Automating Beyond High-Level Synthesis

Jason Cong

Chancellor's Professor, UCLA

Director, Center for Domain-Specific Computing

cong@cs.ucla.edu

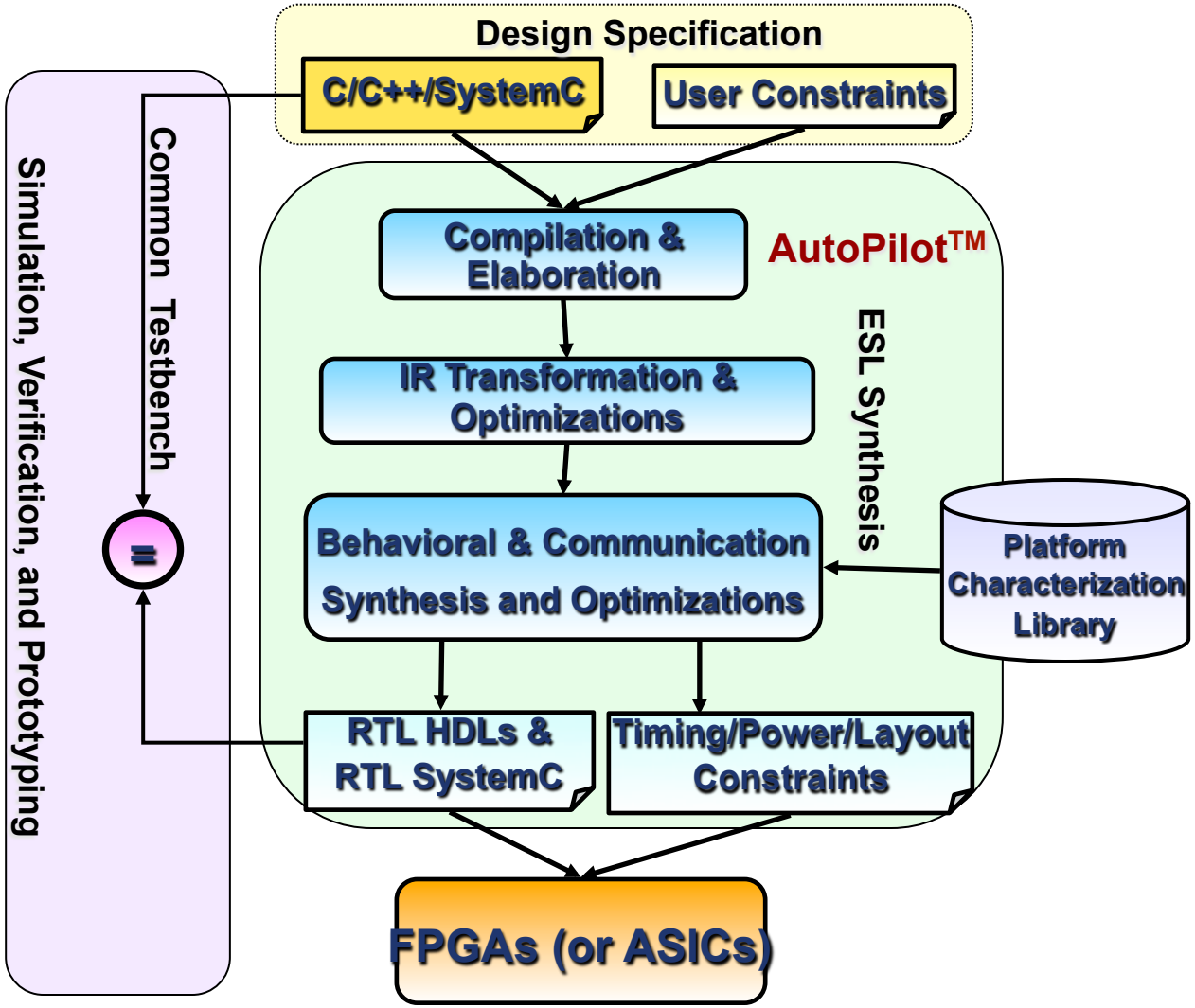
http://cadlab.cs.ucla.edu/~cong

High-Level Synthesis: A Brief History

- **Early attempts**
  - Research projects
  - 1980s ~ early 1990s
- **Rise and fall of early commercialization**
  - Tools from major EDA vendors
  - 1990s~early 2000s
- **Renewed interests**
  - Start-ups, followed by major EDA vendors
  - Mid 2000 ~ present
- **Wide adoption by the FPGA design community**
  - Led by Xilinx Vivado HLS (based on AutoESL acquisition in 2011)
  - 2012 ~ present

C/C++ to FPGA Synthesis

xPilot (UCLA) -> AutoPilot (AutoESL) -> Vivado HLS (Xilinx)



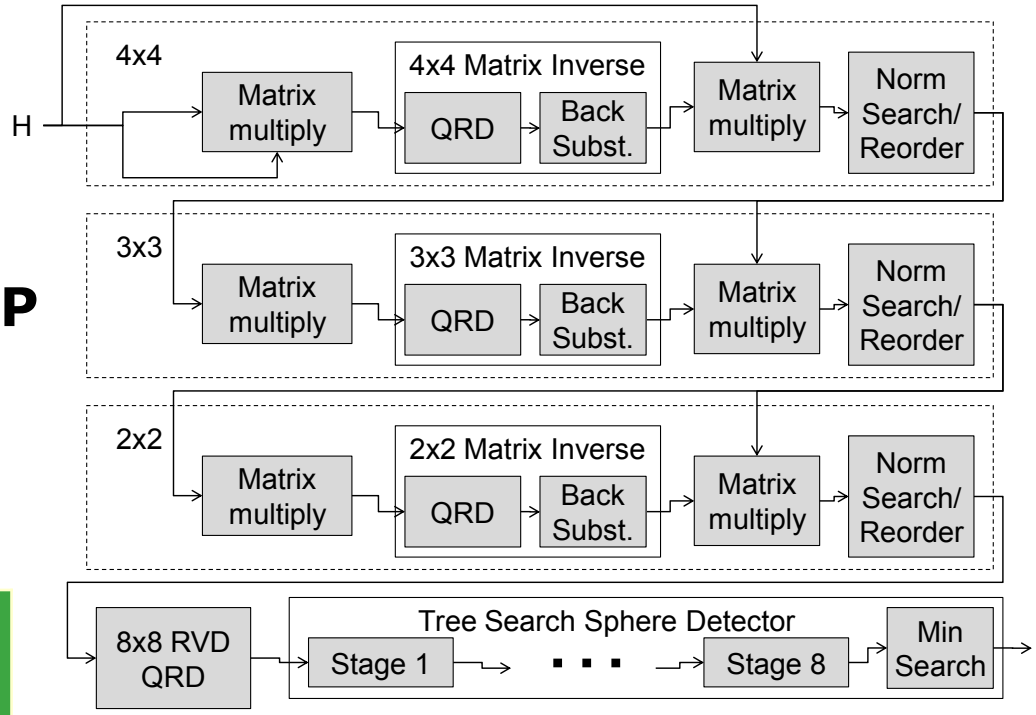
- Platform-based C to RTL synthesis
- Synthesize pure ANSI-C and C++, GCC-compatible compilation flow
- Full support of IEEE-754 floating point data types & operations
- Efficiently handle bit-accurate fixed-point arithmetic
- SDC-based scheduling
- Automatic memory partitioning
- …

QoR matches or exceeds manual RTL for many designs

Developed by AutoESL, acquired by Xilinx in Jan. 2011

AutoPilot Results: Sphere Decoder (from Xilinx)

- **Wireless MIMO Sphere Decoder**
  - **~4000 lines of C code**
  - **Xilinx Virtex-5 at 225MHz**
- **Compared to optimized IP**
  - **11-31% better resource usage**

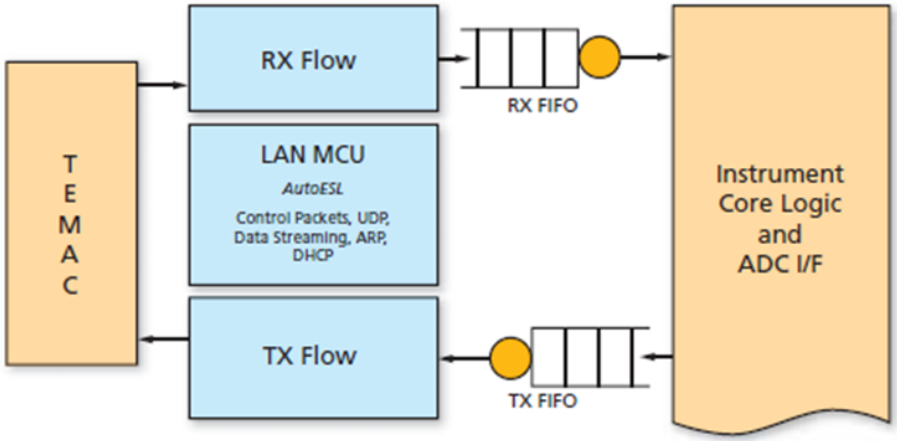


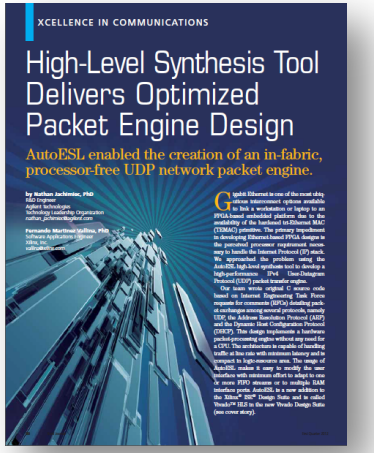
Metric	RTL Expert	AutoPilot Expert	Diff (%)
LUTs	32,708	29,060	-11%
Registers	44,885	31,000	-31%
DSP48s	225	201	-11%
BRAMs	128	99	-26%

TCAD April 2011 (keynote paper)
"High-Level Synthesis for FPGAs: From Prototyping to Deployment"

Xilinx Vivado HLS Results: UDP Network Package Engine (from Agilent & Xilinx)

- **Xcell Journal Issue 79**
  - Angilent: Nathan Jachimiec, PhD
  - Xilinx: Fernando M. Vallina, PhD
- **FSM based design**
  - Main modules
  - ARP and DHCP Flowch





LUTs	
Clock Cycles	
Vivado HLS versio	

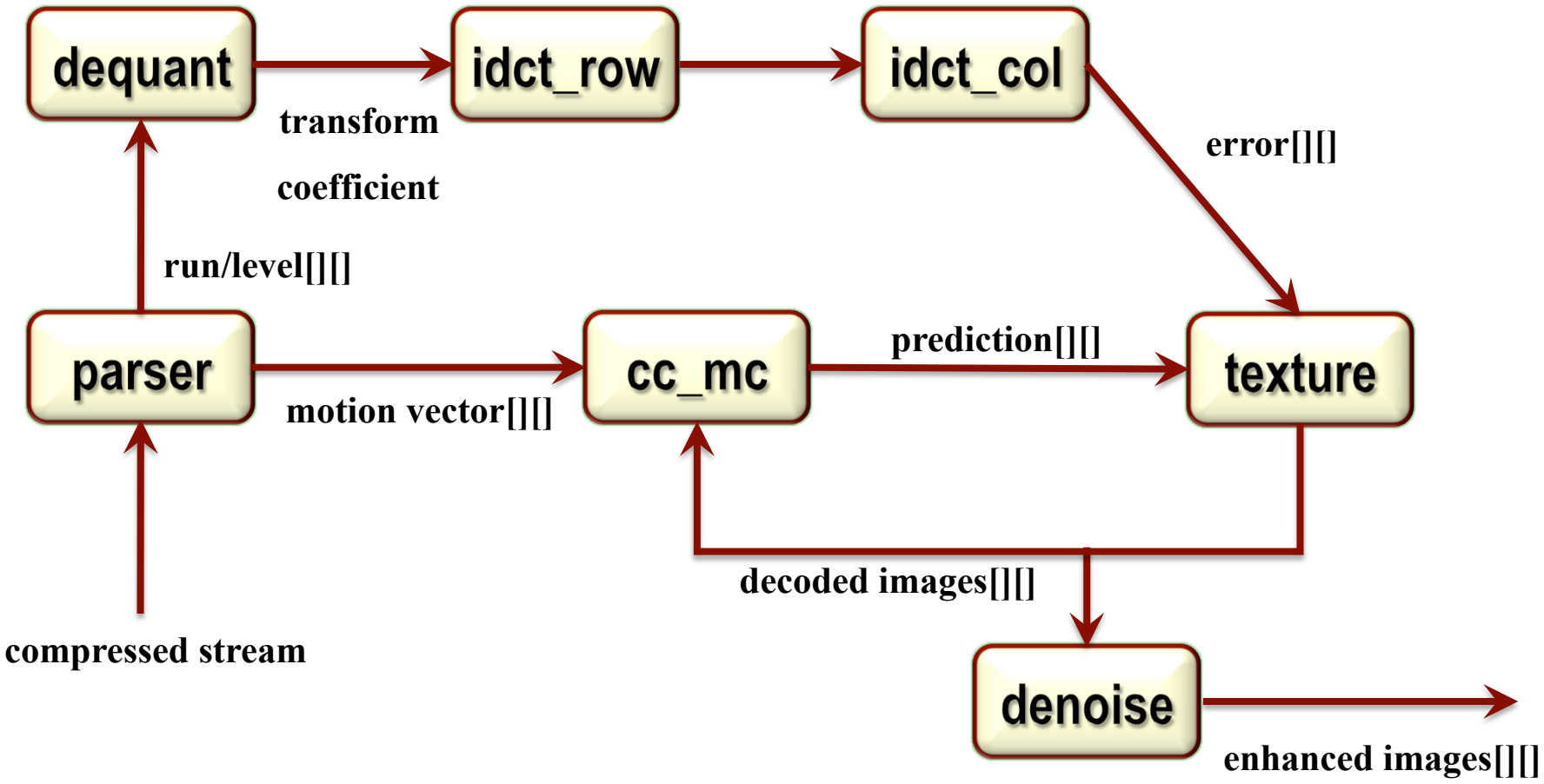
"In an HDL design, each scenario would likely cost an additional day of writing code and modifying the testbench to verify.

With Vivado HLS these changes took minutes"

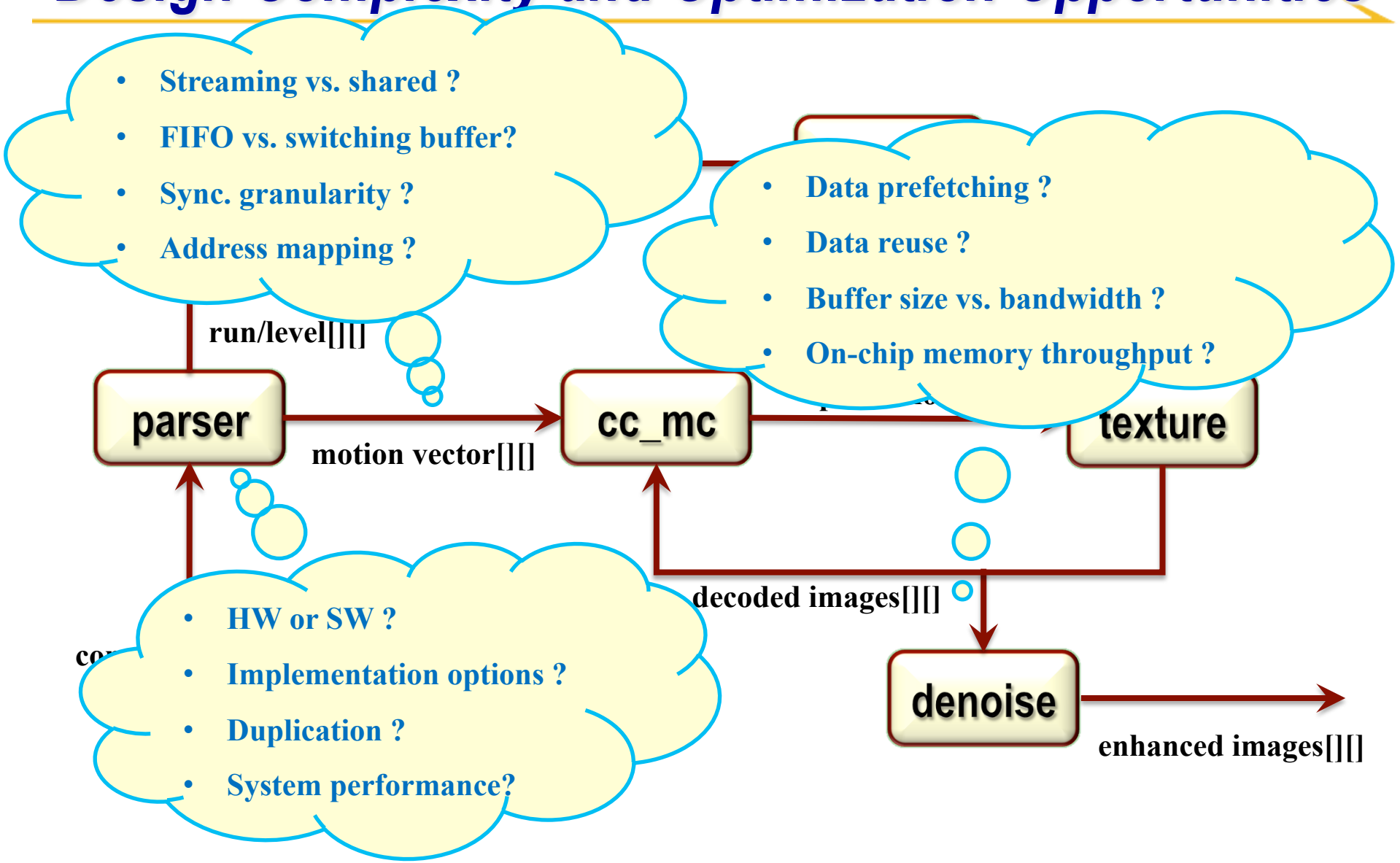
HLS Alone Is Not Enough

- A large design space for software and hardware co-design
- Also need automated source code transformation for HLS friendly C/C++ code to enable
  - Concurrent memory access
  - Data reuse and on-chip buffer generation
  - Data prefetching
  - …

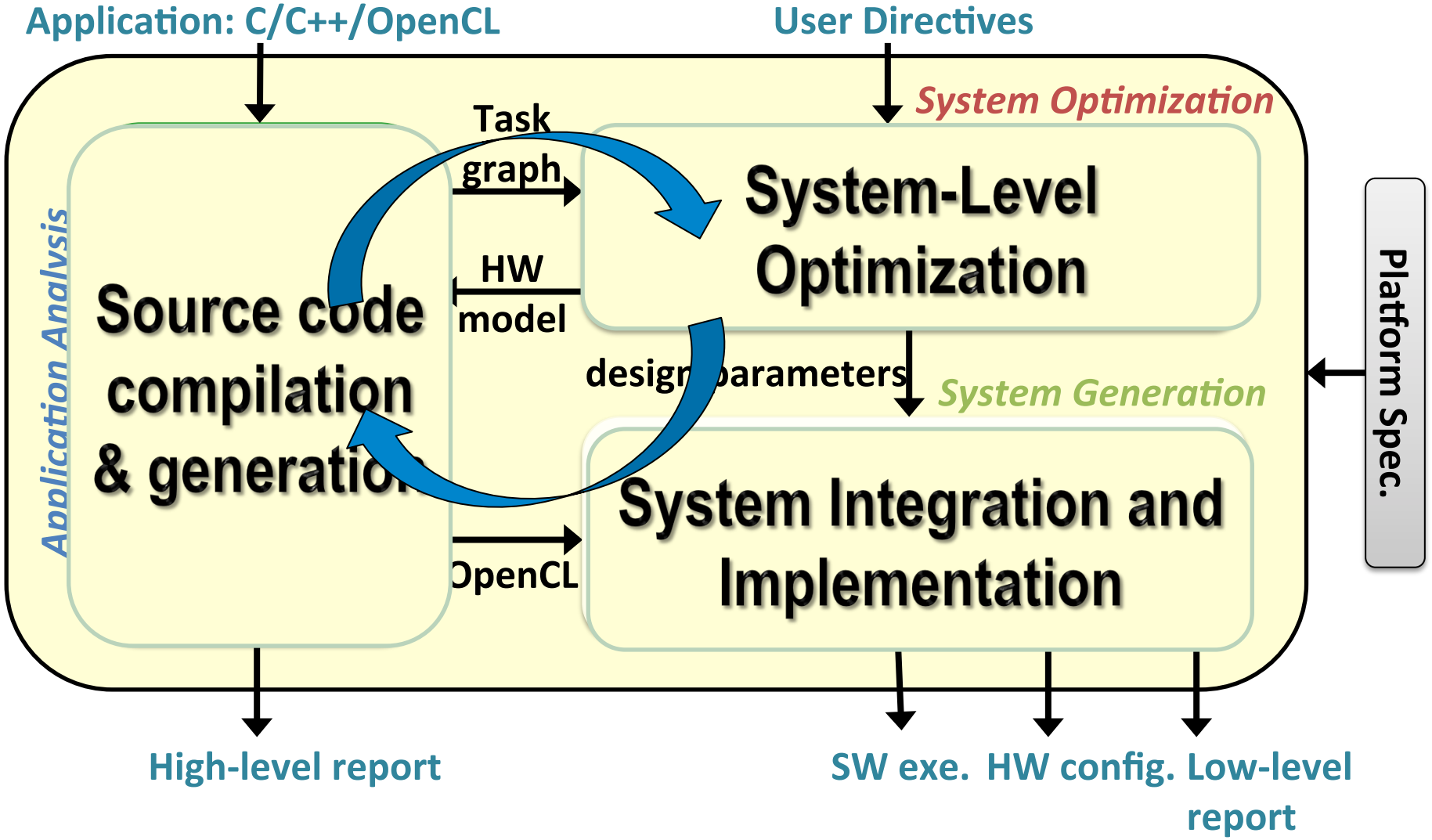
Design Complexity and Optimization Opportunities



Design Complexity and Optimization Opportunities



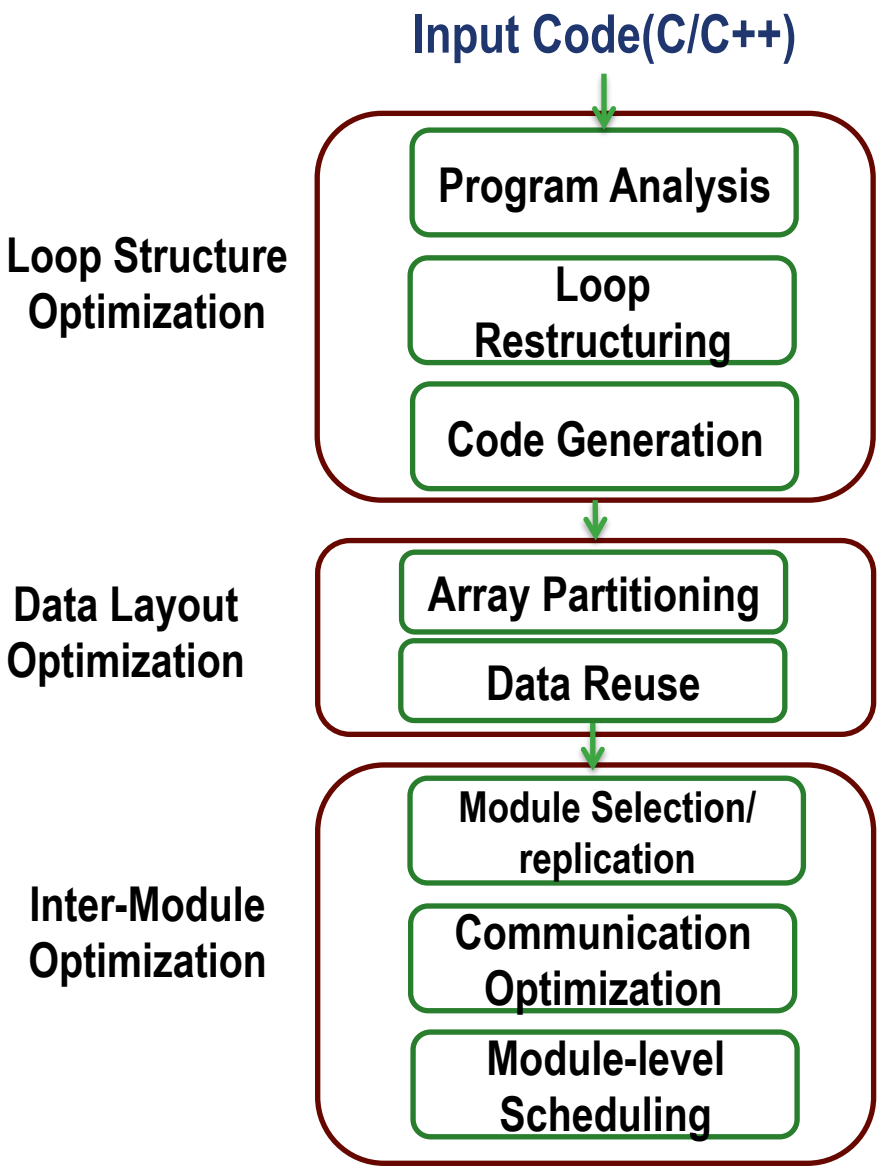
CMOST: Fully Automated Compilation and Mapping Flow



Optimization Beyond HLS

System-Level Optimization

Optimization Beyond HLS



Polyhedral-Based Data Reuse Optimization for Configurable Computing FPGA'13 Best Paper Award

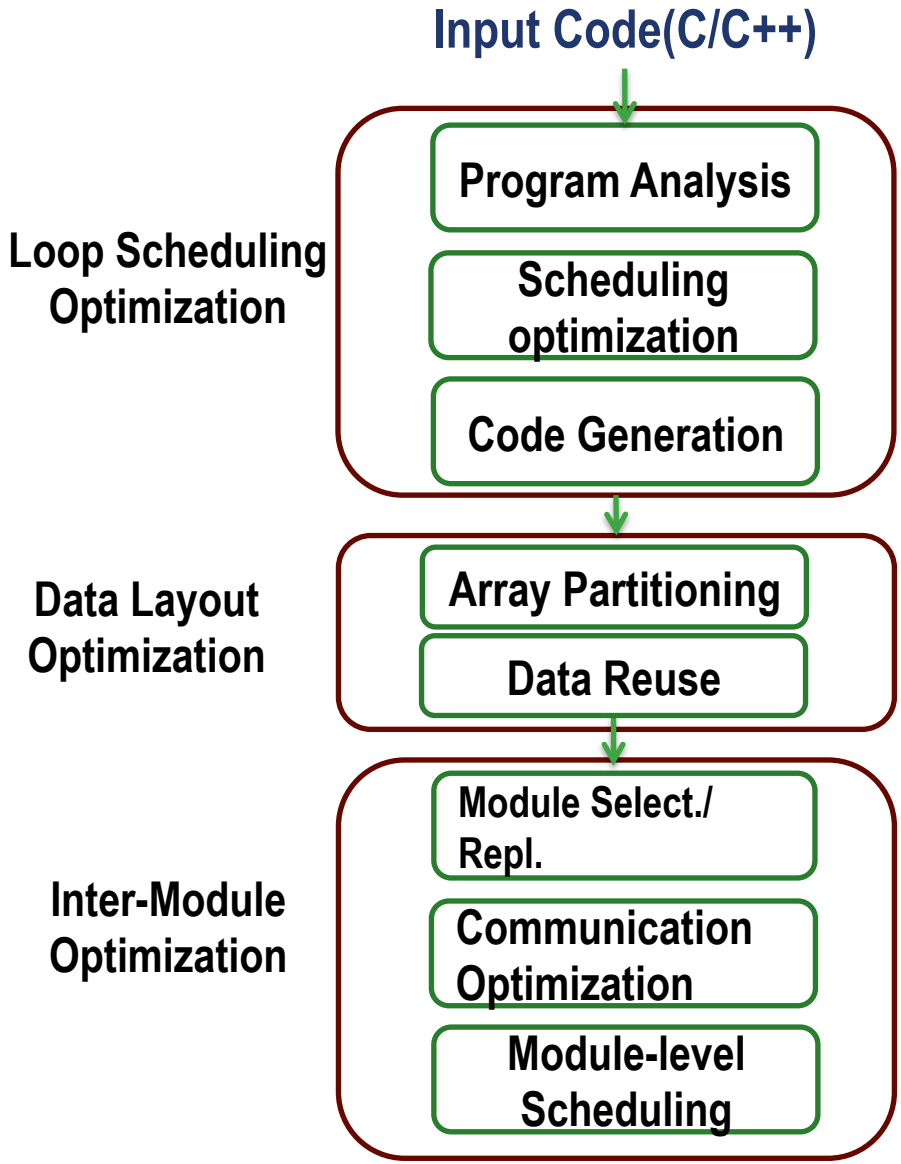
Improving Polyhedral Code Generation for High-Level Synthesis CODES-ISSS'14 Best Paper Award

Theory and Algorithm for Generalized Memory Partitioning in High-Level Synthesis, FPGA'14

An optimal microarchitecture for stencil computation acceleration based on non-uniform partitioning of data reuse buffers (DAC'14)

Combining Computation with Communication Optimization in System Synthesis for Streaming Applications, FPGA'14

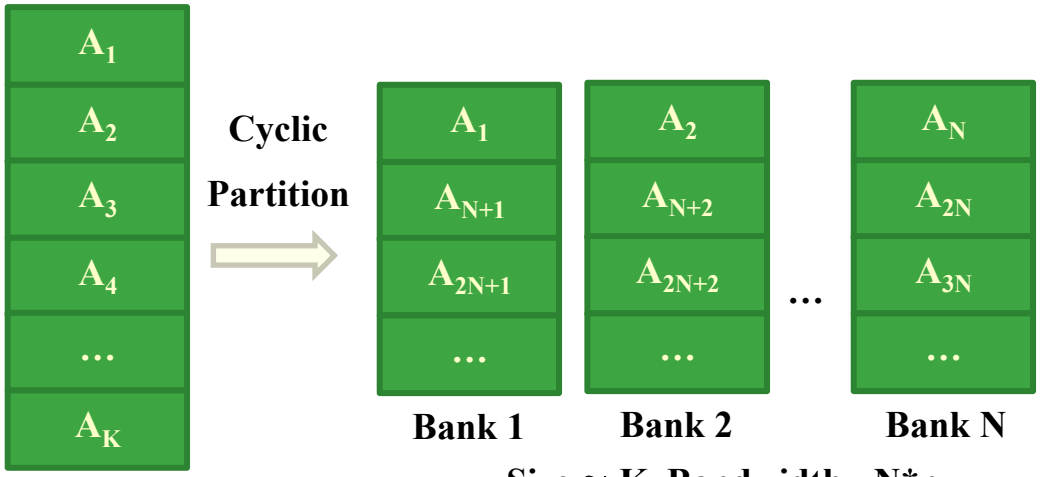
High-level Optimizations



Theory and Algorithm for Generalized Memory Partitioning in High-Level Synthesis, FPGA'14

Memory Partitioning for Throughput Optimization

- Memory is still a bottleneck
  - Data intensive applications: image/video
  - Loop unrolling/tiling/pipelining
- Memory partitioning

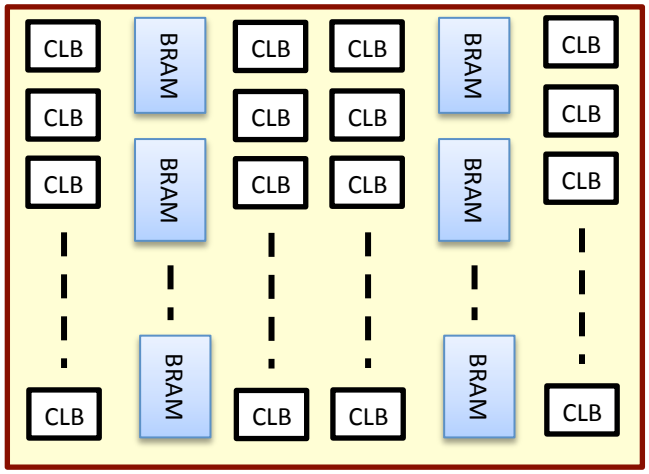


Size = K, Bandwidth = p

p: memory port number

Size ≈ K, Bandwidth= N*p

N: Partition Factor

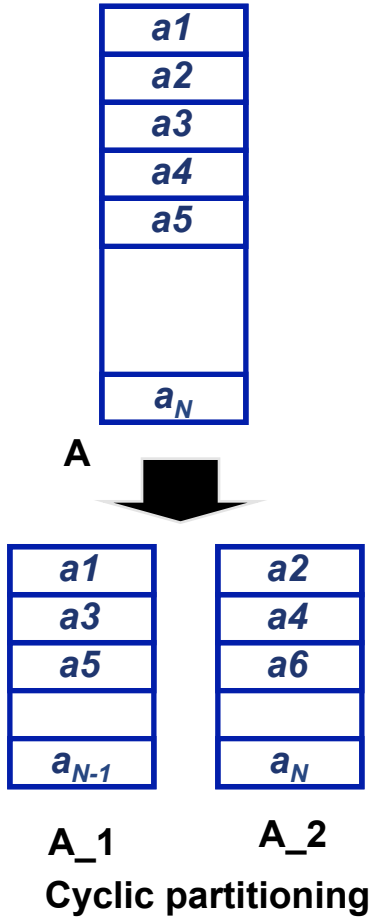


Challenge: generate conflict-free memory partitioning for a given program

Memory Partitioning for HLS*

- **Cyclic partitioning**
  - Easy to implement
  - Very effective in practice
- **Example**
  - Will i and (3*i+1) go to the same memory bank?
- **Theorem**

$$\forall i,\ a_1*i+b_1 \neq a_2*i+b_2 \bmod N$$
$$\Leftrightarrow \gcd(a_1-a_2, N) \nmid (b_1-b_2)$$



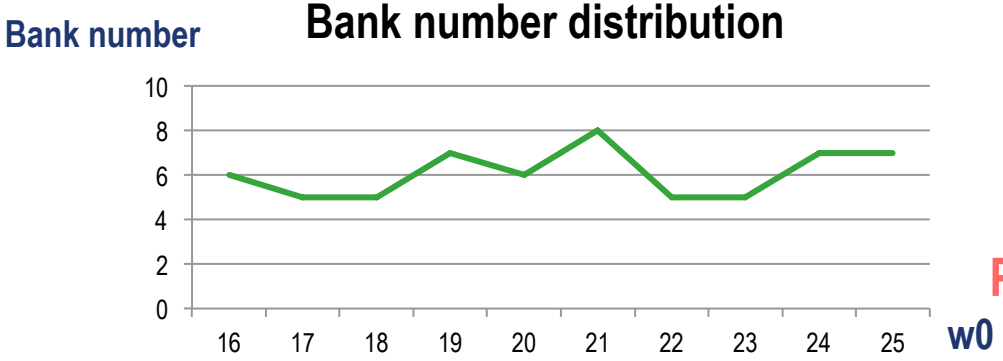
*J. Cong, W. Jiang, B. Liu and Y. Zou. ACM TODAES 2011 (Best Paper Award)

Memory Partitioning for Multidimensional Arrays

- **Flatten-Based Partitioning**
  - Flatten multidimensional array
  - Partition flattened single dimensional array

```
for (j=0; j<w1; j++)
  for (i=0; i<w0; i++)
    foo(A[j][i], A[j][i-1], A[j-1][i ], A[j+1][i], A[j][i+1]);
```

```
foo(A[w0 *j+i], A[w0 *j+i-1], A[w0 *j+i- w0], A[w0 *j+i+ w0], A[w0 *j+i +1]);
```



Conflict free conditions:

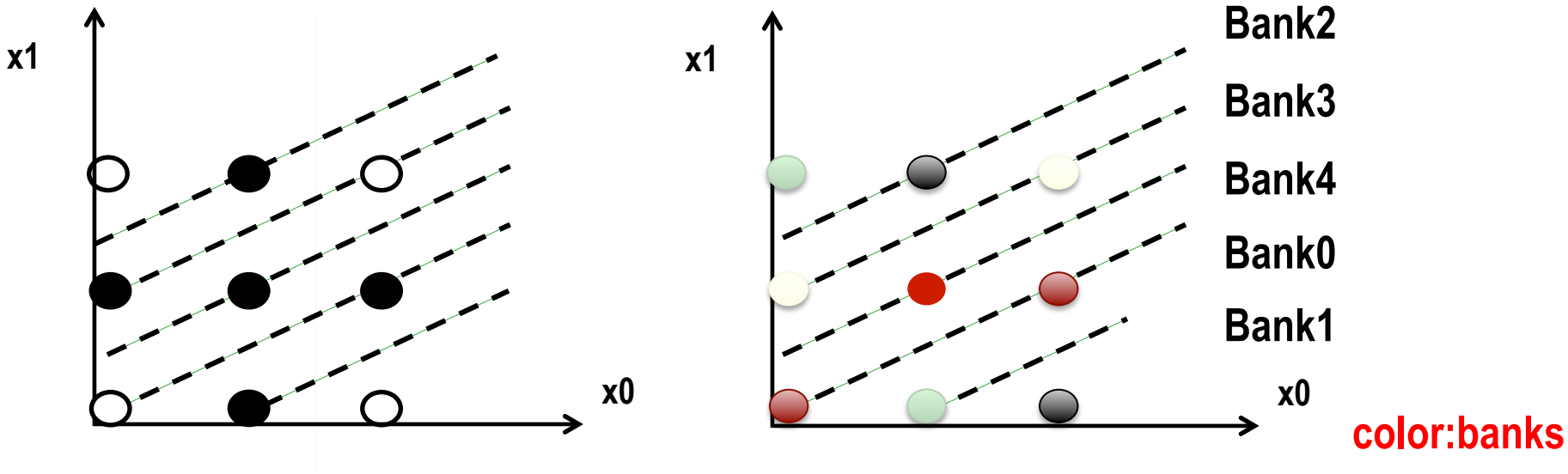
$N \nmid 2$ $N \nmid w0$

$N \nmid (w0-1)$ $N \nmid (w0+1)$

Partition results are related to array sizes!

Linear-Transformation-Based Partitioning*

- Linear transformation-based approach
  - Multidimensional address $\vec{x}$ linearization: $L(\vec{x}) = \vec{\alpha} \cdot \vec{x}$
  - Bank mapping: $\text{bank}(\vec{x}) = L(\vec{x}) \bmod N$ (Cyclic)
- Example: denoise

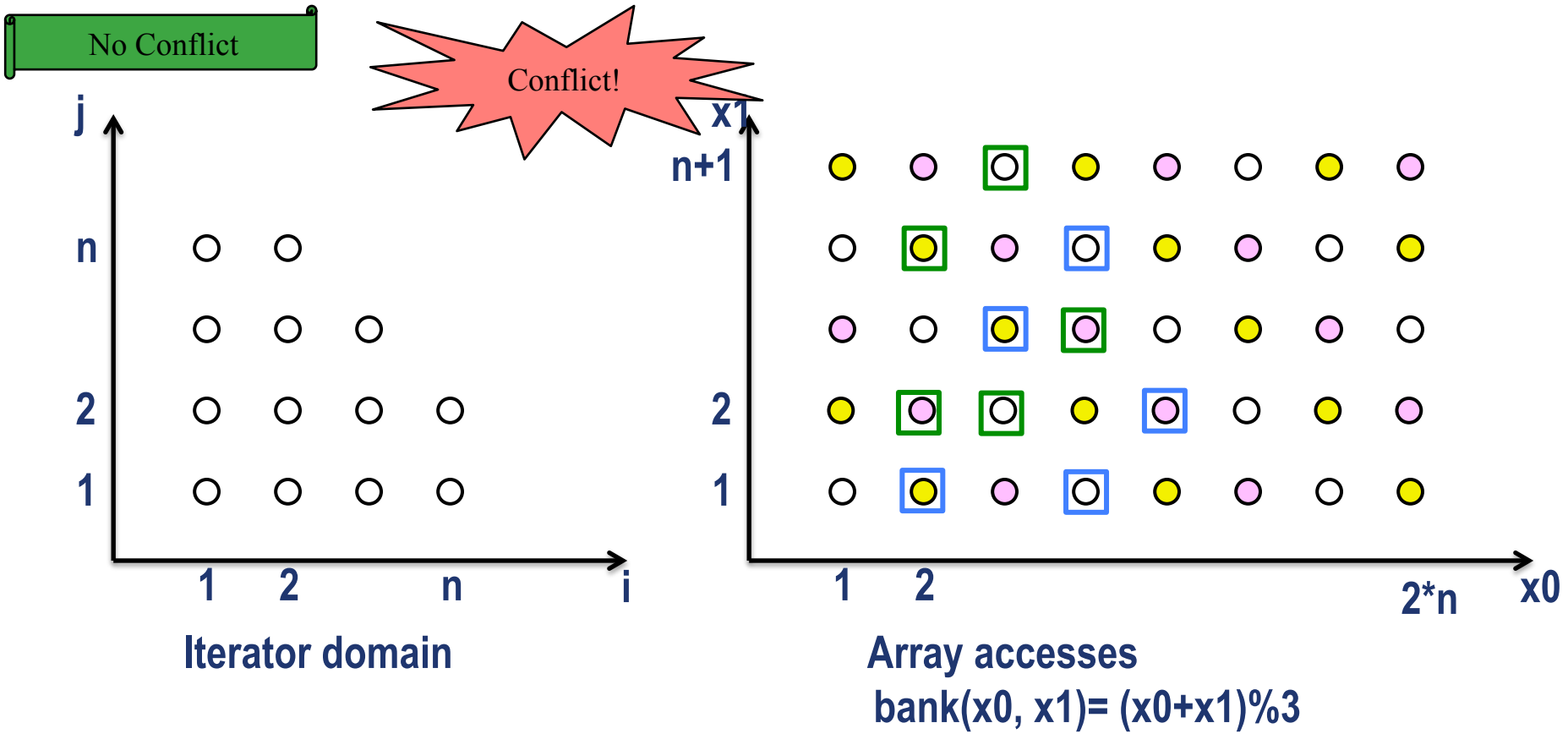


$\forall i,j$, $A[a_1*i+b_1][c_1*j+d_1]$ not conflict with $A[a_1*i+b_1][c_1*j+d_1]$
$\Leftrightarrow \gcd((\alpha_1 (a_1-a_2)+\alpha_2 (c_1-c_2)), N) \nmid (\alpha_1 (b_1-b_2)+\alpha_2 (d_1-d_2))$

*Y. Wang, P. Li, P. Zhang, C. Zhang and J. Cong. DAC 2013

Access Conflict

```
for (i = 1; i <= n; i++)
  for(j = 1; j <= min(n,n-i+2); j++)
    foo(A[j+1][i+1], A[j][2*i]);
```



Conflict Polytope

- *Conflict polytope* of two references is a subset of iteration domain where the two references are mapped on the same bank

$$\begin{cases} 1 \le i \le n \\ 1 \le j \le n \\ i + j \le n + 2 \\ (i+1) + (j+1) \equiv (2i + j) \bmod 3 \end{cases}$$

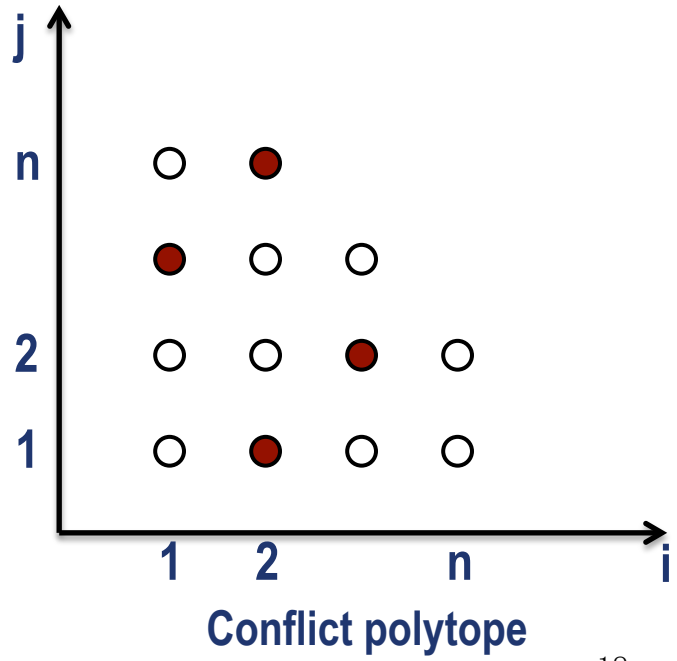
```
for (i = 1; i <= n; i++)
  for(j = 1; j <= min(n,n-i+2); j++)
    foo(A[j+1][i+1], A[j][2*i]);
```

- Insert an extra variable *k* to linearize

$$(i+1) + (j+1) = (2i + j) + 3k$$

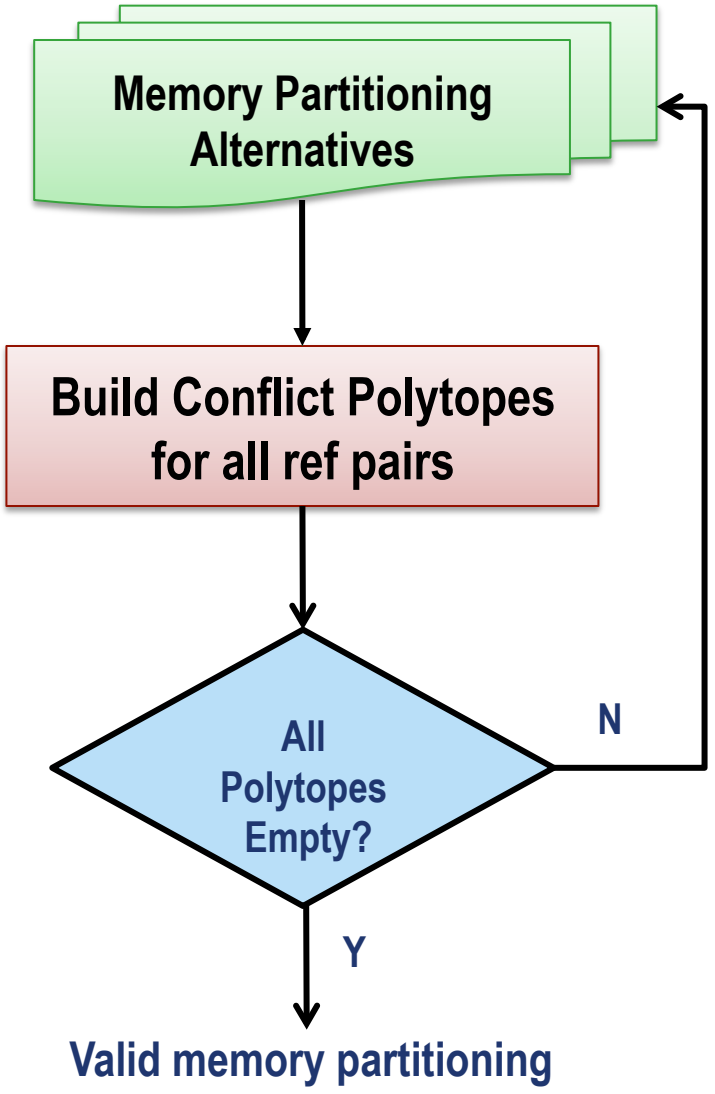
- Fourier–Motzkin Algo. (Fourier 1826, Motzkin1936)
  - Test the emptiness of the conflict polytope
  - Algorithm complexity: $O(m^{2^t})$
    - m: number of inequalities (m=4 in the example)
    - t: number of of variables (t=3 in the example)
    - independent of iteration domain size *n*

bank(x0, x1)= (x0+x1)%3



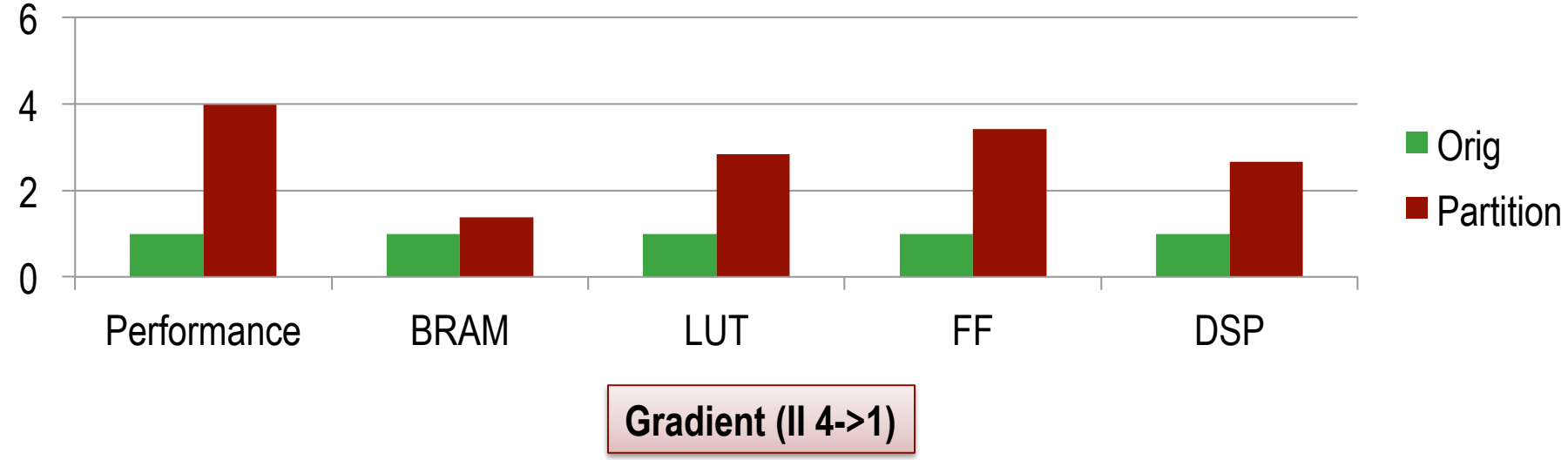
Conflict polytope

Generalized Memory Partitioning (GMP)

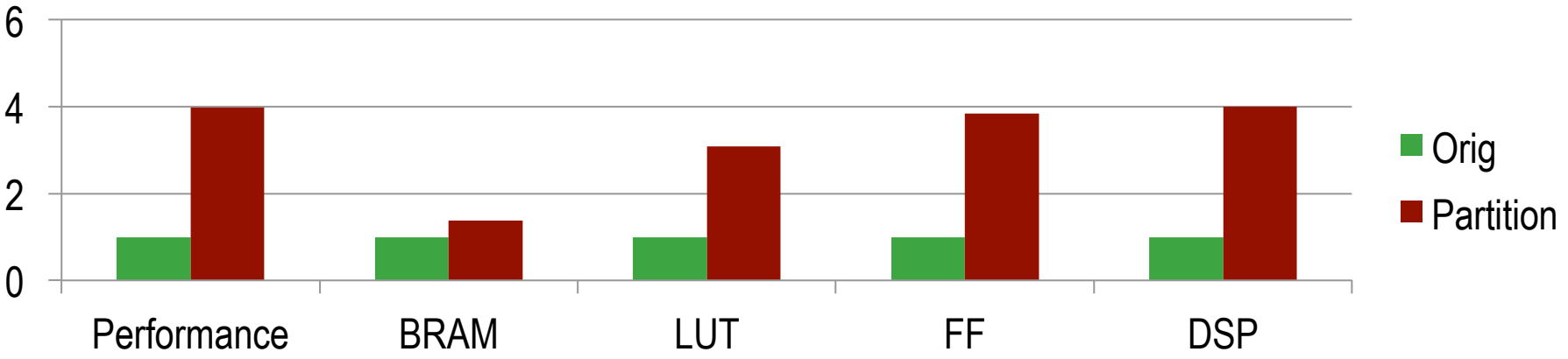


- Complexity independent of
  - Sizes of iteration domain
  - Sizes of array
- Complexity related to
  - Dimensions of iteration domain and array
    - In real case <= 3
  - Number of References
    - In real case <= 100
- Number of memory partitioning alternatives ≤ number of references

Experimental Results for Denoise

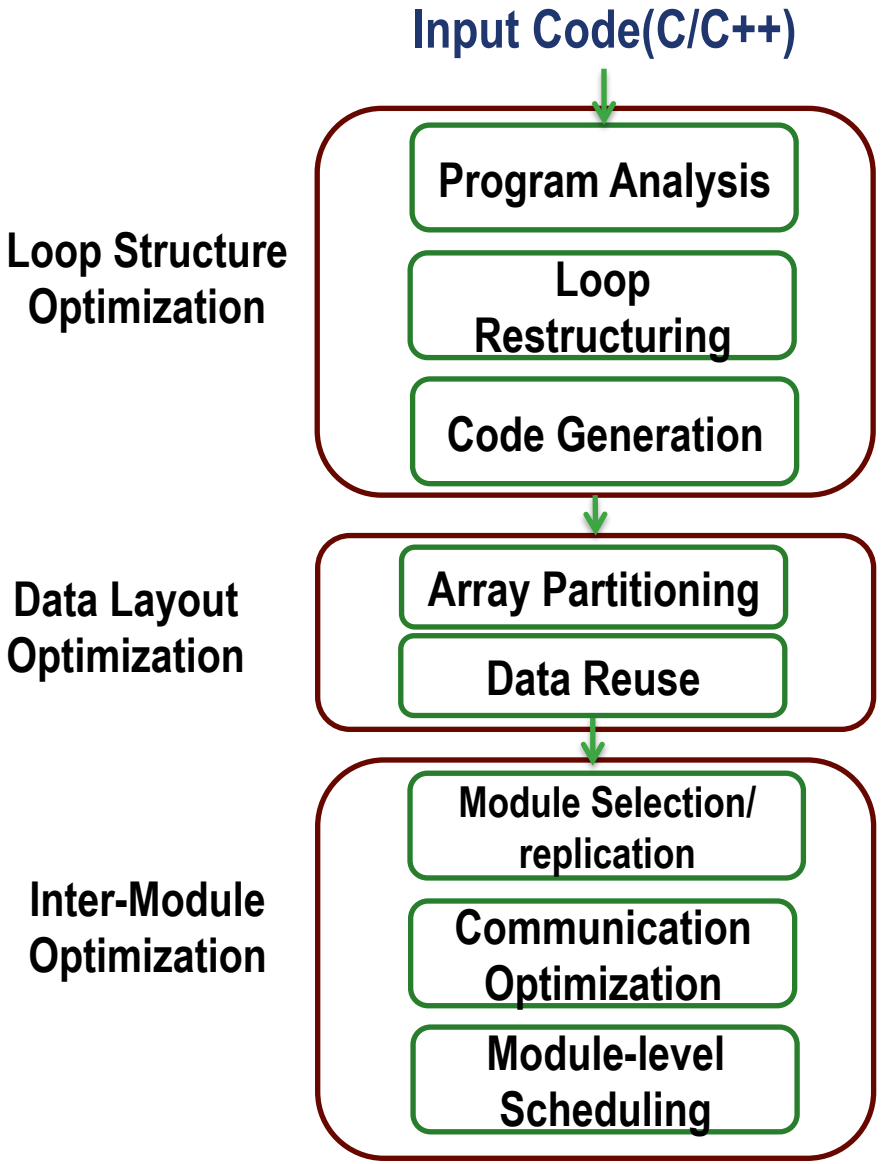


Gradient (II 4->1)



Rician (II 4->1)

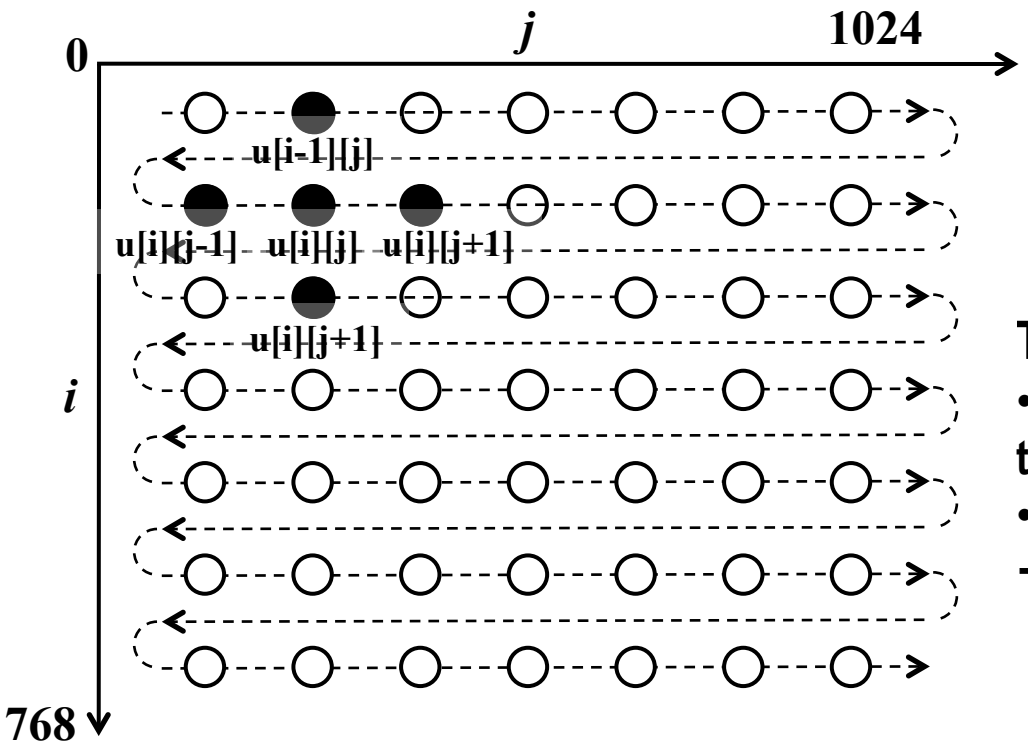
Optimization Beyond HLS



An optimal microarchitecture for stencil computation acceleration based on non-uniform partitioning of data reuse buffers (DAC'14)

Stencil Applications

- An important pattern in many application domains: image processing, constituent kernels in multigrid methods, partial differential equation solvers, etc



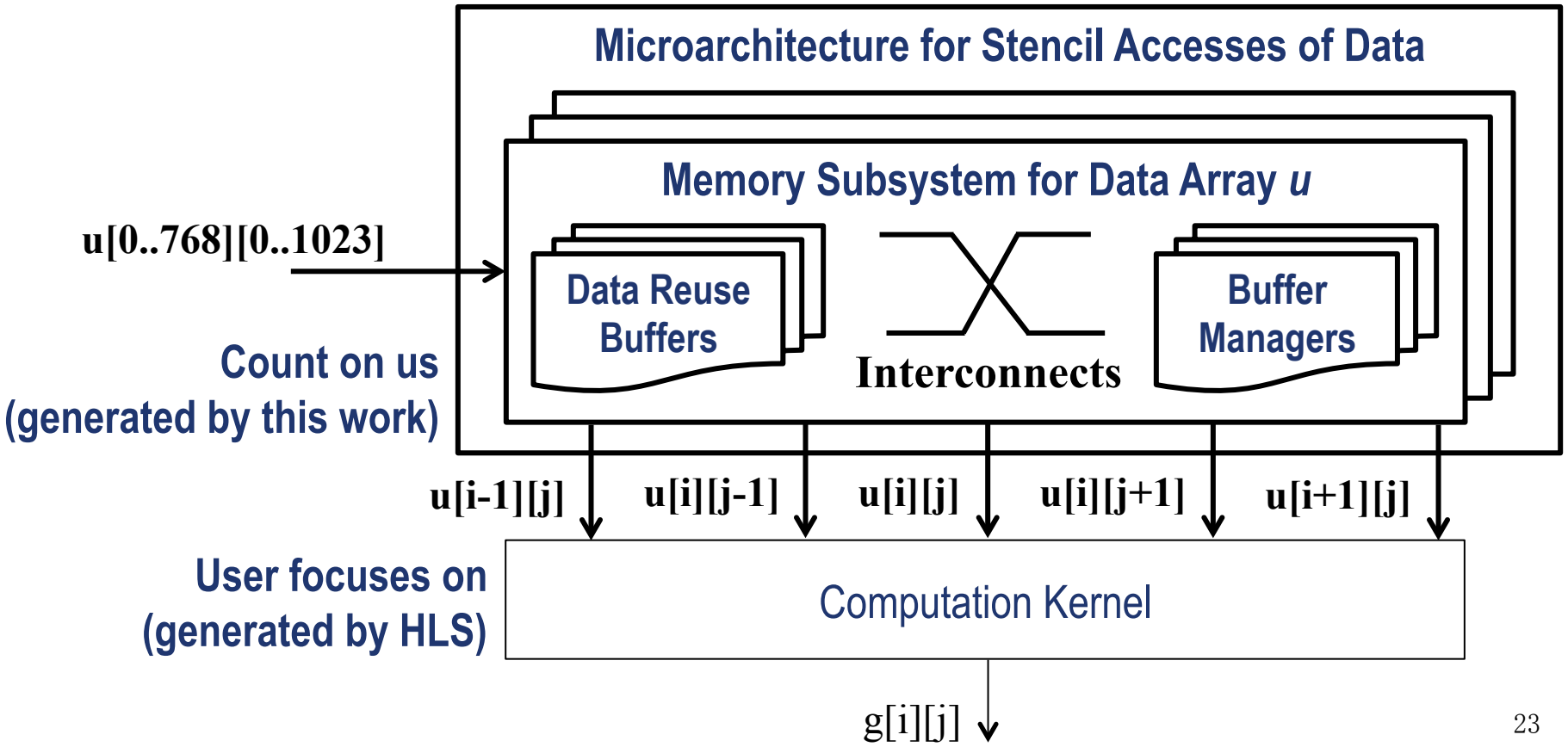

```
for (j = 1; j < 1023; j++)
  for (i = 1; i < 767; i++)
{
        g[j][i] = f1(u[j][i], u[j-1][i],
u[j+1][i], u[j][i-1], u[j][i+1]);
}
```

Two observations:

- Each data accessed for multiple time as time goes → data reuse via on-chip memory
- Multiple data accesses at each time point → partitioning of on-chip memory

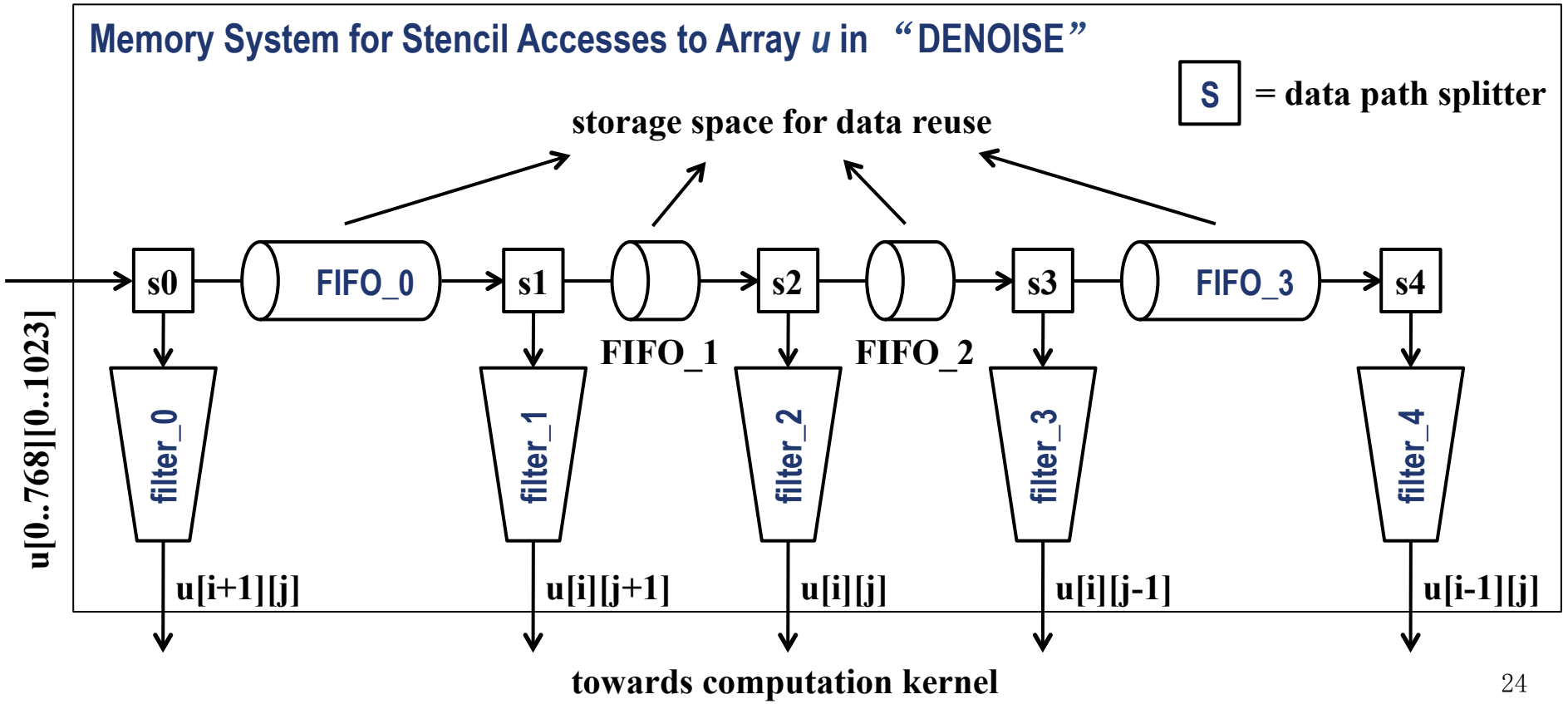
Microarchitecture for Stencil Applications (DAC'14)

- **Motivation**
  - Save accelerator designers from struggling with memory access optimization
  - We give the optimal solution in our architecture for any stencil access pattern!



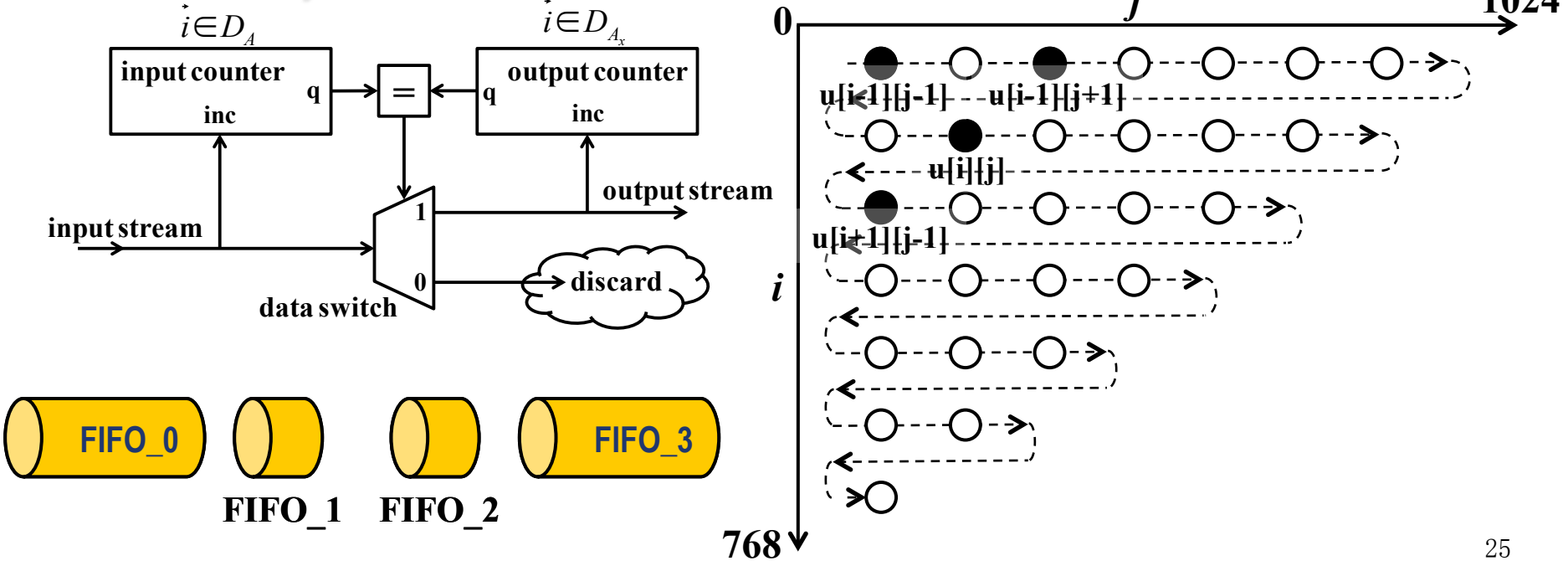
Our Microarchitecture w/ Non-Uniform Buffers

- Each data reuse buffer has a different size
- Use distributed controllers based on data streaming



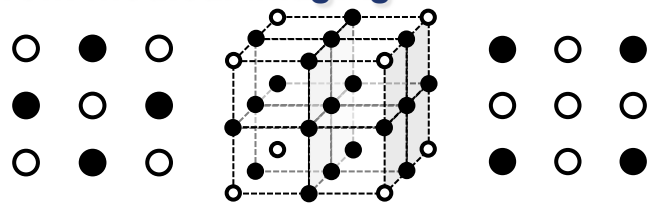
Features of Our Microarchitecture

- Full design space → can always achieve the optimal design point
  - minimum reuse buffer size
  - minimum # of reuse buffer banks
- Simple logic in each distributed component works together to achieve automatic filling of reuse buffers
- Automatic adjustment of reuse data amount



Experiments

- Real-life stencil computation kernels with non-rectangular stencil windows
  - DENOISE (2D/3D), RICIAN (2D), SEGMENTATION (3D) from medical imaging
  - BUCUBIC (2D) from bicubic interpolation process
  - SOBEL (2D) from Sobel edge detection algorithm



- Non-Uniform Buffer (NUB) vs Generalized Memory Partitioning (GMP)
- Always produces the minimum # of buffer banks and the minimum buffer size

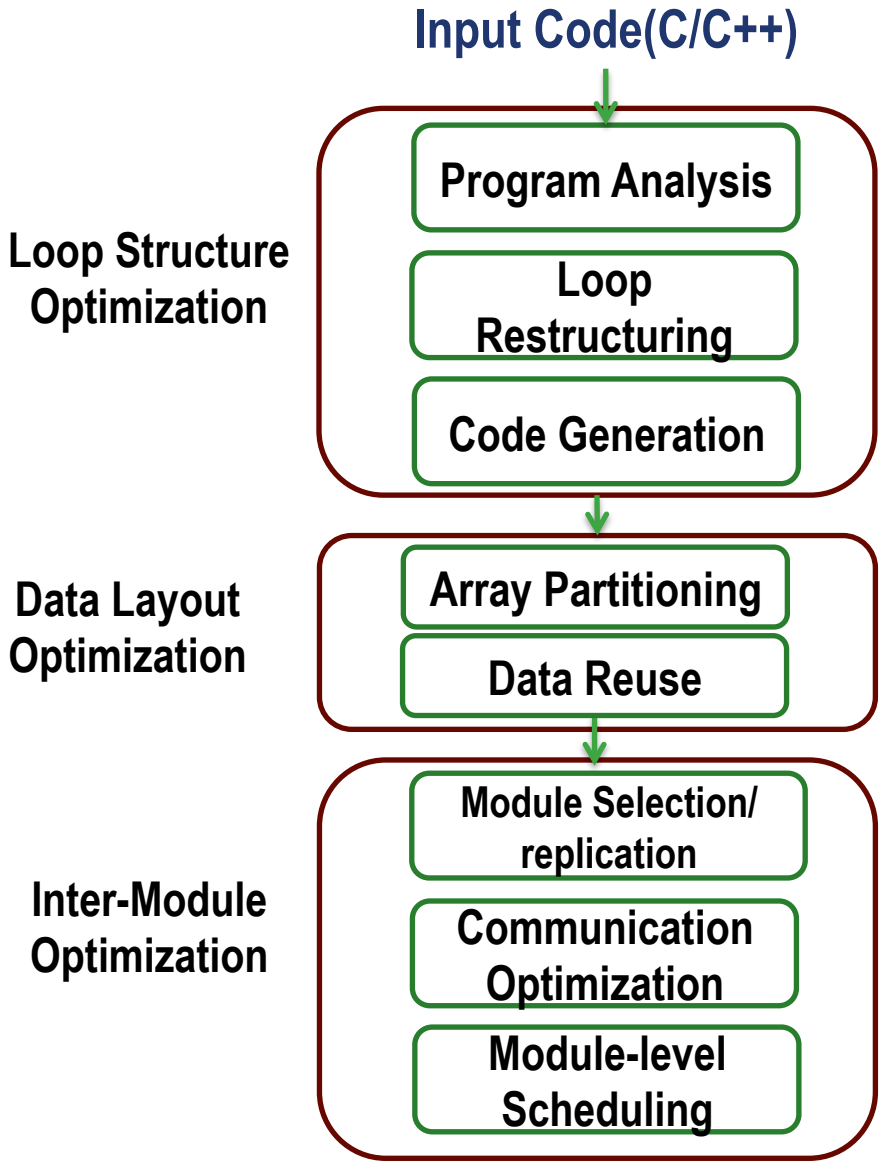
	Original Throughput	Achieved Throughput	# of Buffer Banks		Total Buffer Size	
			GMP	NUB	GMP	NUB
DENOISE	1/5	1	5	4	2050	2048
RICIAN	1/4	1	4	3	2050	2048
SOBEL	1/9	1	9	8	2054	2050
BICUBIC	1/4	1	4	3	2050	2048
DENOISE_3D	1/7	1	7	6	2240	2048
SEGMENTATION	1/19	1	20	18	2630	2112

Experiments (Cont'd)

- Synthesize for Xilinx Virtex7 FPGA XC7VX485T via ISE 14.2

		BRAM	Slice	DSP	CP(ns)
DENOISE	GMP	5	703	0	4.502
	NUB	2	636	0	4.519
RICIAN	GMP	4	582	0	4.472
	NUB	2	544	0	4.337
SOBEL	GMP	9	1937	0	4.416
	NUB	2	1088	0	4.239
BICUBIC	GMP	4	535	0	4.309
	NUB	2	493	0	4.196
DENOISE_3D	GMP	7	980	0	4.656
	NUB	2	859	0	4.762
SEGMENTATION	GMP	20	7533	0	4.995
	NUB	2	2251	0	4.985
Avg Savings		66%	25%	-	1.2%

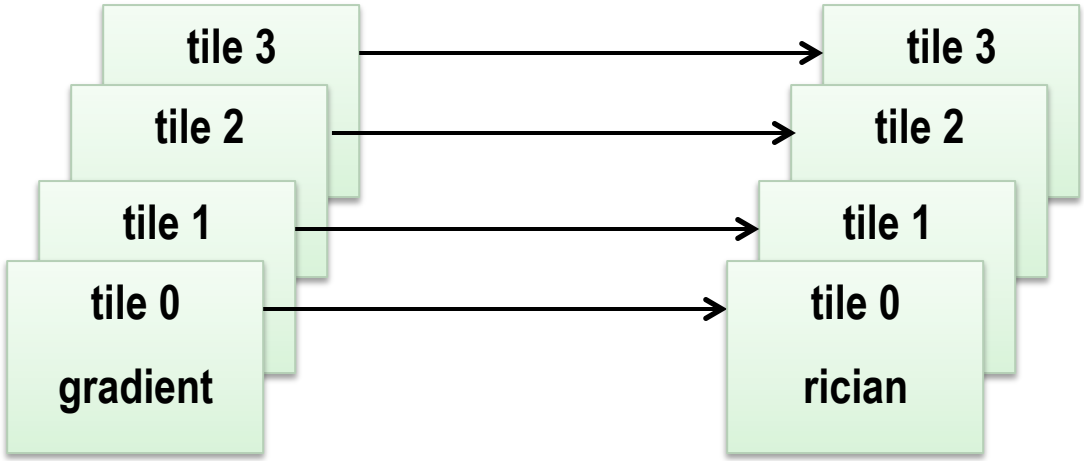
Optimization Beyond HLS



Combining Computation with Communication Optimization in System Synthesis for Streaming Applications, FPGA'14

Motivation

Tile size: 32x32
Image: 64x64, 4 tiles

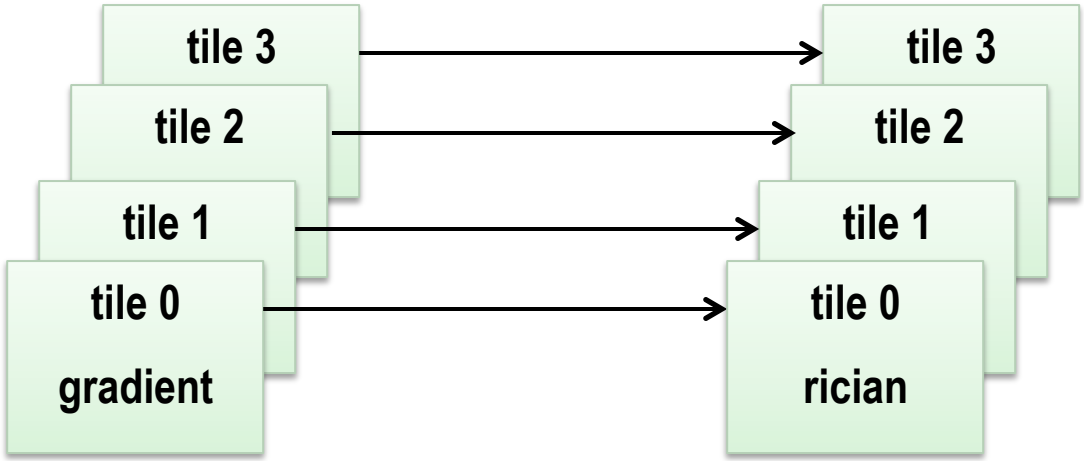


- Which implementation to use for each module?
  - Memory partitioned v.s. Memory non-partitioned

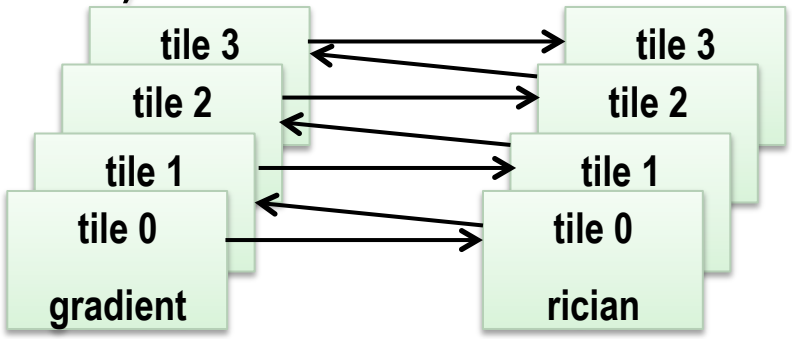
	BRAM	DSP	FF	LUT
non-partitioned gradient	128	21	2511	2125
partitioned gradient	176	56	7147	7262
partitioned rician	128	22	4692	3991
non-partitioned rician	176	88	14475	15537

Motivation

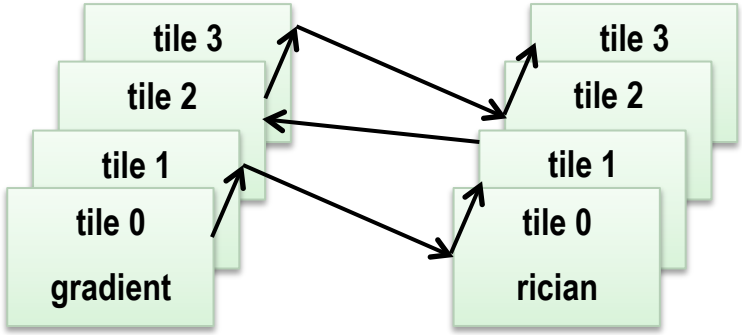
Tile size: 32x32
Image: 64x64, 4 tiles



- How many number of replicas?
- Scheduling and Communication cost (number of tiles in the communication channel)?



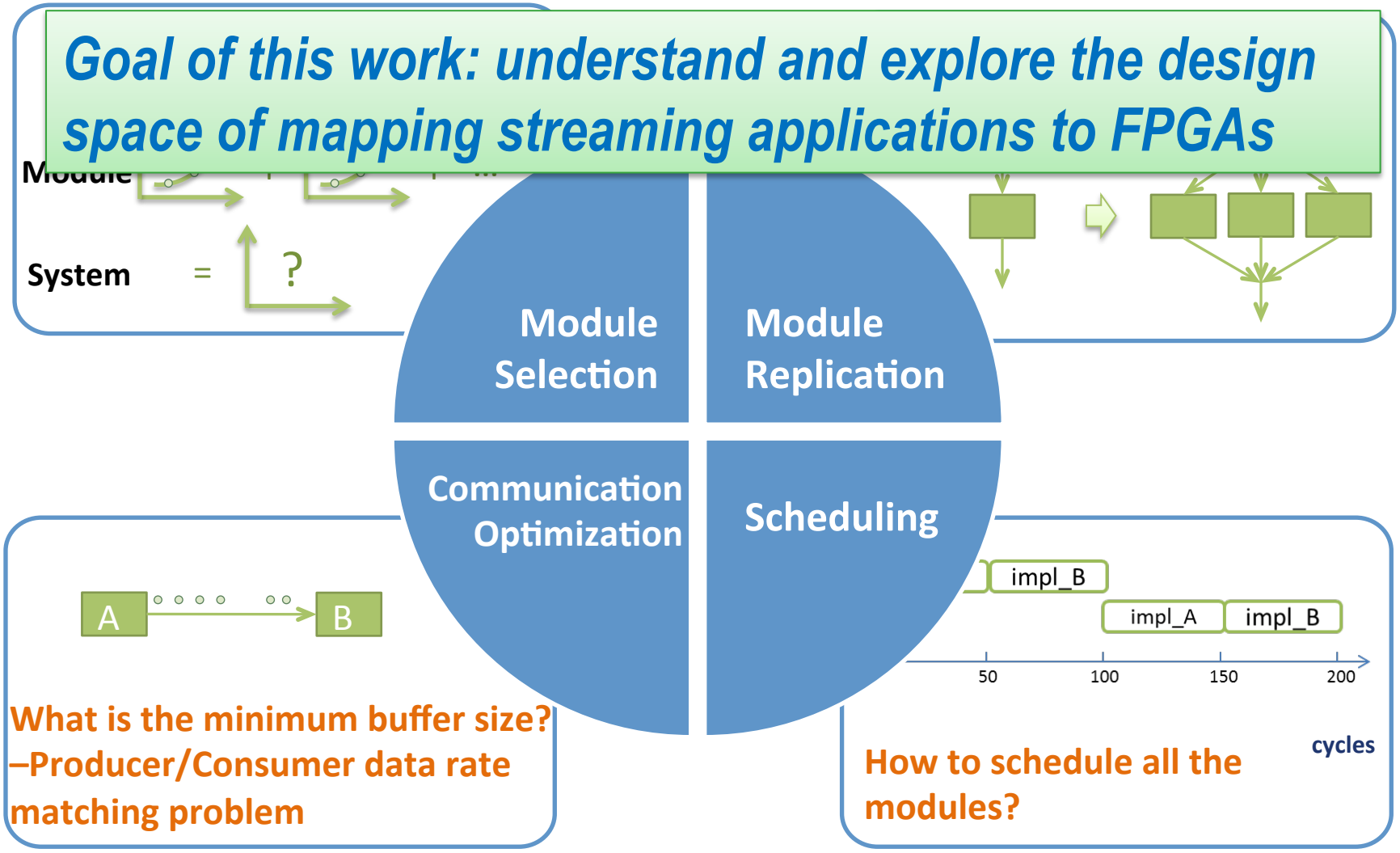
scheduling 0 →1 tile



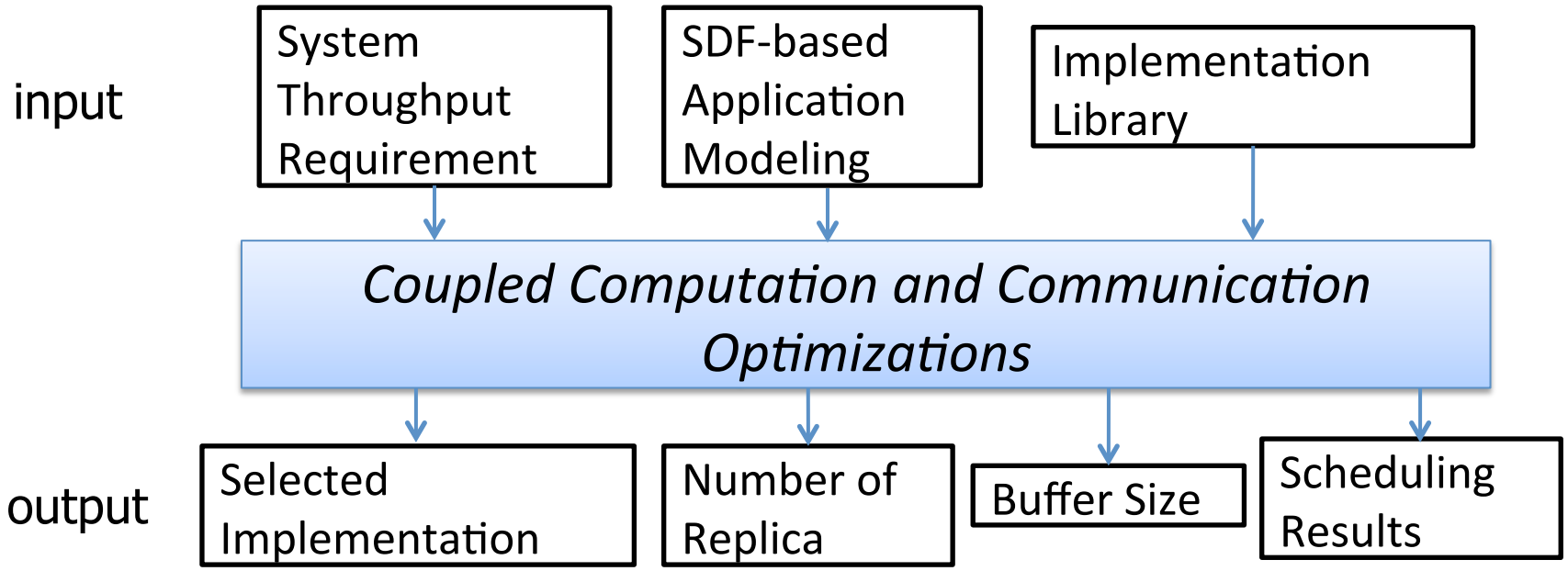
scheduling 0 →2 tiles

System-Level Synthesis in HLS for Streaming Applications

Goal of this work: understand and explore the design space of mapping streaming applications to FPGAs

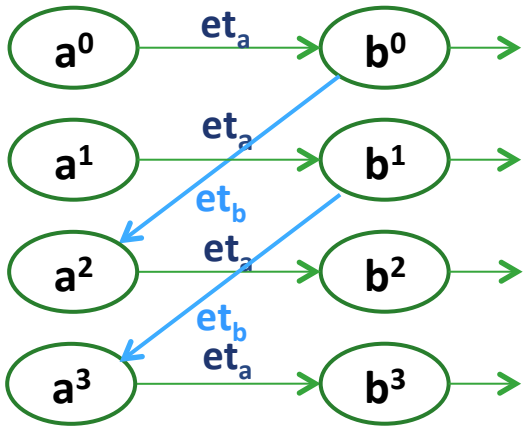


Proposed System Synthesis Framework



Formulation (1/2)

- Derive a scheduling graph
  - Associate each node with a time variable, denoting the starting time of the node
  - Scheduling graph: delineates all the scheduling constraints
    - Module latency, Module replication, System throughput requirement, Buffer constraints

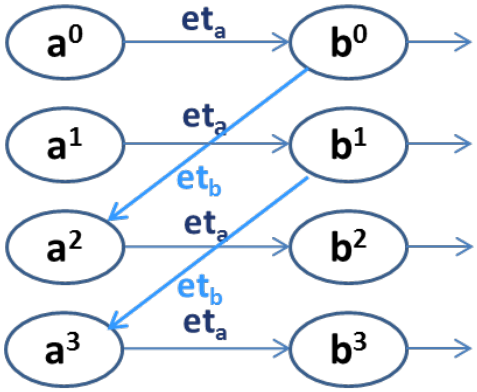


Module latency constraints
et_a: execution time of task a
et_b: execution time of task b

Buffer Constraints
If buffer size between a and b is 2, then add edges: $b^0 \rightarrow a^2$ $b^1 \rightarrow a^3$

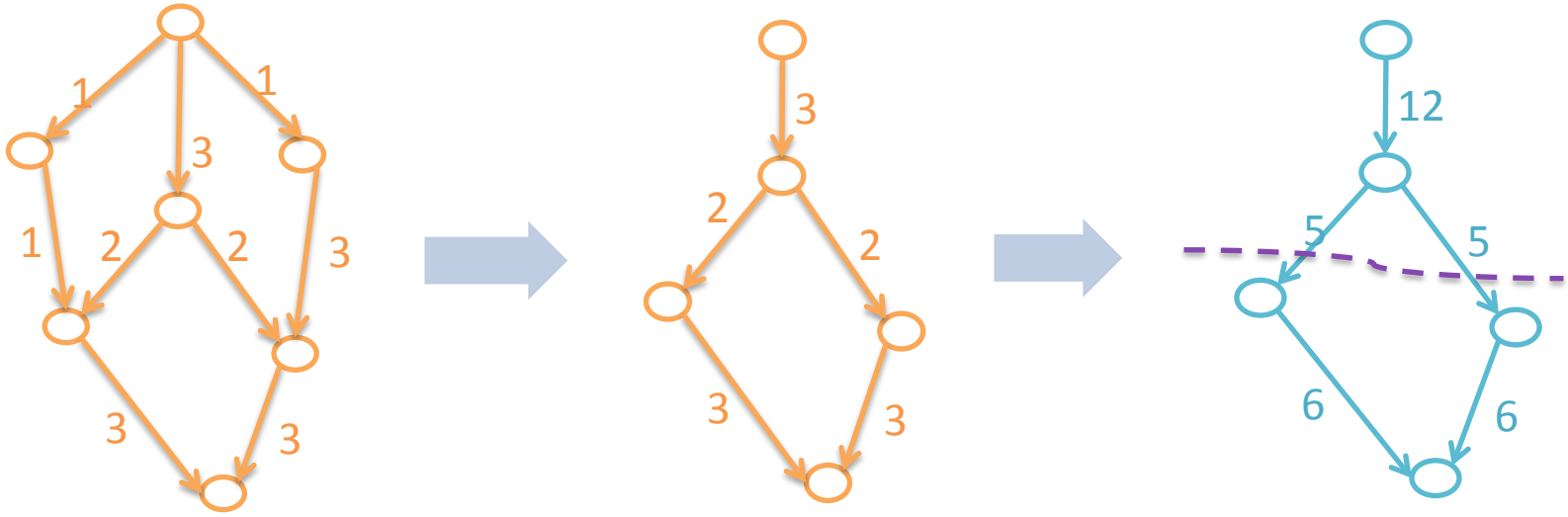
Formulation (2/2)

- Associate each node with a scheduling variable
  - $t(b^0) - t(a^0) >= et_a$
  - $t(a^2) - t(b^0) >= et_b$
  - ...
  - Scheduling variables are integer variables



- Schedulability checking problem is a ***System of Difference Constraints (SDC)*** problem
  - It can be solved optimally *in polynomial time* by linear programming relaxation
  - And the solution is guaranteed to be integers

Exploration



Scheduling Graph

- Find the length of longest path *(maxL)*
- In this example, *maxL* = 8

Find critical paths

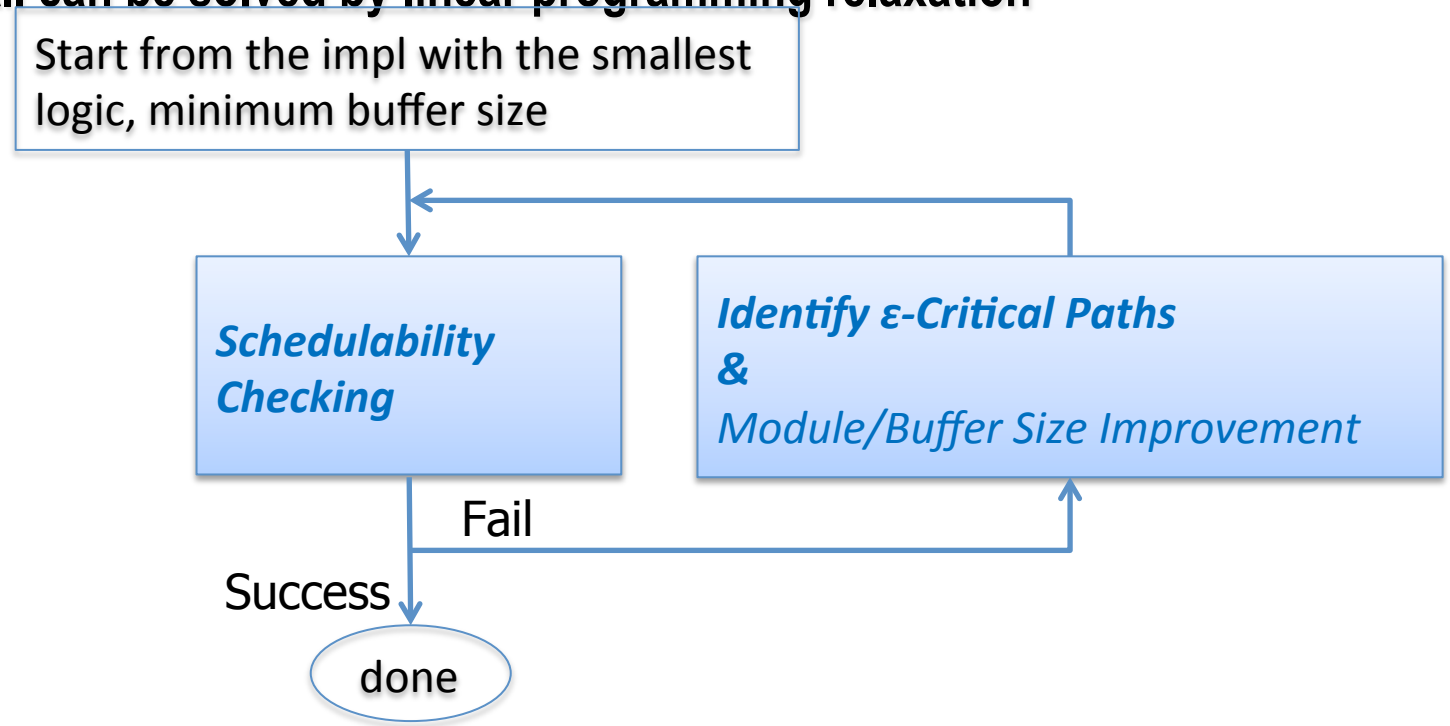
- Find all the paths whose lengths are *maxL,*
- **o**r more aggressively, *(1-ε)*maxL*

Module Improvement

- Associate each edge a new weight – the area penalty to remove this edge from the critical paths
- Find a minimum cut on the graph

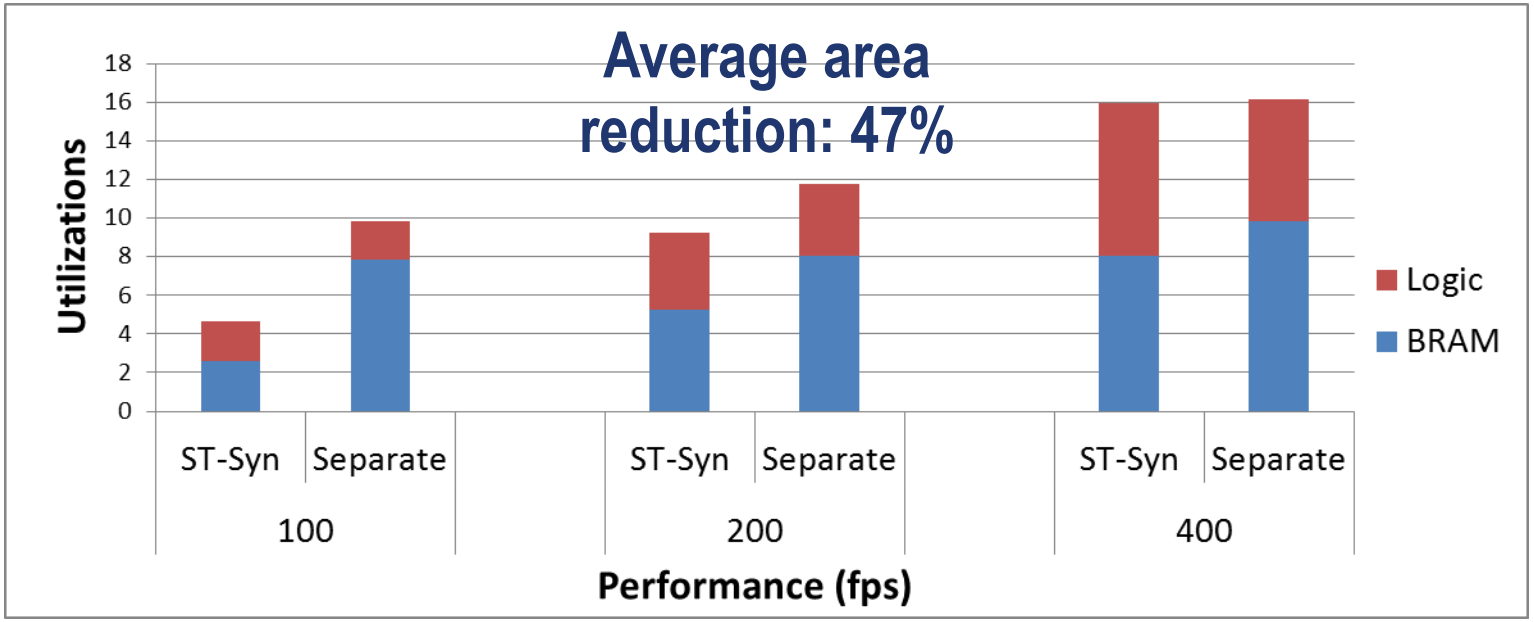
Streaming Synthesis (ST-Syn)

- ***Formulation – Schedulability checking***
  - System of Difference Constraint Problem
- ***Exploration – Identify critical path, module/buffer improvement***
  - Find ε-critical paths in the scheduling graph
  - Minimum cut problem
- **All can be solved by linear programming relaxation**



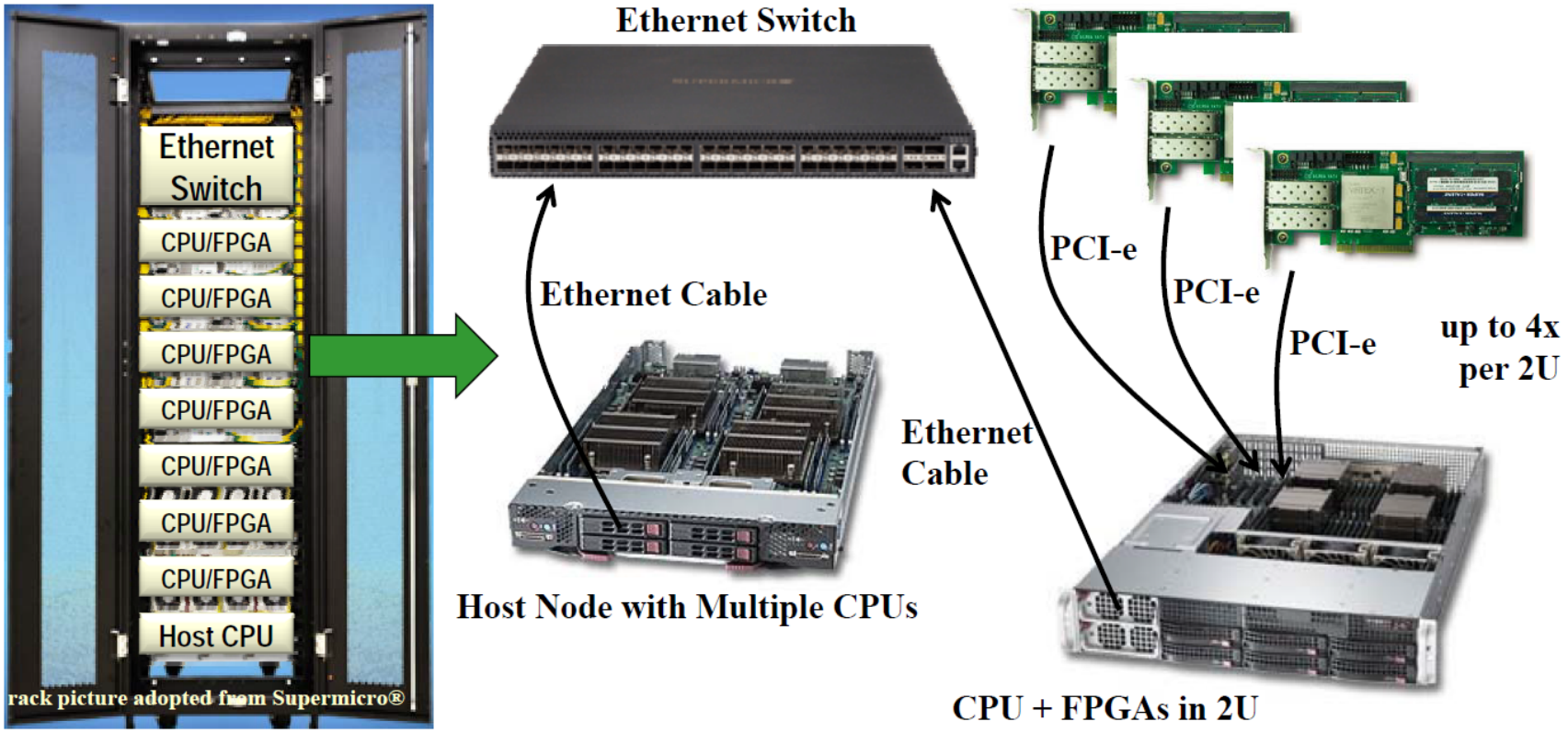
Experiments on Denoise

- Our methodology: ST-Syn
  - computation & communication co-optimization
- Separate:
  - separate computation opt. + communication opt.
- → Communication and computation should be considered in a unified framework



Ongoing Work:
Enabling Customized Computing in Data Centers

- Big Data Era → Massive Data-Level Parallelism → Supercomputer in a Rack
- A rack with tens of FPGA boards connected by heterogeneous interconnects
- Task/data mapping for minimum traffic, highest throughput, energy efficiency, etc



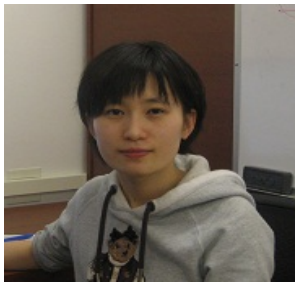
Concluding Remarks

- High-level synthesis (HLS) is gaining wider adoption
- Input C/C++ code may have a significant impact on HLS results
- CMOST: Automated source-code level transformation and optimization techniques for HLS and customized computing
- Ongoing work: large-scale customized computing in data centers
  - Automated accelerator scheduling and management is key

Acknowledgements

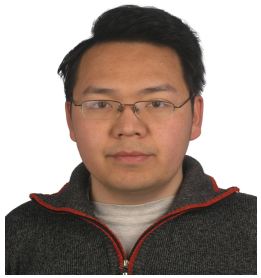
- CDSC supported by the NSF Expeditions in Computing Award
- C-FAR: one of six STARnet centers

Prof. Deming Chen (UIUC/ADSC)



Hui Huang (UCLA)

Muhuan Huang (UCLA)

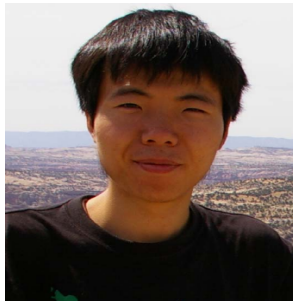


Dr. Peng Li (UCLA)

Prof. Louis-Noël Pouchet (UCLA)

Prof. P. Sadayappan (OSU)

Yuxin Wang (PKU)



Bingjun Xiao (UCLA)

Dr. Peng Zhang (UCLA)

Wei Zuo (UIUC)