

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

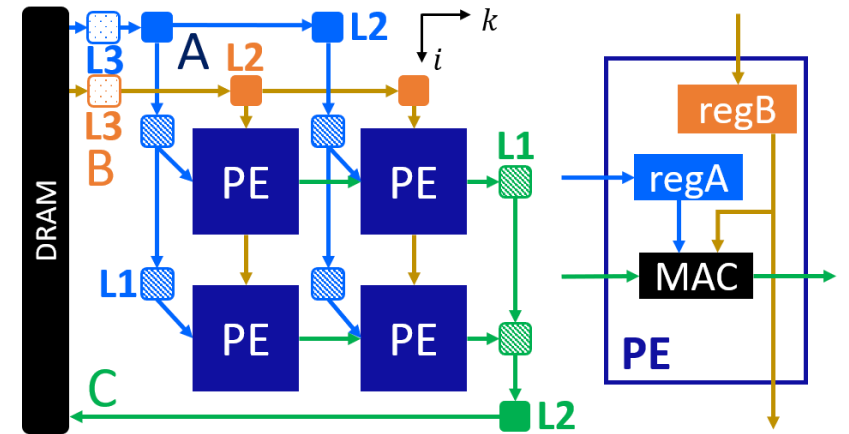
Wang, Jie, Licheng Guo, and Jason Cong. "AutoSA: A Polyhedral Compiler for High-Performance Systolic Arrays on FPGA." Proceedings of the 2021 ACM/SIGDA international symposium on Field-programmable gate arrays. 2021.

```
#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 endsco
```

Input: C code



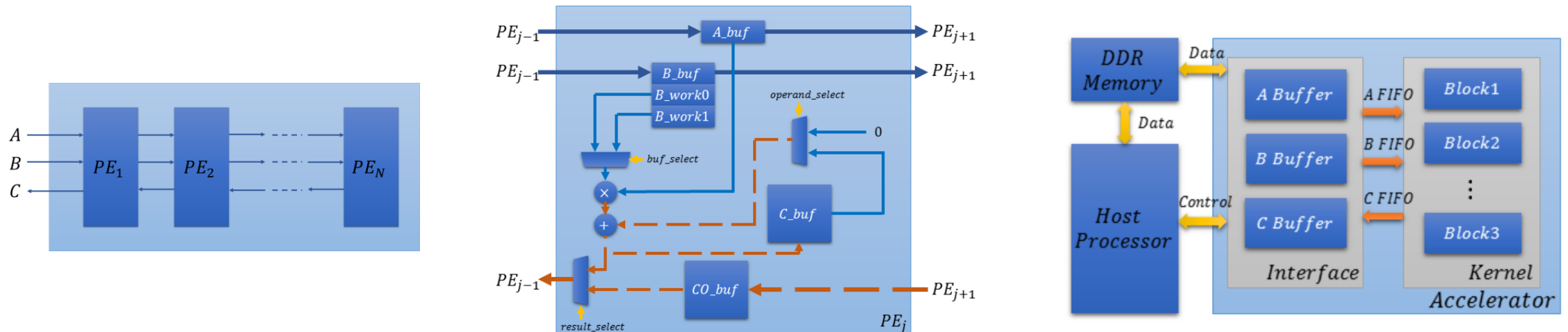
AutoSA



Output: Systolic array design in HLS C

How Did We Get Started?

- Manual systolic array design for matrix multiplication in Summer'2014
- Time: ~5 months
- A manual-designed RTL-based 1D systolic array with over 1,700 line of codes
- 198.1 GFLOPs on Xilinx VC709



Related Work on Automated SA Compilation

MMAAlpha '01

PolySA '18

AutoSA '21

Bondhugula et al. '07

SuSy '20

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.

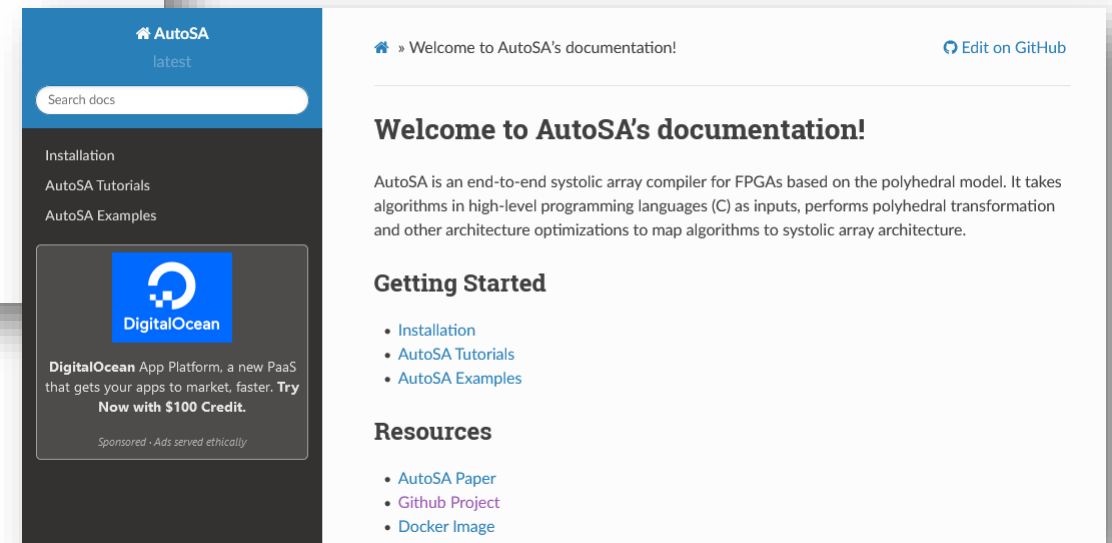
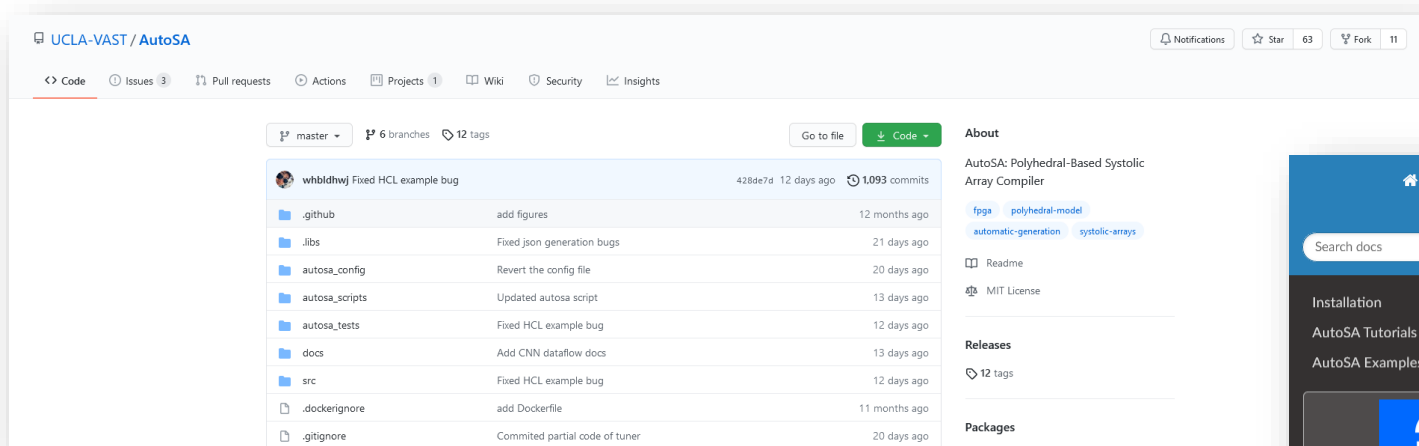
Auto-Tuning	No	No	Yes	No	Yes
-------------	----	----	-----	----	-----

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://autosar.readthedocs.io/en/latest/>



Current Capabilities

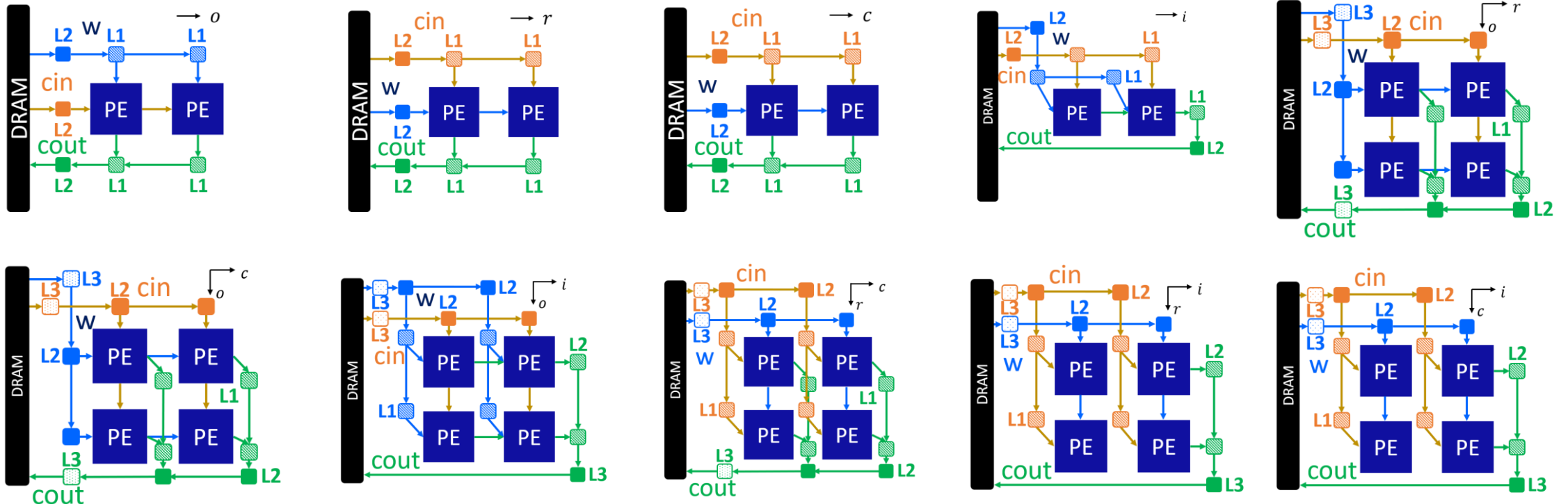
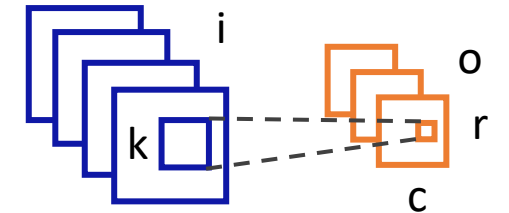
- Back-end
 - Xilinx HLS C (Mature, recommended)
 - Intel OpenCL (Experimental)
 - Catapult HLS C (Experimental)
- High-Performance designs on Xilinx platforms

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

Board: Xilinx Alveo U250

Current Capabilities

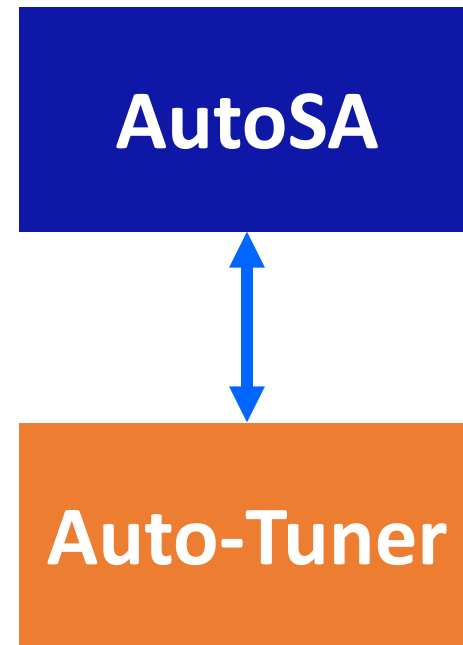
- **Architectural exploration** of different dataflows
- Example: 10 dataflows of CNN
(<https://autosa.readthedocs.io/en/latest/examples/cnn.html>)



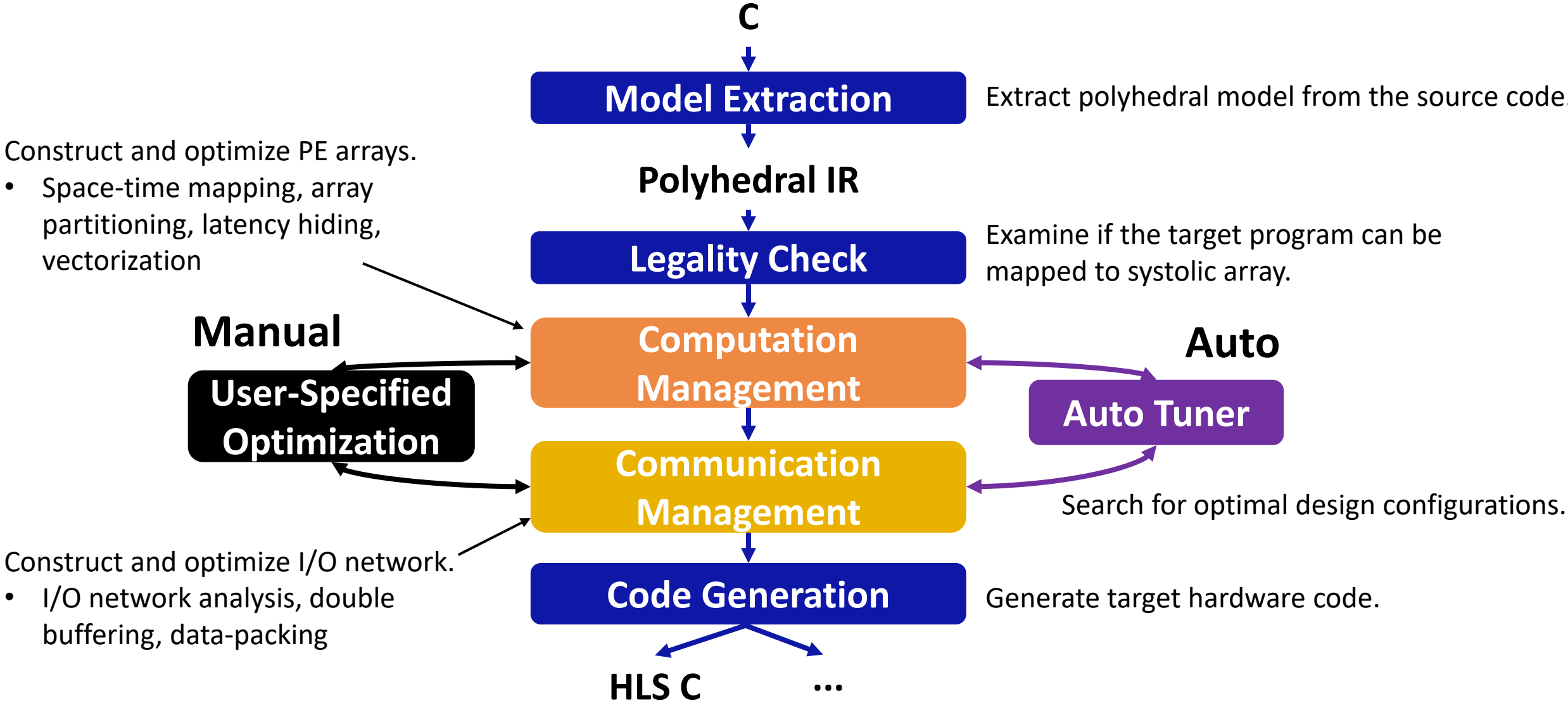
Current Capabilities

- Auto-tuning support

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



Compilation Flow



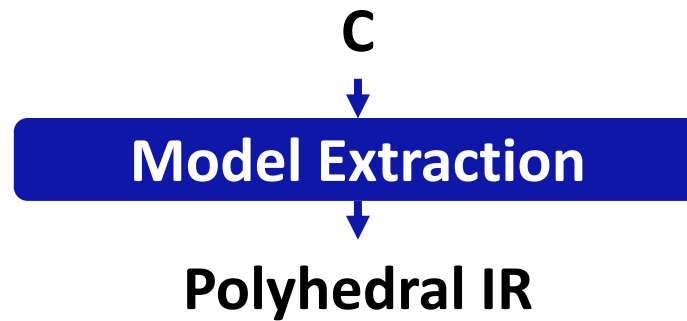
Compilation Flow

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

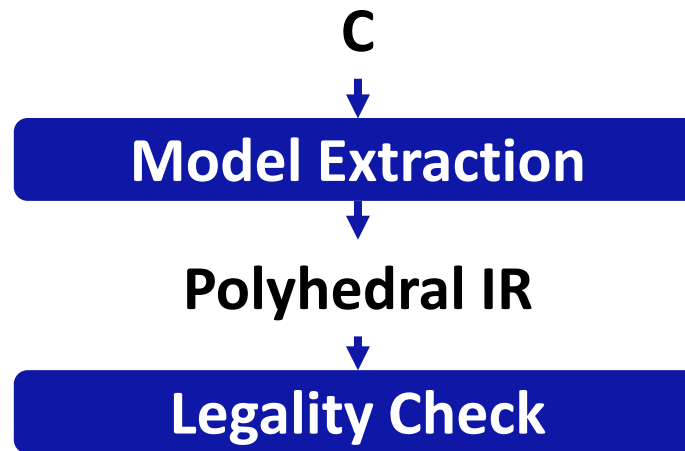
Compilation Flow



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.
 - Integer Set Library

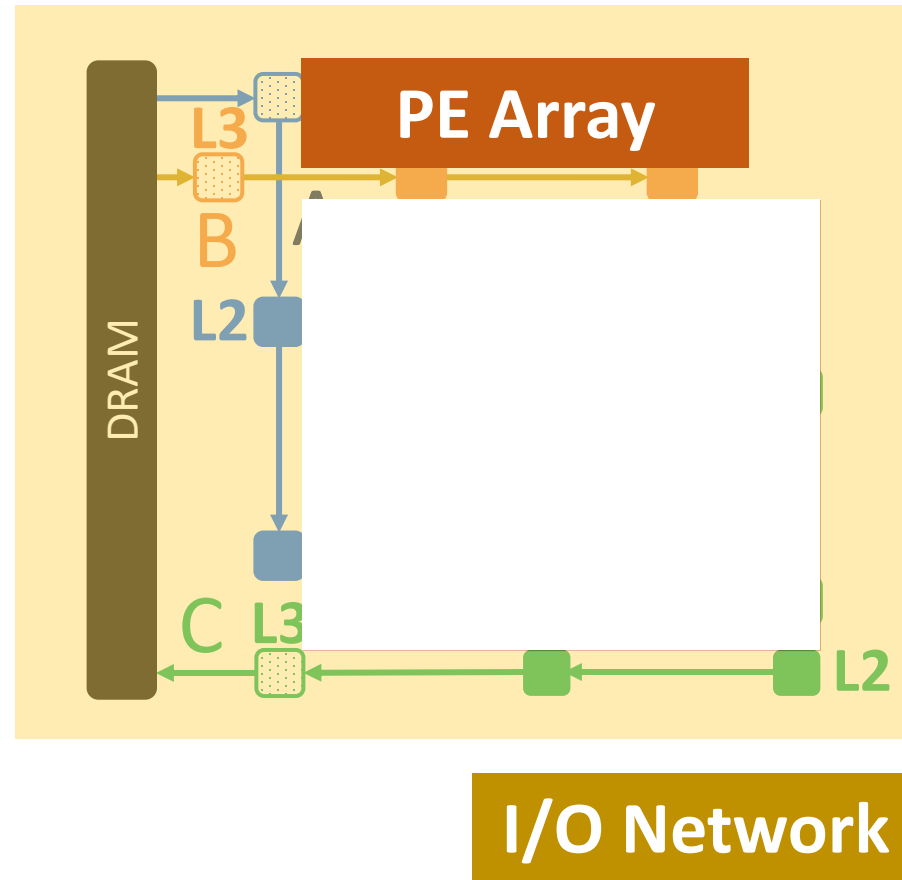
Compilation Flow



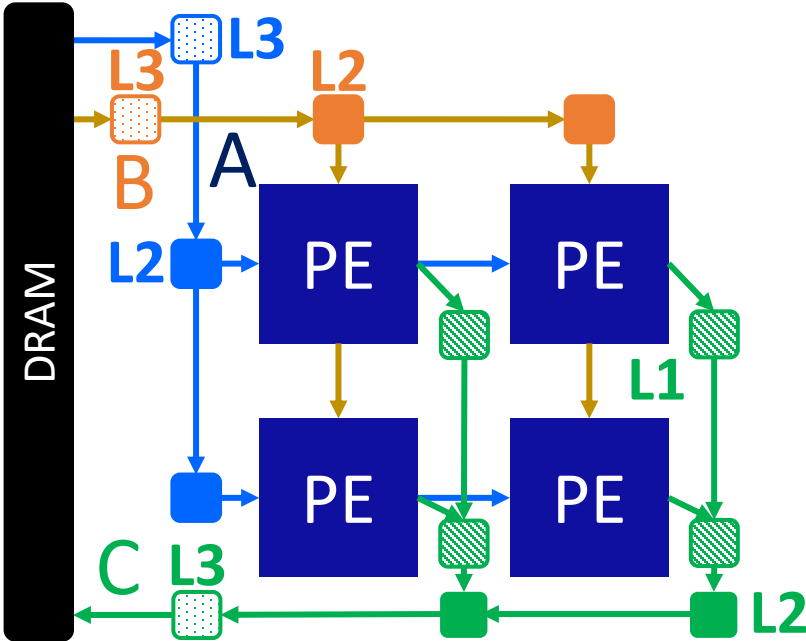
Legality Check

- All dependences are **uniform** dependences (with constant dependence distance)
➡ **Statically-scheduled** design
- Dependence distances on space loops (loops to be mapped to PE dimensions) to be **no greater than one.**
➡ **Local** communication

Compilation Flow



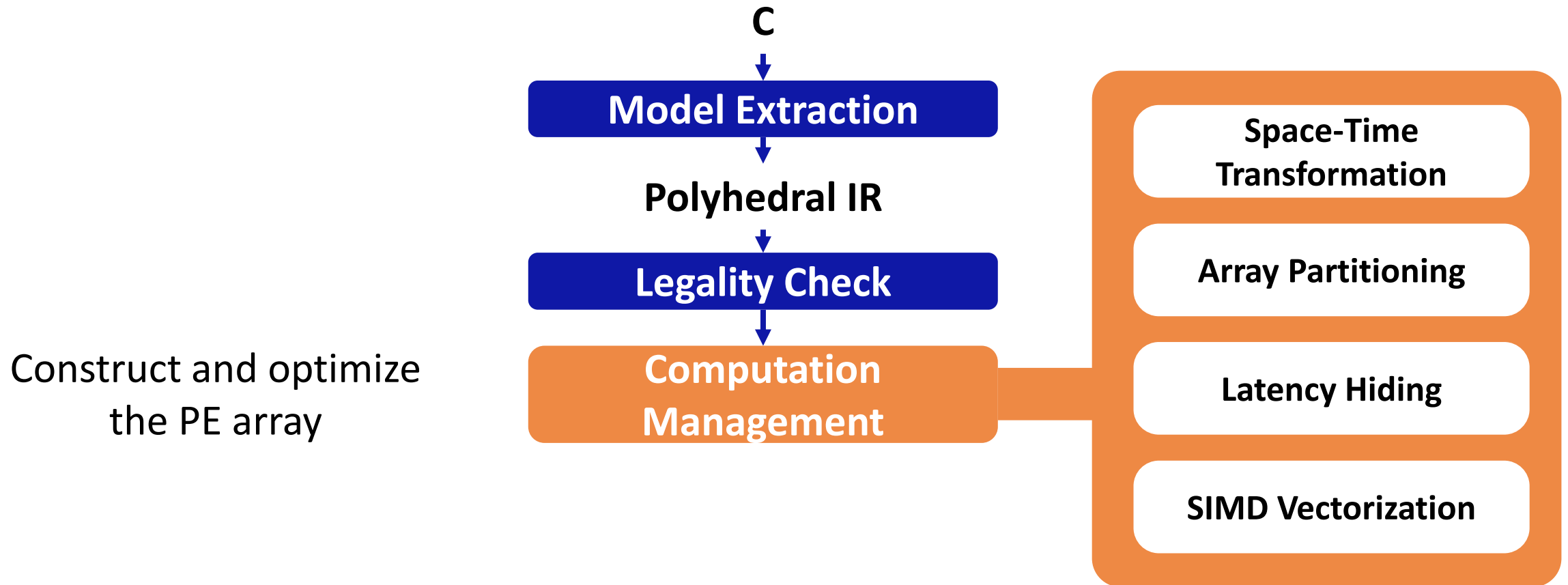
Compilation Flow



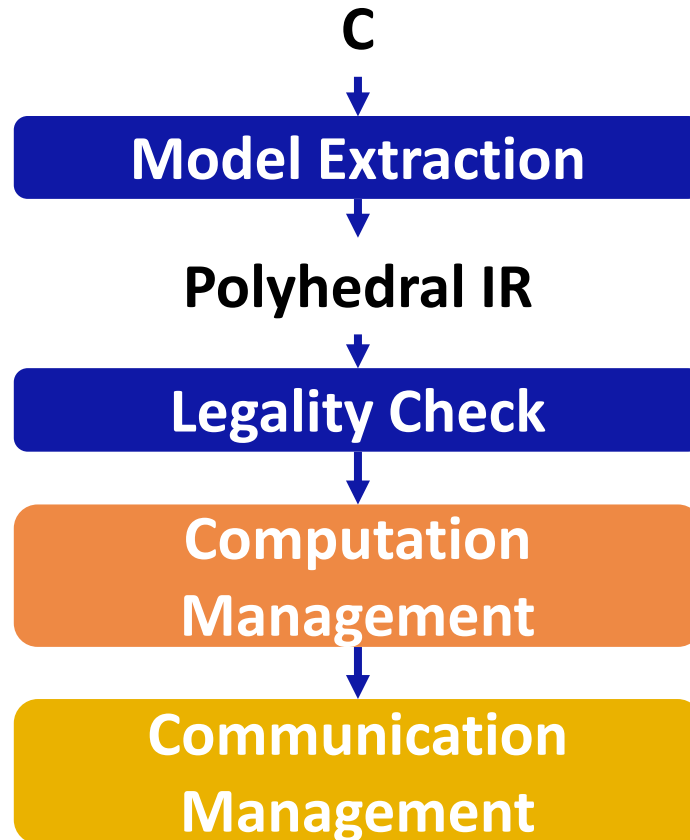
Computation Management

Communication Management

Compilation Flow



Compilation Flow



Construct and optimize
the I/O network

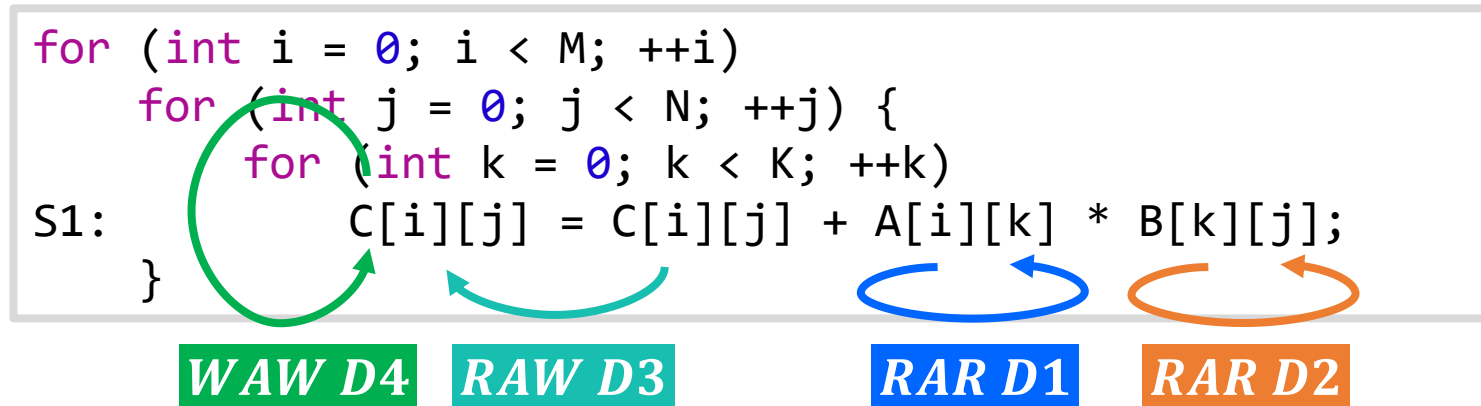
Communication Management

How to generate the I/O network?

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

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

Communication Management



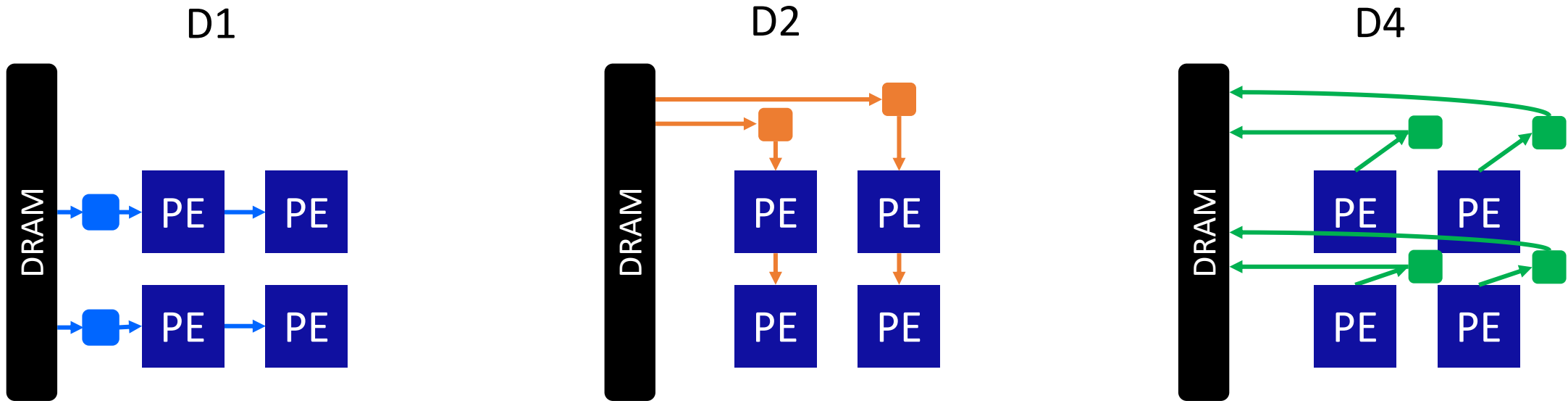
Dependence

Dependence Type

Array Access

**Dependence
Distance**

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



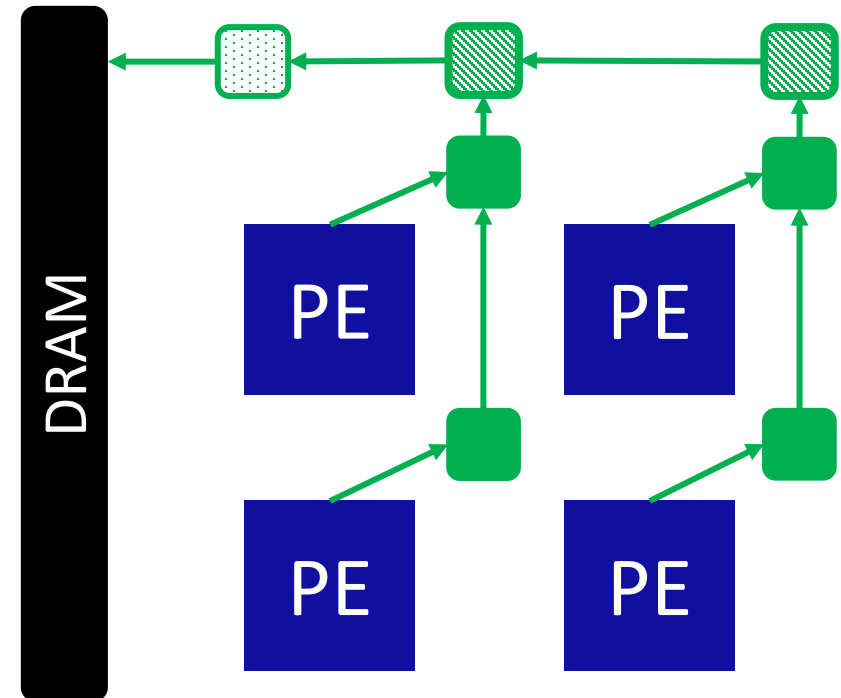
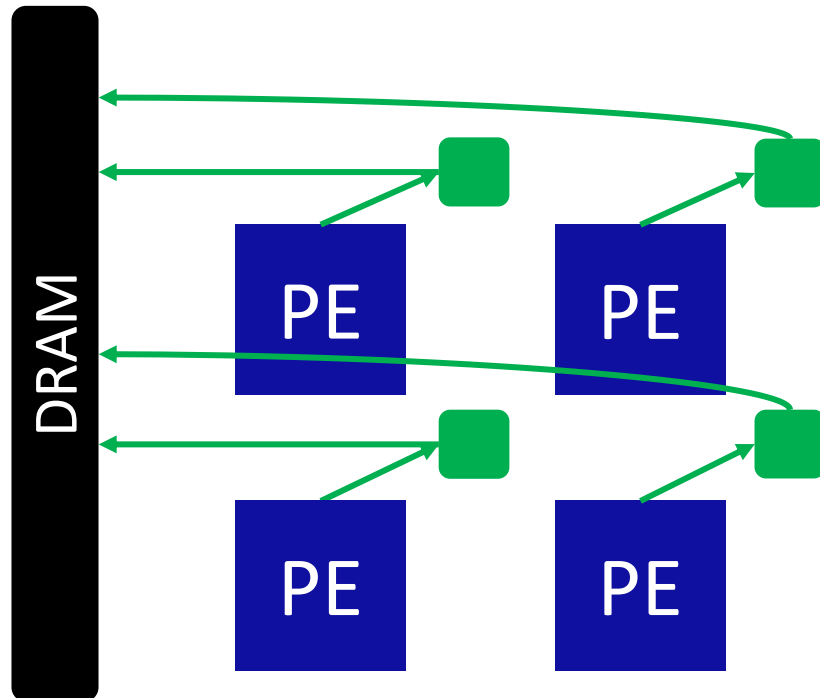
Dependence	Dependence Type	Array Access	Dependence Distance
D1	Read (RAR)	$A[i][k]$	$(0, 1, 0)$
D2	Read (RAR)	$B[k][j]$	$(1, 0, 0)$
D3	Flow (RAW)	$C[i][j]$	$(0, 0, 1)$
D4	Output (WAW)	$C[i][j]$	$(0, 0, 1)$

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

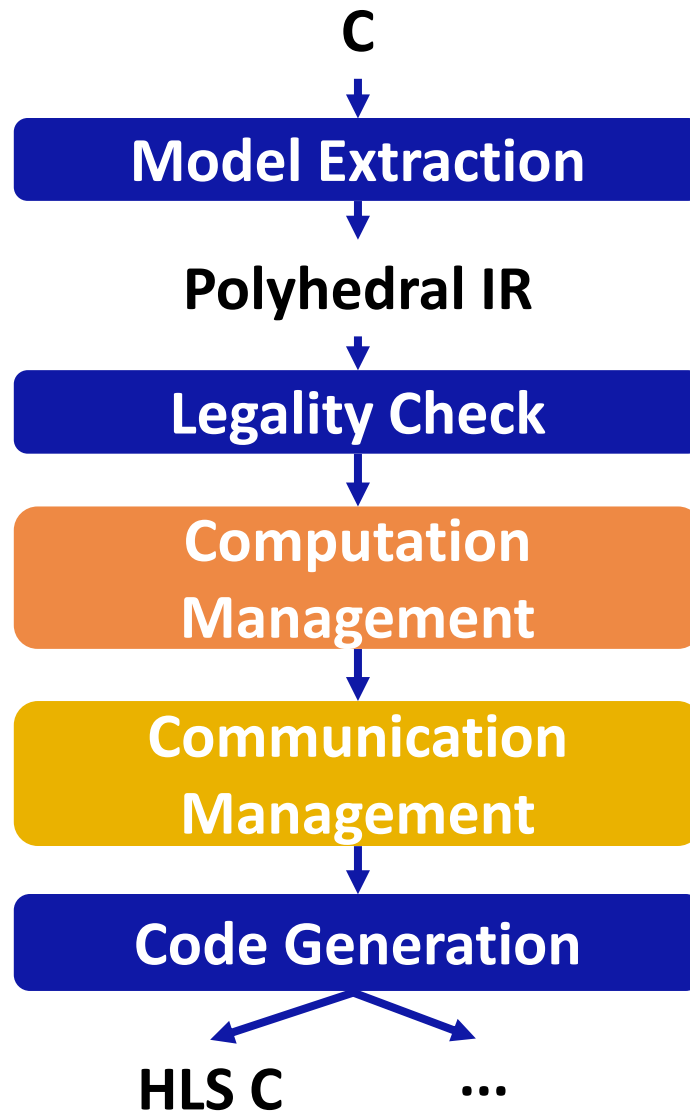
Communication Management

How to optimize the I/O network?

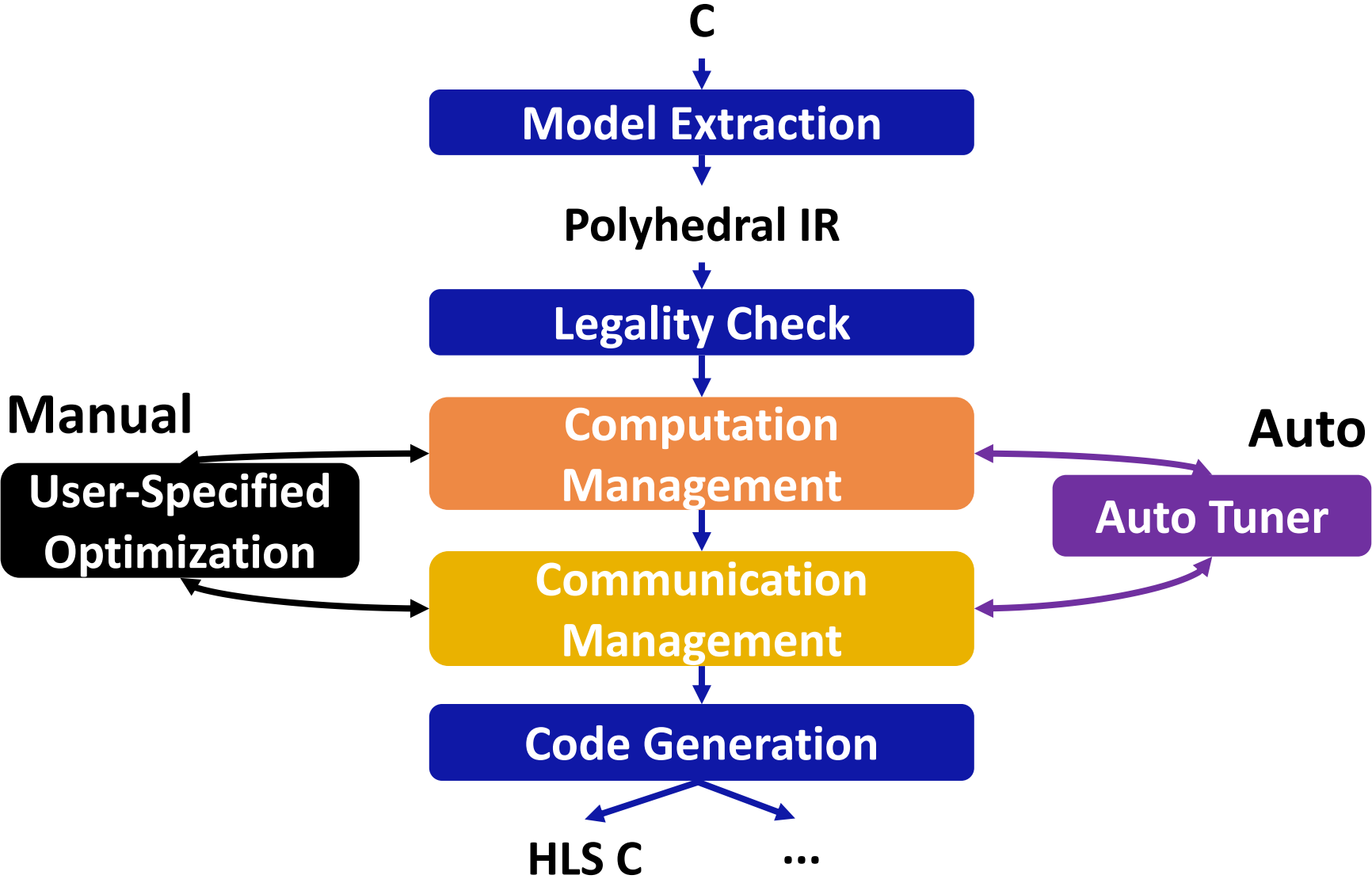
- I/O network localization
- Double buffering, data packing, etc.



Compilation Flow



Compilation Flow

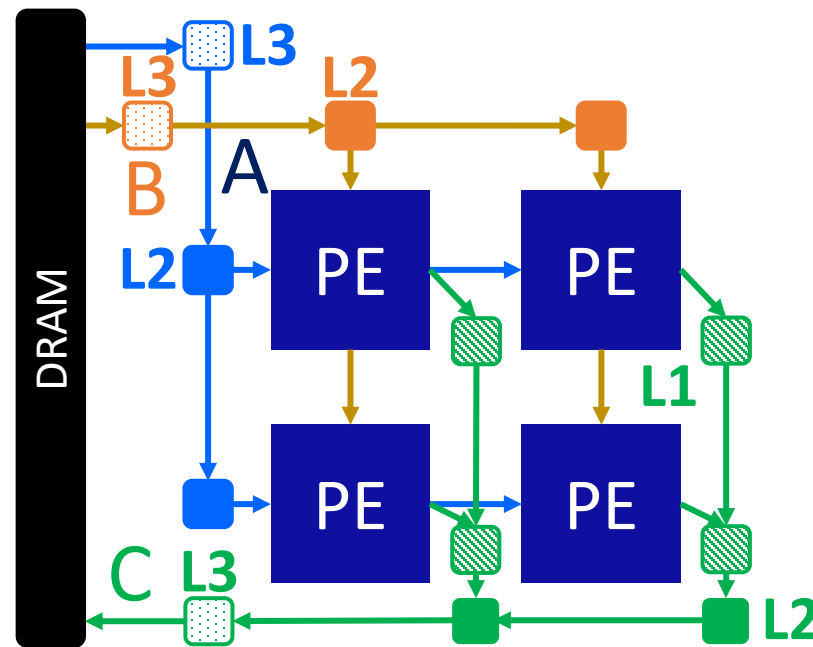


Outline

- 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

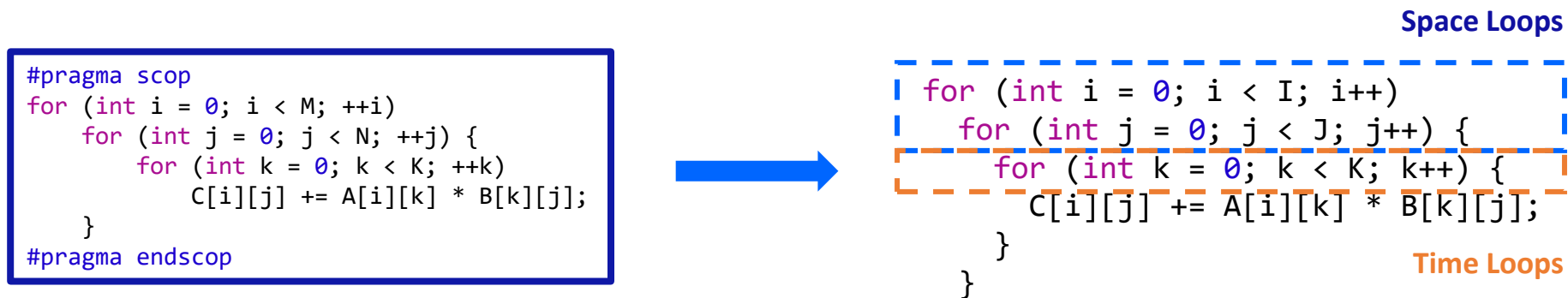
Demo 1: Matrix Multiplication

- 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



--sa-sizes="{Kernel[]->space_time[3]}"

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 < 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];
    }
  }
```

Space Loops

Time Loops

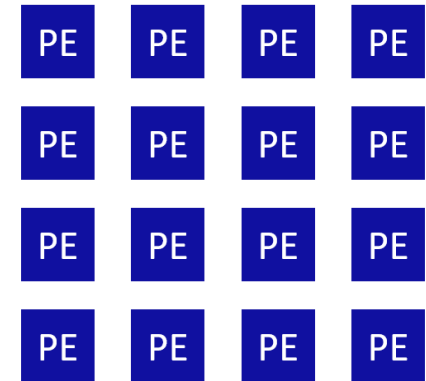


```
for (int i = 0; i < M/4; i++)
  for (int j = 0; j < N/4; j++)
    for (int k = 0; k < K/4; k++)
      for (int ii = 0; ii < 4; ii++)
        for (int jj = 0; jj < 4; jj++)
          for (int kk = 0; kk < 4; kk++)
            S1;
```

Array Partitioning Loops

Sub-Array Loops

--sa-sizes="{Kernel[]->array_part[4,4,4]}"



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

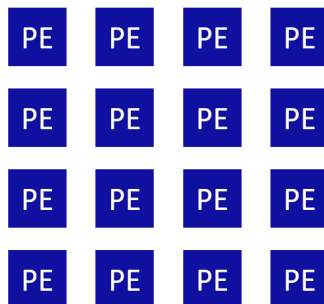
Array Partitioning Loops

```

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

```
--sa-sizes="{Kernel[]->latency[2,2]}"
```



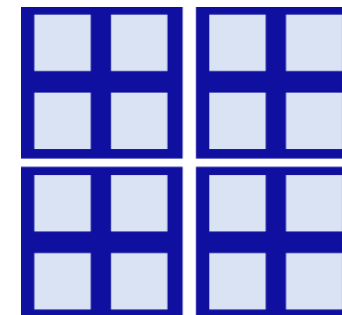
4x4 array



Latency Hiding 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 (int iii = 0; iii < 2; iii++)
              #pragma HLS PIPELINE II=1
              for (int jjj = 0; jjj < 2; jjj++)
                S1;
  
```



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 (int iii = 0; iii < 2; iii++)
              #pragma HLS PIPELINE II=1
                for (int jjj = 0; jjj < 2; jjj++)
                  S1;

```

Latency Hiding 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 < 2; kk++)
            for (int iii = 0; iii < 2; iii++)
              #pragma HLS PIPELINE II=1
                for (int jjj = 0; jjj < 2; jjj++)
                  #pragma HLS UNROLL
                    for (int kkk = 0; kkk < 2; kkk++)
                      S1;

```

SIMD Loops

```
--sa-sizes="{Kernel[]-> simd[2]}"
```

Demo 2: CNN

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

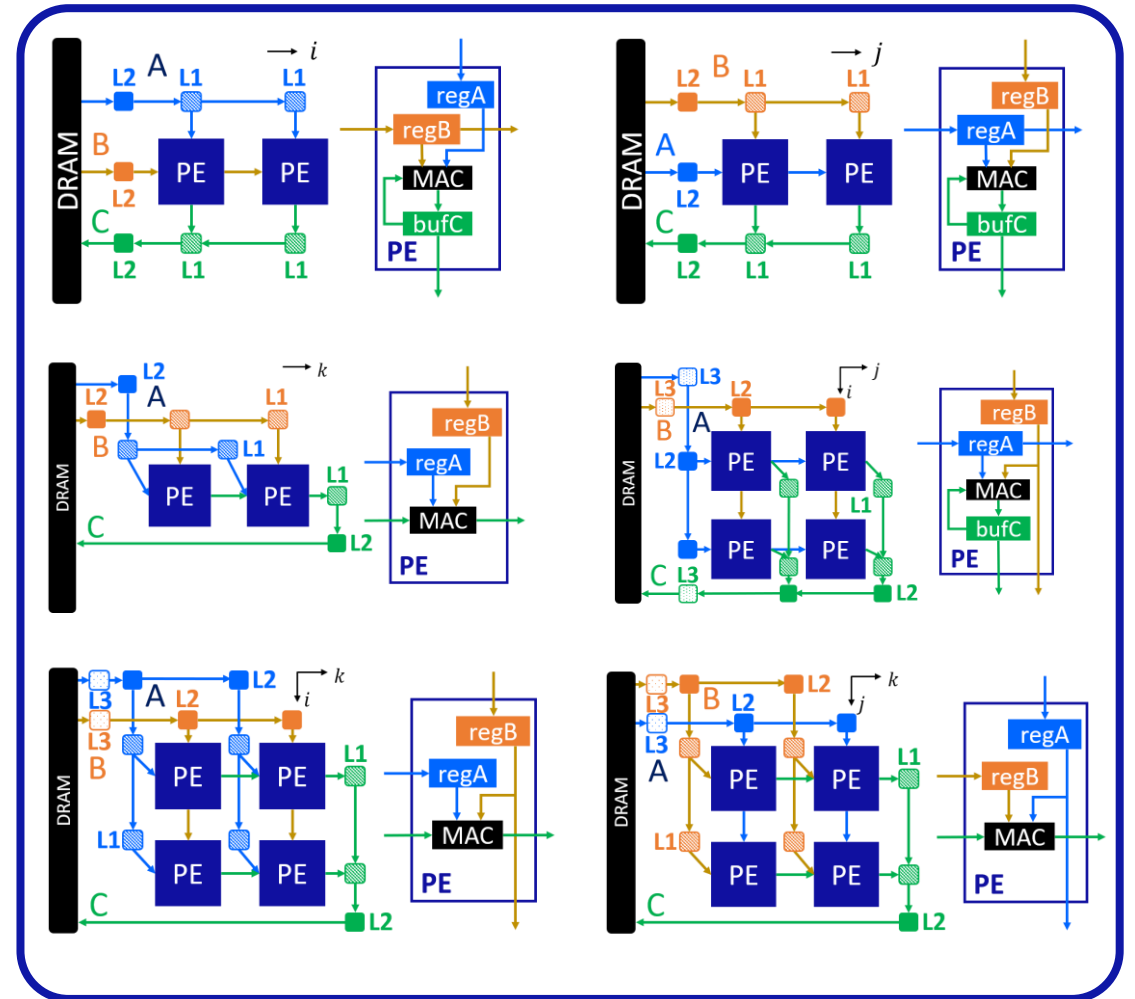
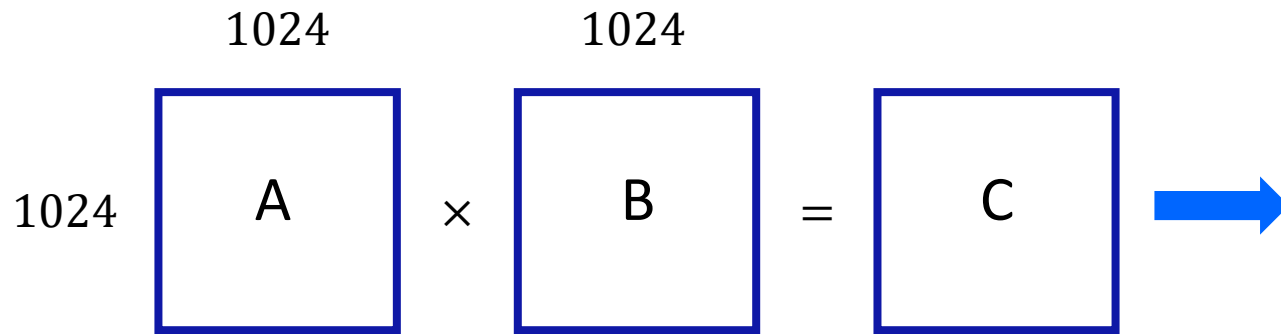
Outline

- 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

Auto-Tuning

A Large Design Space to Explore


Example: Matrix Multiplication



Dataflow types (6) x Dataflow configurations ($O(2^{40})$)

Auto-Tuning

Generality

- 
- Program with restrictions
 - Rectangular iteration domain
 - DSP, BRAM
 - Arbitrary program
 - DSP, BRAM, LUT, FF

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

Auto-Tuning

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

Step 1: Generate Design Descriptor

Contains necessary information for resource and latency estimation.

Step 2: Search

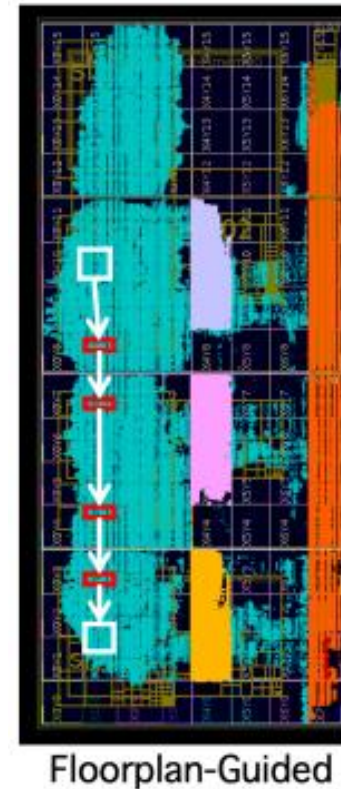
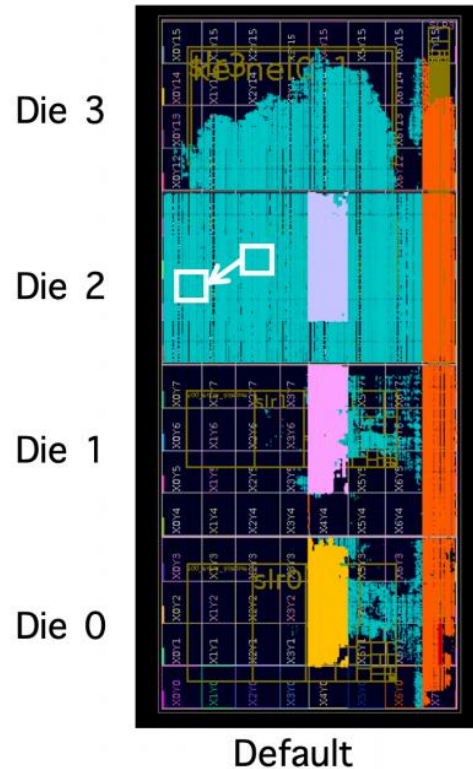
Use genetic search to find near-optimal designs.

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**

Getting High-Frequency Systolic Array Designs with **Floorplanning**

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



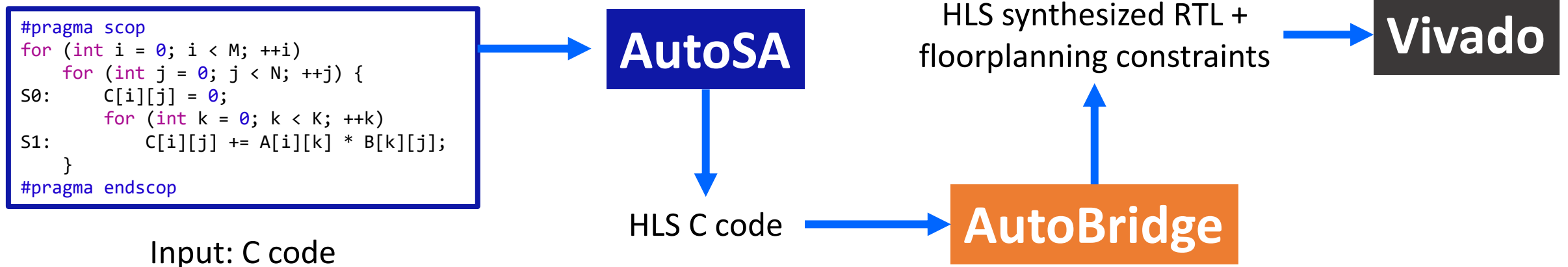
Floorplanning-guided approach spreads the logic across chip to reduce local congestion.
Design Freq.: **329MHz**

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>

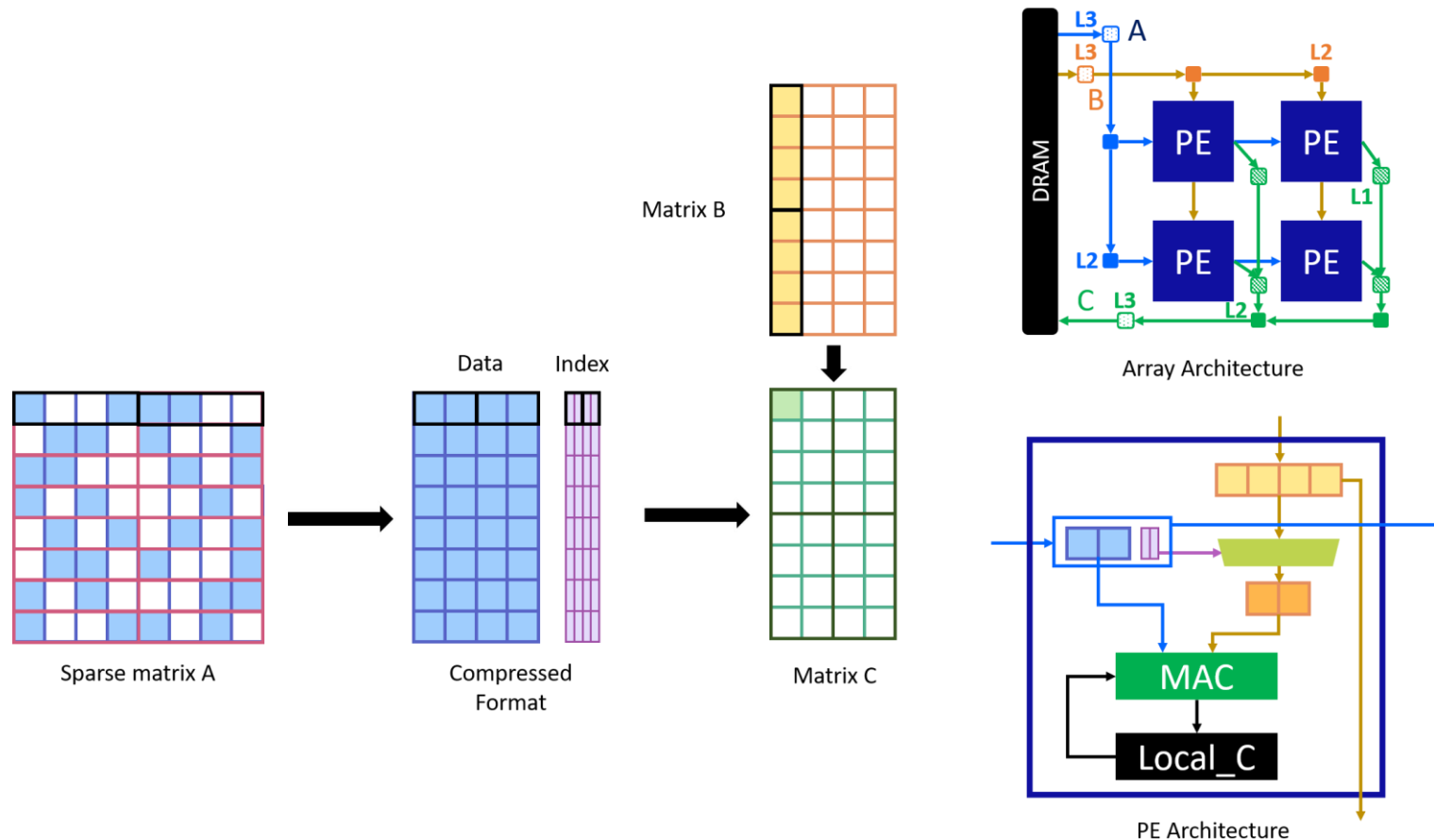


More Details

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

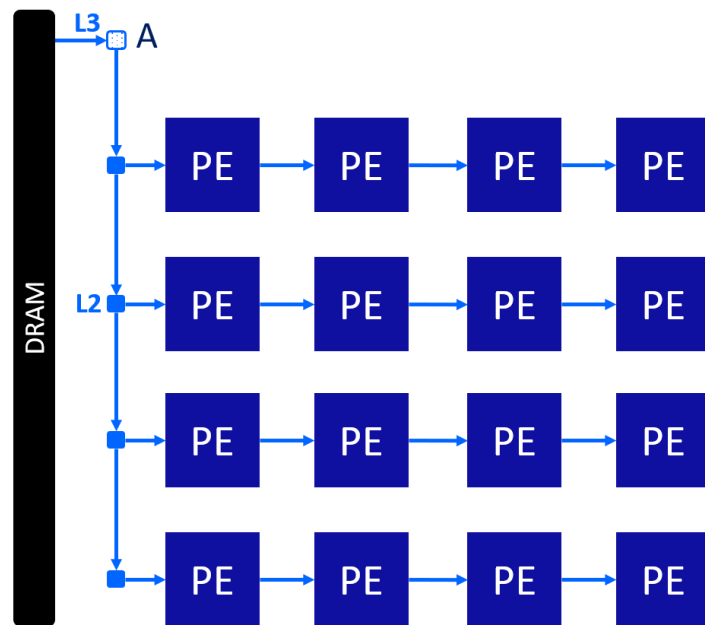
Other Features

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

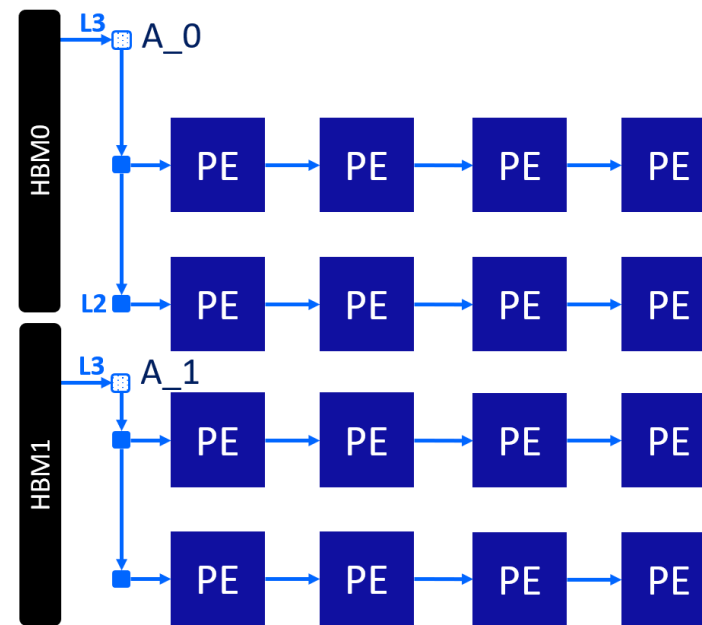


Other Features

- 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?
- jiawang@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-infinite	Apply loop infinite 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.	[kernel[]->array_part[4,4]]