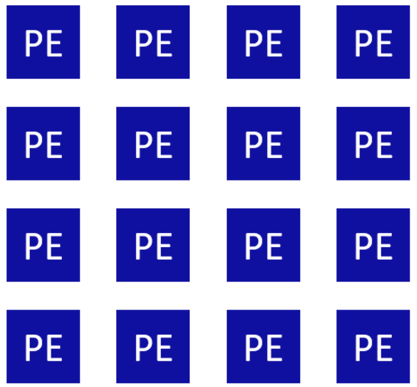
```
for (int i = 0; i < I; i++)
  for (int j = 0; j < J; j++) {
    for (int k = 0; k < K; k++) {
      C[i][j] += A[i][k] * B[k][j];
    }
  }
```

Time Loops

Array Partitioning Loops

```
for (int i = 0; i < M/4; i++)
  for (int j = 0; j < N/4; j++)
    for (int k = 0; k < K/4; k++)
      for (int ii = 0; ii < 4; ii++)
        for (int jj = 0; jj < 4; jj++)
          for (int kk = 0; kk < 4; kk++)
            S1;
```

Sub-Array Loops



A Closer Look at Computation Management

- **Space-time mapping**: transforming the program to a systolic array with space-time mapping
- **Array partitioning**: partitioning the array into smaller sub-arrays to fit limited on-chip resource
- **Latency hiding**: permuting parallel loops inside to hide computation latency
- **SIMD vectorization**: vectorizing computation to amortize the PE control overheads

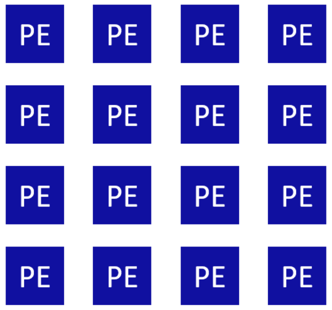
Array Partitioning Loops

```
for (int i = 0; i < M/4; i++)
  for (int j = 0; j < N/4; j++)
    for (int k = 0; k < K/4; k++)
      for (int ii = 0; ii < 4; ii++)
        for (int jj = 0; jj < 4; jj++)
          for (int kk = 0; kk < 4; kk++)
            S1;
```

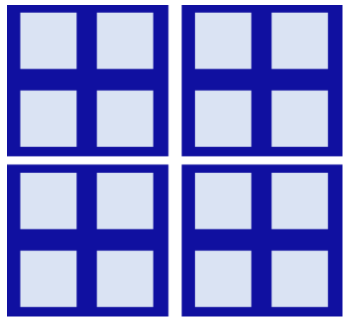
Sub-Array Loops

```
for (int i = 0; i < M/4; i++)
  for (int j = 0; j < N/4; j++)
    for (int k = 0; k < K/4; k++)
      for (int ii = 0; ii < 2; ii++)
        for (int jj = 0; jj < 2; jj++)
          for (int kk = 0; kk < 4; kk++)
            for (iii = 0; iii < 2; iii++)
              #pragma HLS PIPELINE II=1
              for (jjj = 0; jjj < 2; jjj++)
                S1;
```

Latency Hiding Loops



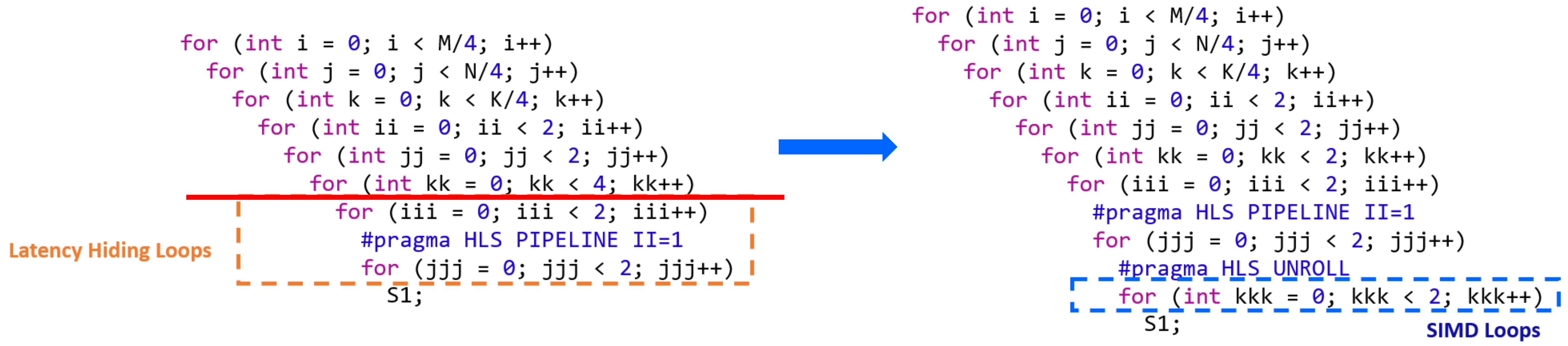
4x4 array



2x2 array

A Closer Look at Computation Management

- **Space-time mapping**: transforming the program to a systolic array with space-time mapping
- **Array partitioning**: partitioning the array into smaller sub-arrays to fit limited on-chip resource
- **Latency hiding**: permuting parallel loops inside to hide computation latency
- **SIMD vectorization**: vectorizing computation to amortize the PE control overheads



```
for (int i = 0; i < M/4; i++)
  for (int j = 0; j < N/4; j++)
    for (int k = 0; k < K/4; k++)
      for (int ii = 0; ii < 2; ii++)
        for (int jj = 0; jj < 2; jj++)
          for (int kk = 0; kk < 4; kk++)
            for (iii = 0; iii < 2; iii++)
              #pragma HLS PIPELINE II=1
              for (jjj = 0; jjj < 2; jjj++)
                S1;
```

```
for (int i = 0; i < M/4; i++)
  for (int j = 0; j < N/4; j++)
    for (int k = 0; k < K/4; k++)
      for (int ii = 0; ii < 2; ii++)
        for (int jj = 0; jj < 2; jj++)
          for (int kk = 0; kk < 2; kk++)
            for (iii = 0; iii < 2; iii++)
              #pragma HLS PIPELINE II=1
              for (jjj = 0; jjj < 2; jjj++)
                #pragma HLS UNROLL
                for (int kkk = 0; kkk < 2; kkk++)
                  S1;
```

Demo 2: CNN

- Document: https://autosa.readthedocs.io/en/latest/examples/cnn.html
- Example directory: ${AUTOSA_ROOT}/autosa_tests/cnn

Auto-Tuning

Generality

Program with restrictions

- Rectangular iteration domain

Arbitrary program

Genetic Search

Exhaustive Search with Pruning

< 1 minute

Minutes to hours

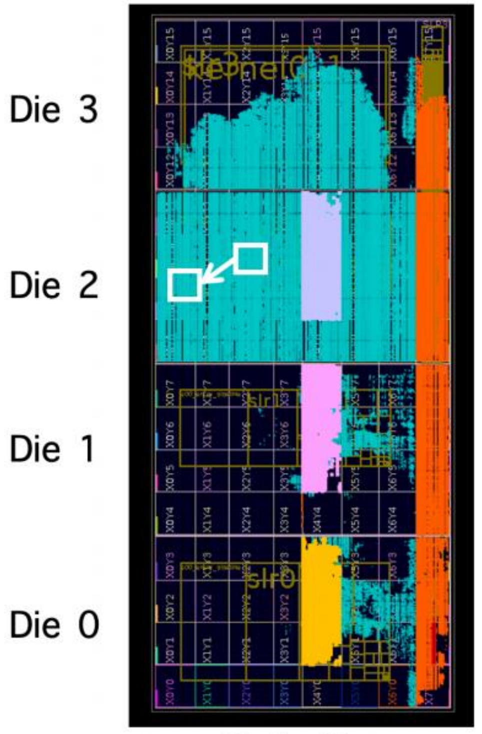
Convergence Time

Auto-Tuning

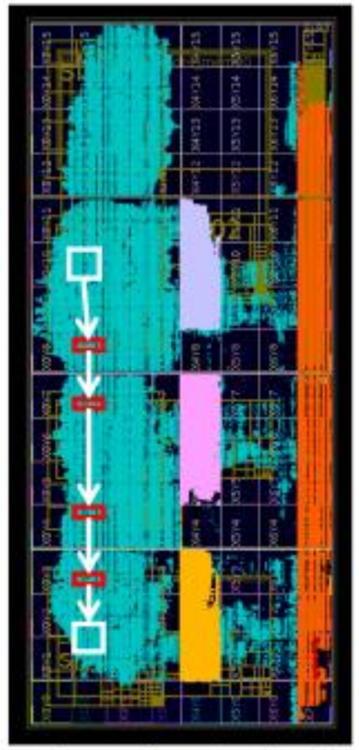
- Exhaustive search
  - https://autosa.readthedocs.io/en/latest/tutorials/auto_tuning_exhaustive.html
- Genetic search
  - https://autosa.readthedocs.io/en/latest/tutorials/auto_tuning_genetic.html
- Demo:
  - Use genetic search for matrix multiplication on Alveo U250

Getting High-Frequency Systolic Array Designs with Floorplanning

Default place/router make suboptimal choices.
Design Freq.:
216MHz



Default



Floorplan-Guided

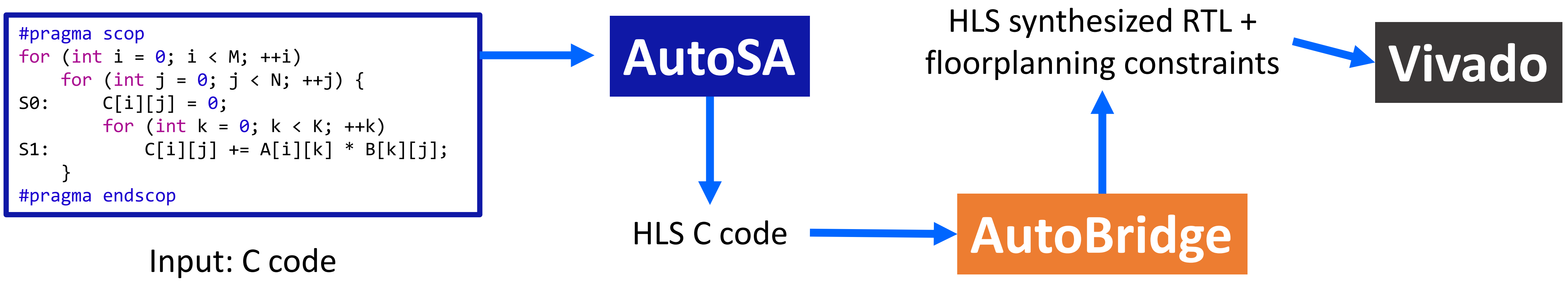
Floorplanning-guided approach spreads the logic across chip to reduce local congestion.
Design Freq.:
329MHz

Example: Improve the CNN systolic array design frequency using AutoBridge

Automated Interconnect Pipelining for Multi-Die FPGA Designs

Guo, Licheng, et al. "**AutoBridge**: Coupling Coarse-Grained Floorplanning and Pipelining for High-Frequency HLS Design on Multi-Die FPGAs." (**FPGA 2021 Best Paper**)

- AutoBridge is an HLS-based automatic floorplanning flow.
- Improved the average frequency of 43 designs from 147MHz to 297MHz with a negligible area overhead.
- Github: https://github.com/Licheng-Guo/AutoBridge



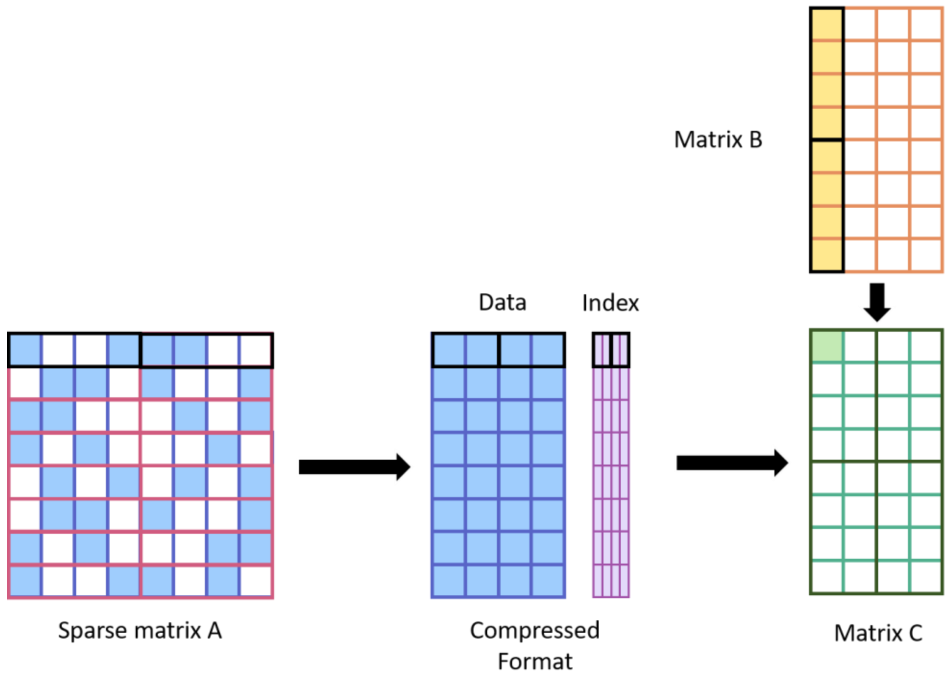
```
#pragma scop
for (int i = 0; i < M; ++i)
    for (int j = 0; j < N; ++j) {
S0:     C[i][j] = 0;
        for (int k = 0; k < K; ++k)
S1:         C[i][j] += A[i][k] * B[k][j];
    }
#pragma endscop
```

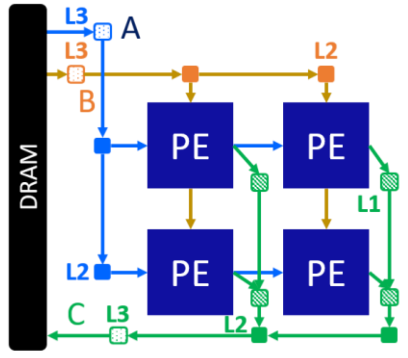
More Details

- An example of using AutoBridge with AutoSA
- https://autosa.readthedocs.io/en/latest/tutorials/auto_bridge.html

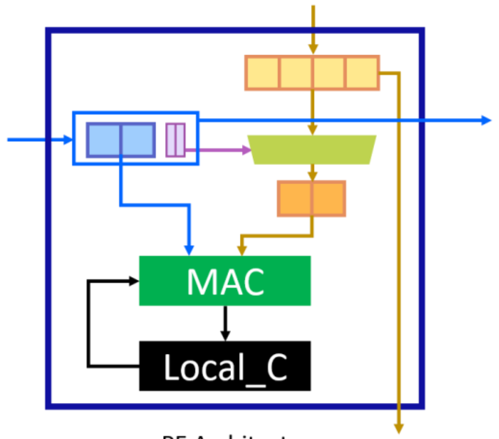
Other Features

- Support for structural sparsity
  - https://autosa.readthedocs.io/en/latest/tutorials/structural_sparsity.html





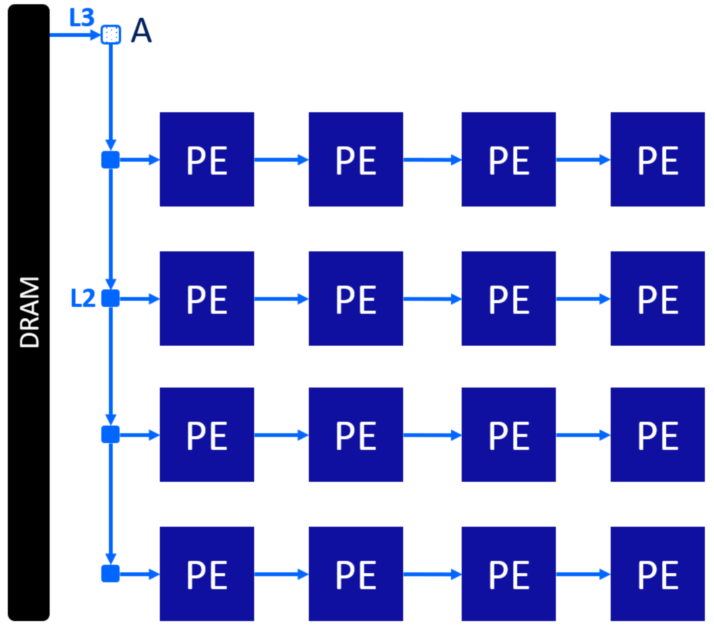
Array Architecture



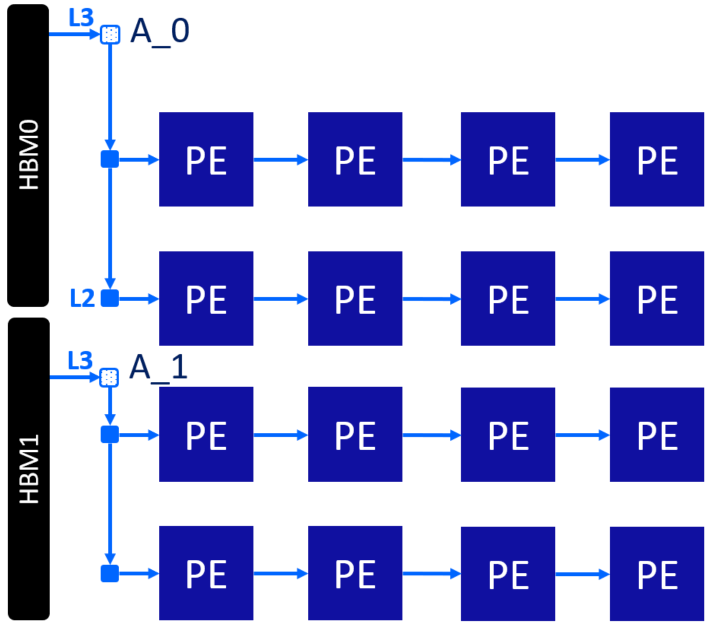
PE Architecture

Other Features

- Support for HBM
  - https://autosa.readthedocs.io/en/latest/examples/mm_hbm.html



Array Using Single DDR/HBM Bank



Array Using Multiple DDR/HBM Banks

How Can This Tool Benefit the Community?

- Understand the mechanisms of systolic array architecture, HLS design methodology, and polyhedral compilation
  - UCLA CS259
- Integrate AutoSA-generated designs into the accelerator framework
  - FlexCNN, HeteroCL
- Architecture studies based on an open-sourced and performant baseline
  - Architecture trade-offs for applications (dataflows, array configs, …)
  - Programmable systolic array
  - Comparison against CGRA
  - …

Conclusion

- AutoSA, an end-to-end systolic array compiler for FPGA
- Goal: maximize generality, performance, and productivity when generating systolic arrays.
- Open-source and under active development. Welcome for any feedbacks and contribution!
- Any other questions?
- jiewang@cs.ucla.edu

User-Specified Optimizations

Option	Explanation	Example
array-contraction	Apply array contraction to reduce the memory usage.	True/False
axi-stream	Generate AXI stream interface.	True/False
block-sparse	Use block sparsity.	True/False
block-sparse-ratio	Block sparsity ratio.	[kernel[]->A[2,4]] Matrix A is sparse, every 4 elements are grouped into a vector, with 2 non-zero elements.
data-pack	Enable data-packing.	True/False
data-pack-sizes	Specifies the maximal data-packing factors at each I/O level.	[kernel[]->data_pack[8,32,64]]
double-buffer	Enable double buffering.	True/False
double-buffer-assign	Assign specific arrays to be double-buffered.	[kernel[]->A[]]

User-Specified Optimizations

Option	Explanation	Example
explore-loop-permute	Explore loop permutation in the step of array partitioning.	True/False
fifo-depth	FIFO depth.	2
hbm	Use multi-port DRAM/HBM	True/False
hbm-port-num	Specify HBM port number per array.	2
hls	Generate Xilinx HLS host.	True/False
host-serialize	Serialize/deserialize the host data.	True/False
io-module-embedding	Embed I/O modules inside PEs if possible.	True/False
loop-infinitize	Apply loop infinitization optimization (Intel OpenCL only)	True/False

User-Specified Optimizations

Option	Explanation	Example
local-reduce	Generate non-output-stationary array with local reduction.	True/False
reduce-op	Reduction operator.	“+”
max-sa-dim	Maximal systolic array dimension.	2
sa-sizes	Computation management parameters.	[kenrel[]->array_part[4,4]]