



High-Level Synthesis Revisited Progress and Applications

Jason Cong

Chancellor's Professor, UCLA

Director, Center for Domain-Specific Computing

Co-Director, PKU/UCLA Joint Research Institute in Science and Engineering

cong@cs.ucla.edu

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



Outline

- ◆ **Brief review of history of HLS**
- ◆ **The UCLA effort**
 - Original motivation
 - The Wave of ESL Synthesis
 - Commercialization
- ◆ **Highlights of some algorithmic innovations**
- ◆ **Applications**
 - Used in design of accelerator-centric architectures

High-Level Synthesis: A Brief History

◆ Early attempts

- Research projects
- 1980s ~ early 1990s

◆ Early commercialization

- Tools from major EDA vendors
- 1990s~early 2000s

◆ Recent efforts

- Major vendors + startups
- 2000 ~ present

Recent Efforts

◆ A new generation of tools emerge after 2000

◆ Capture design using C-based languages

- C/C++/SystemC
 - Extensions/pragmas/directives to overcome limitation of C
- Others: Matlab, BlueSpec, Esterel, etc.
- Nobody seems to use VHDL/Verilog for HLS now

◆ A plethora of new HLS systems starting from C-based languages

- Academic or chip providers: ASC (Imperial College), CASH (CMU), C2H (Altera), DIME-C (Nallatech), GAUT (UBS, France), ROCCC(UCR), Spark (UCSD), Trident (Los Alamos National Lab), xPilot (UCLA)
- EDA vendors: AutoESL (now Xilinx), Cadence (C-to-Silicon), Celoxica (now Mentor DK Design Suite), Forte (Cynthesizer), Impulse C, Mentor Catapult (sold to Calypto), NEC (Cyber), Synfora (acquired by Synopsys)

◆ Many chips delivered, ongoing adoption

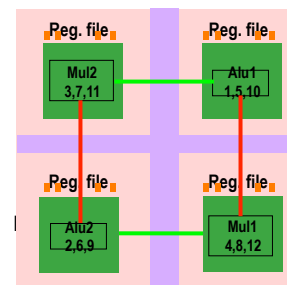
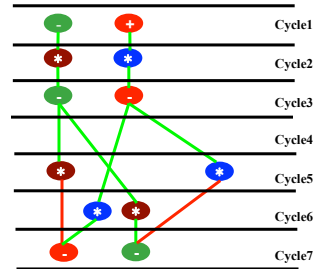
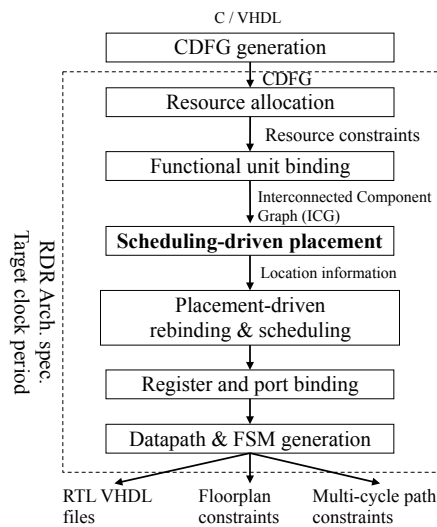
Outline

- ◆ Brief review of history of HLS
- ◆ The UCLA effort
 - Original motivation
 - The Wave of ESL Synthesis
 - Commercialization
- ◆ Highlights of some algorithmic innovations
- ◆ Applications
 - Used in design of accelerator-centric architectures

Original Motivation: Overcome Interconnect Bottleneck in Nanometer Designs

- ◆ Interconnect delay dominates the performance
- ◆ But not all long wires are equally bad
 - ◆ Those to be crossed in a single clock cycle are worst
- ◆ A physical synthesis idea –
 - ◆ Can we allocate multiple clock cycles to all long wires?
- ◆ Our study:
 - ◆ Does not work if we start from RTL
 - ◆ Only feasible if we start from a behavior description
 - ◆ Need to total freedom to schedule communication and communication

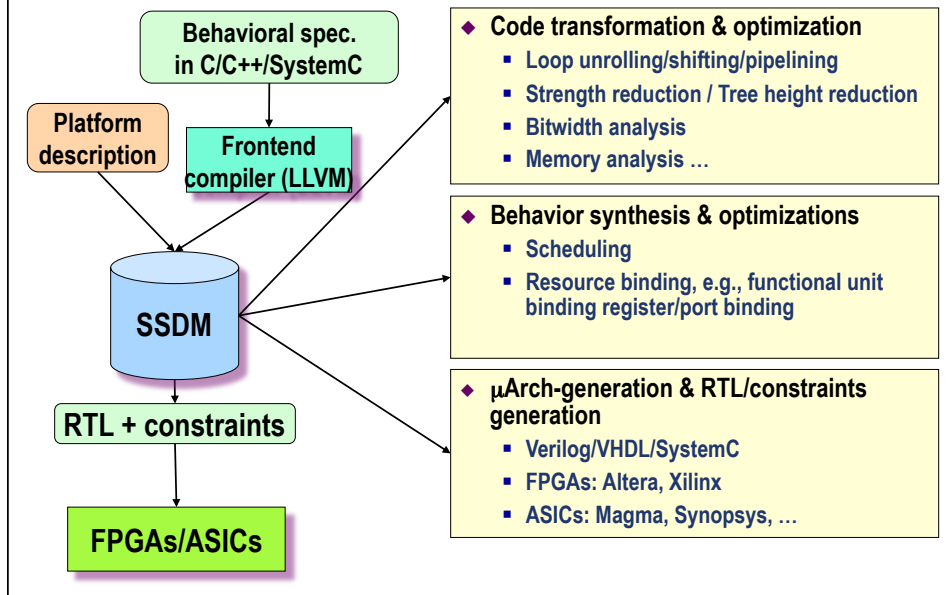
MCAS: Architectural Synthesis for Multi-cycle Communication [TCAD 2004]



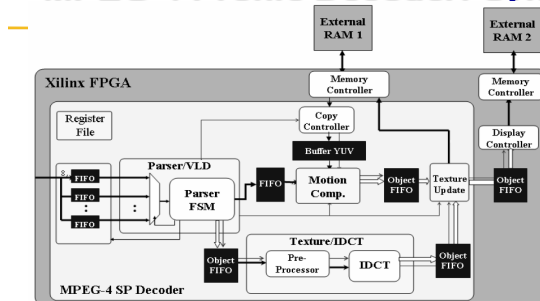
The Wave of ESL (Electronic-System Level) Synthesis

- ◆ Embedded processors are in almost every SoC
 - Need SW/HW co-design and exploration
 - C/C++/SystemC is a more natural starting point
- ◆ Huge silicon capacity requires high-level of abstraction
 - 700,000 lines of RTL for a 10M gate design is too much!
- ◆ Verification drives the acceptance of SystemC
 - Need executable model to verify against RTL design
 - More and more SystemC models are available
- ◆ Need and opportunity for aggressive power optimization
- ◆ Accelerated computing or reconfigurable computing also need C/C++ based compilation/synthesis to FPGAs

xPilot: Behavioral-to-RTL Synthesis Flow [SOCC'2006]



MPEG-4 Profile Decoder: Synthesis Results

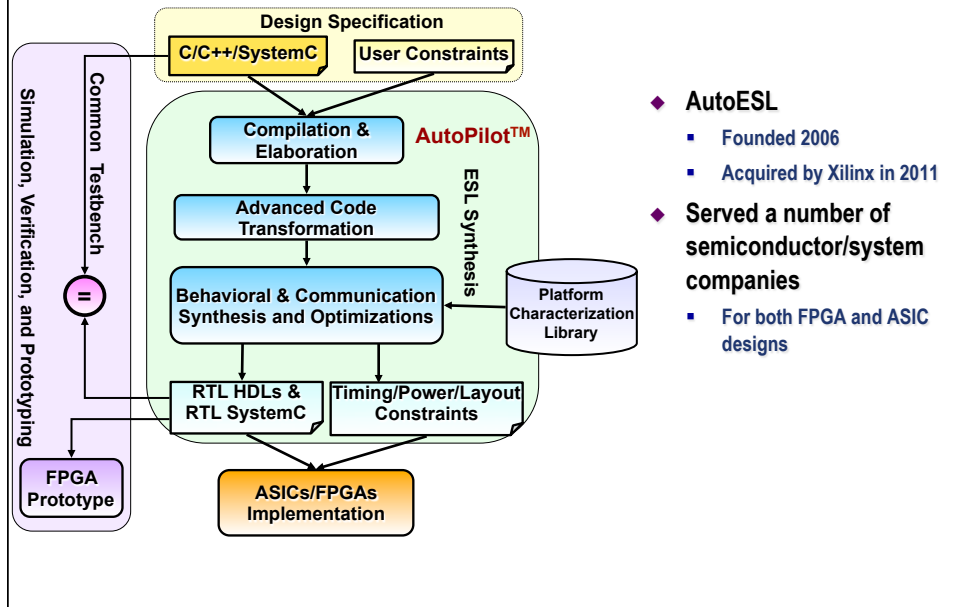


Setting	Module	Slices	BRAMs	Period (ns)
Video: CIF @ 30fps Device: v2 p30	Parser/VLD	2693	16	8.0
	Motion Comp.	899		
	Texture/IDCT	2032		
	Texture Update & Copy Control	1407		

• Complexity of synthesized RTLs

Module Name	Orig. C Source File (+ line#)	VHDL line#
Copy Controller	copyControl.c (287)	2815
Motion Comp.	Motion-compensation.c (312)	4681
Parser/VLD	bitstream.c (439)	6093
	motion_decode.c (492)	10934
	parser.c (1095)	12036
	texture_vld.c (504)	6089
Texture/IDCT	texture_idct.c (1819)	11537
Texture Update	textureUpdate.c (220)	2736
Total	5168	56921

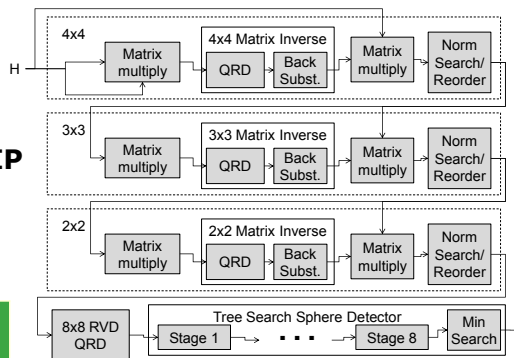
Commercialization: xPilot was Licensed to AutoESL



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”

AutoPilot Results: Optical Flow (from BDTI)

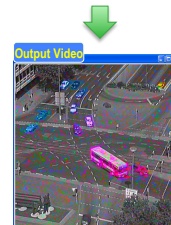
◆ Application

- Optical flow, 1280x720 progress scan
- Design too complex for an RTL team

◆ Compared to high-end DSP:

- 30X higher throughput, 40X better cost/fps

	Chip Unit Cost	Highest Frame Rate @ 720p (fps)	Cost/performance (\$/frame/second)
Xilinx Spartan3ADSP XC3SD3400A chip	\$27	183	\$0.14
Texas Instruments TMS320DM6437 DSP processor	\$21	5.1	\$4.20



BDTi evaluation of AutoPilot

<http://www.bdti.com/articles/AutoPilot.pdf>

AutoPilot Results: DQPSK Receiver (from BDTI)

◆ Application

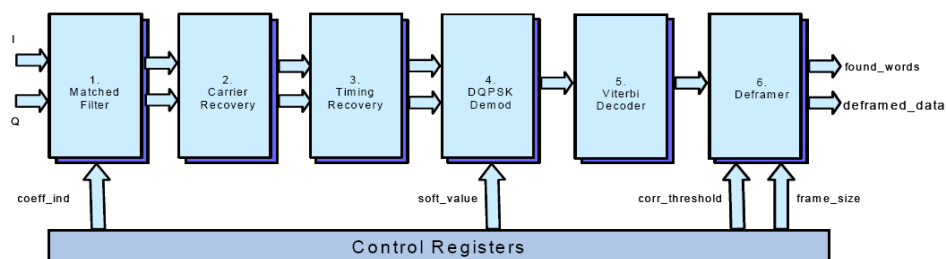
- DQPSK receiver
- 18.75Msamples @75MHz clock speed

◆ Area better than hand-coded

	Hand-coded RTL	AutoPilot
Xilinx XC3SD3400A chip utilization ratio (lower the better)	5.9%	5.6%

BDTi evaluation of AutoPilot

<http://www.bdti.com/articles/AutoPilot.pdf>



What Made xPilot/AutoPilot Successful

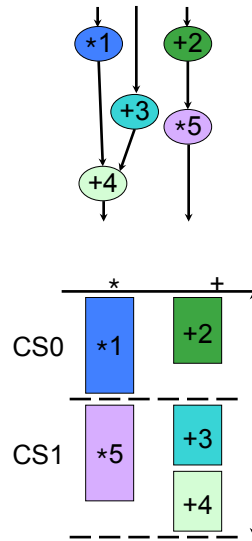
- ◆ **Use of LLVM compilation infrastructure**
 - A good decision made in 2004
- ◆ **Platform-based synthesis**
 - RTLs are optimized for different implementation platforms
 - Cell and interconnect delays, memory configurations, I/O ports/types
 - ...
- ◆ **Most importantly, algorithmic innovations**
 - Global optimization under multiple constraints, objectives
 - Be competitive to manual RTL designs

Outline

- ◆ **Brief review of history of HLS**
- ◆ **The UCLA effort**
 - Original motivation
 - The Wave of ESL Synthesis
 - Commercialization
- ◆ **Highlights of some algorithmic innovations**
- ◆ **Applications**
 - Used in design of accelerator-centric architectures

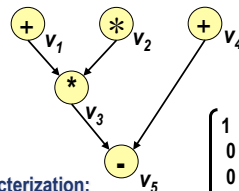
Example: Versatile Scheduling Algorithm Based on SDC (DAC'06)

- ◆ Scheduling problem in behavioral synthesis is NP-Complete under general design constraints
- ◆ ILP-based solutions are versatile but very inefficient
- ◆ Hueristics are often used (e.g. list scheduling)
- ◆ Our solution: An efficient and versatile scheduler based on SDC (system of difference constraints)
 - Applicable to a broad spectrum of applications
 - Computation/Data-intensive, control-intensive, memory-intensive, partially timed.
 - Salable to large-size designs (finishes in a few seconds)
 - Amenable to a rich set of scheduling constraints:
 - Resource constraints, latency constraints, frequency constraints, relative IO timing constraints.
 - Capable of a variety of synthesis optimizations:
 - Operation chaining, pipelining, multi-cycle communication, incremental scheduling, etc.



Scheduling - Our Approach (DAC'06)

- ◆ Overall approach
 - Current objective: high-performance
 - Use a system of integer difference constraints to express all kinds of scheduling constraints
 - Represent the design objective in a linear function



- Platform characterization:
 - adder (+/-): 2ns
 - multiplier (*): 5ns
- Target cycle time: 10ns
- Resource constraint: Only ONE multiplier is available

$$\begin{bmatrix} 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 1 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \leq \begin{bmatrix} 0 \\ -1 \\ 0 \\ 0 \\ -1 \end{bmatrix}$$

A
 x
 b

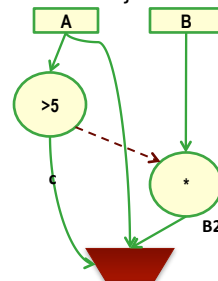
Totally unimodular matrix: guarantees integral solutions

- Dependency constraint
 - $v_1 \rightarrow v_3 : x_3 - x_1 \geq 0$
 - $v_2 \rightarrow v_3 : x_3 - x_2 \geq 0$
 - $v_3 \rightarrow v_5 : x_5 - x_3 \geq 0$
 - $v_4 \rightarrow v_5 : x_5 - x_4 \geq 0$
- Frequency constraint
 - $\langle v_2, v_5 \rangle : x_5 - x_2 \geq 1$
- Resource constraint
 - $\langle v_2, v_3 \rangle : x_3 - x_2 \geq 1$

Extensions to Handle Soft Constraints [ICCAD'09]

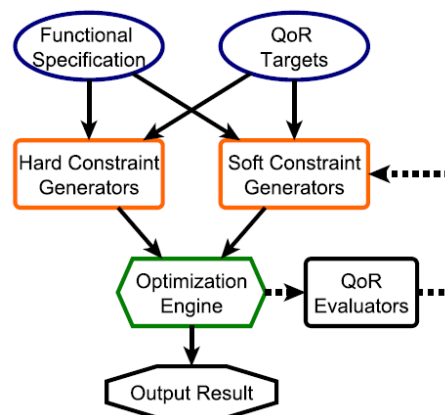
- Some constraints are preferred but not required
- Example: operation gating
 - For power optimization
 - Can be done during scheduling
 - Essentially adding predicate to operations to avoid executing unnecessarily.

```
int module(int A, int B)
{
    int B2 = B * B;
    bool c = A > 5;
    int r = c ? A : B2;
    return r;
}
```



Scheduling with Soft Constraints

- ◆ **Soft constraint**
 - A constraint that is preferably satisfied but not necessarily
 - Penalty associated with violation
- ◆ **Benefits of using soft constraints**
 - Always consistent
 - Easy to specify an intention.
- ◆ **Hard constraints**
 - For dependency, throughput, latency, frequency, etc.



Soft Constraint Solver

$$\begin{array}{ll} \min & c^T s \\ \text{s.t.} & Gs \leq p \quad (\text{hard constraints}) \\ & Hs \leq q \quad (\text{soft constraints}). \end{array}$$

introduce variable v to denote the violation

Add penalty term $\phi(v)$

$$\begin{array}{ll} \min & c^T s + \sum_{j=1}^n \phi_j(v_j) \\ \text{s.t.} & \begin{array}{ll} Gs & \leq p \\ Hs & -v \leq q \\ & -v \leq 0. \end{array} \quad \text{or} \quad \begin{bmatrix} G & O \\ H & -I \\ O & -I \end{bmatrix} \begin{bmatrix} x \\ v \end{bmatrix} \leq \begin{bmatrix} p \\ q \\ 0 \end{bmatrix} \end{array}$$

Theorem: $\begin{bmatrix} G & O \\ H & -I \\ O & -I \end{bmatrix}$ is totally unimodular when both hard constraints and soft constraints are of the *integer-difference* form.

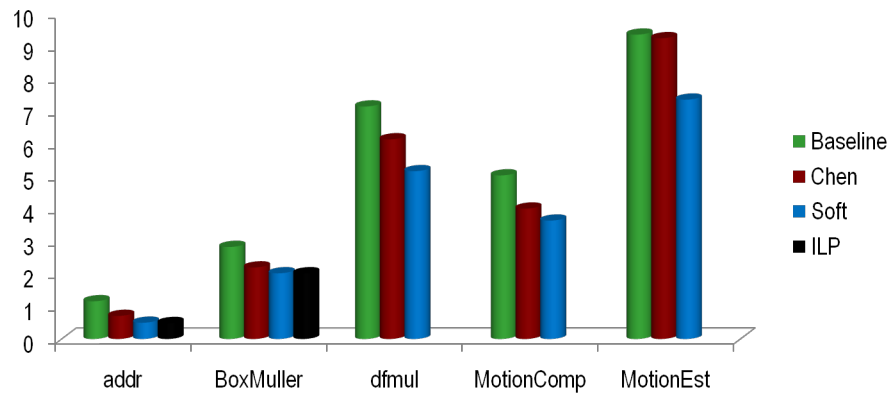
Implication: the problem can be solved optimally in *polynomial* time if the penalty term is convex.

21

Experimental Results

- **Benchmarks with control flow**
 - Arithmetic: BoxMuller, dfmul
 - Multimedia: addr, MotionComp, MotionEst
- **Compare 4 approaches**
 - Scheduler without considering operation gating (Baseline).
 - An iterative algorithm proposed by C. Chen [DATE'02] (Chen).
 - Soft constraints with penalty approximation (Soft).
 - An ILP formulation that handles soft constraints precisely (ILP).

Comparison on Power



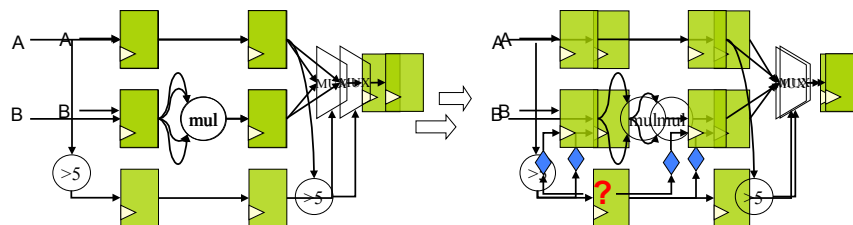
Our approach provides

- 33.9% power reduction compared to baseline on average
- 17.1% power reduction compared to Chen's method on average
- Close result to the ILP method

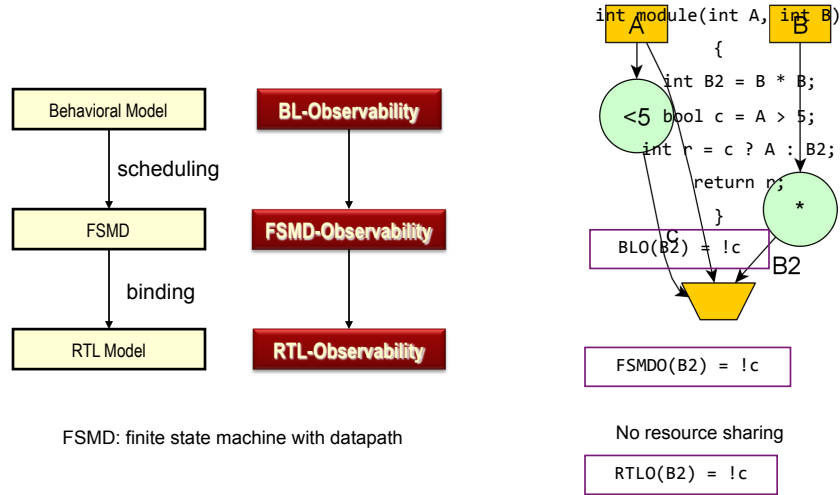
23

Background of Observability Analysis

- ◆ Power management based on observability don't-cares
 - Clock gating, operand isolation, power gating, etc.
- ◆ Opportunity limited when RTL is fixed
 - Need observability analysis and optimization when RTL is created.



Observability at Different Levels



25

Observability Analysis at Behavior Level [ISLPED'09]

◆ For a select operation

- $y = c ? a : b;$
- $BLO(c) = BLO(y)$
- $BLO(a) = BLO(y) \wedge c$
- $BLO(b) = BLO(y) \wedge !c$

◆ For a Boolean and

- $y = a \& b;$
- $BLO(a) = BLO(y) \wedge b$
- $BLO(b) = BLO(y) \wedge a$

◆ Similar for Boolean or

```

int module(int A, int B)
{
    int B2 = B * B;
    bool c = A > 5;
    int r = c ? A : B2;
    return r;
}

```

$BLO(r) = true$
 $BLO(c) = BLO(r) = true$
 $BLO(A) = BLO(r) \wedge c = c$
 $BLO(B2) = BLO(r) \wedge !c = !c$
 $BLO(A) = BLO(A) \vee BLO(c) = true$
 $BLO(B) = BLO(B2) = !c$

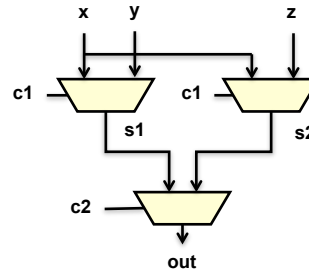
Complexity: $O(|V| + |E|)$ Boolean condition calculations

$|V|$: number of nodes, $|E|$: number of edges (in dataflow graph).

26

Problem: Inaccurate with Correlated Select Inputs

- ◆ The previous way to handle select operation
 - $BLO(c2) = \text{true}$
- ◆ However, we claim that
 - $BLO(c2) = !c1$
 - Because when $c1$ is false, $s1$ and $s2$ both equal to x
- ◆ To consider such opportunities, we merge select operations in the graph into a single one.
 - A (k, l) -select is one that takes k control inputs ($b_1, \dots, b_k$), l data inputs ($d_1, \dots, d_l$) and generates one output z .
 - Conditions absorbed in the select



```
int module(int x, int y, int z,
           bool c1, bool c2)
{
    int s1 = c1 ? x : y;
    int s2 = c1 ? x : z;

    int out = c2 ? s1 : s2;
    return out;
}
```

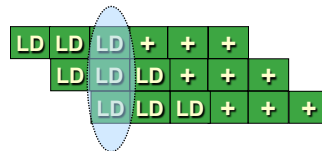
Dataflow Graph Abstraction [TODAES'2010]

- ◆ Observability can be analyzed using previous method in Boolean network analysis
 - However, need to break word-level network into bit-level
- ◆ Desired abstraction
 - Word-level analysis for efficiency
 - Ignore details of complex operations (such as multiplication)
 - Treat them as black boxes
 - The results called *smoothed* observability
 - Keep all information regarding observability-masking operations
 - Including select, Boolean operation for condition evaluation
 - Consider data correlations
 - Condition correlation: Boolean operations
 - Data correlation (for input of select operations)
- ◆ Details in TODAES'2010.

Automatic Memory Partitioning -- Motivation

- ◆ Design input for HLS are often captured by loop nests
- ◆ Most commonly used objective of HLS is throughput optimization
- ◆ Loop pipelining allows a new iteration to begin processing before the previous iteration completes
- ◆ Memory port number is often a limiting factor

```
for (i = 2; i < N; i++)
    sum += A[...] + A[...] + A[...];
```



Pipelining with $II=1$ is infeasible
using a dual-port memory

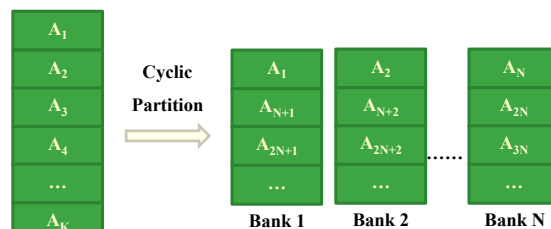
Initiation interval (II) : number of time steps before the next iteration begin processing

Meeting memory port constraint in scheduling is a challenging problem!

29

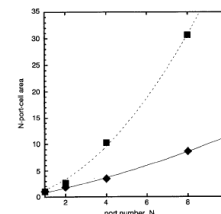
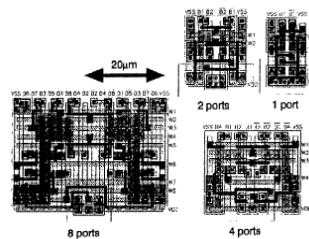
Increase Memory Bandwidth

- ◆ Increase cell ports
- ◆ Data duplication
 - Increase read ports
 - Huge area overhead
- ◆ Memory partitioning



Size = K , Bandwidth = p
 p : memory port number

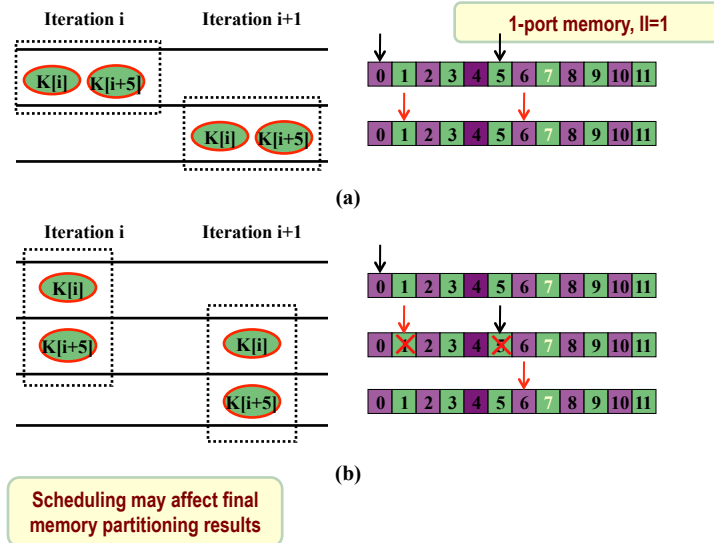
Size $\approx K$, Bandwidth = $N \cdot p$
 N : Partition Factor



Area of SRAM cells
against ports

30

Motivational Example



31

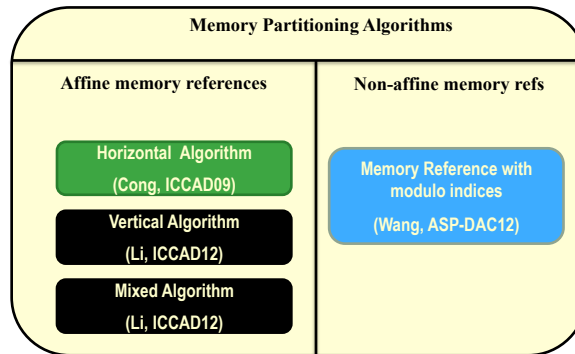
Problem Formulation

- ◆ Given
 - ◆ Computation kernels in loop nests
 - ◆ Array references in each loop nest
 - ◆ Platform information (memory ports limitation)
 - ◆ Throughput constraints (ll)
- ◆ Find memory partitioning P and scheduling for each memory access
 - ◆ Meet throughput constraints ll
 - ◆ At each time step, $\#accesses < \#ports$

32

Memory Partitioning

For given memory inputs, how to generate efficient memory partitioning and scheduling results?



33

Memory and Operation Scheduling

Coupling

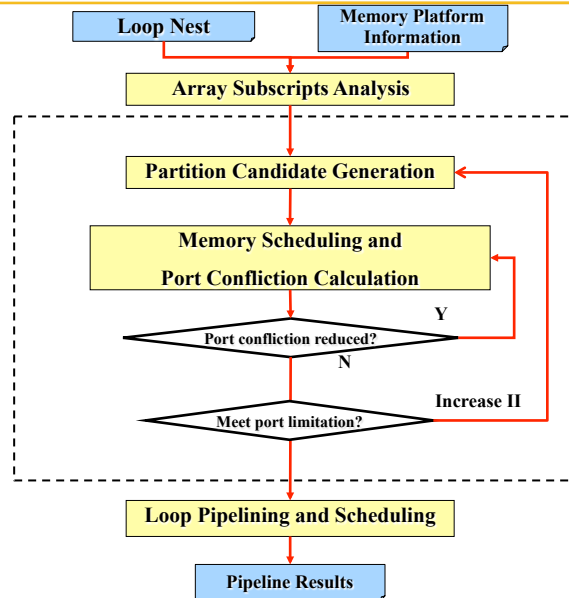
- ◆ Use one set of constraints and objective functions for both memory scheduling and operation scheduling
- ◆ Pros
 - Good QoR: remove some unnecessary temporal registers
- ◆ Cons
 - The source code of operation scheduling maybe unavailable
 - Larger complexity, longer runtime
- ◆ ICCAD 2009, ASP-DAC 2012

Decoupling

- ◆ Decouple memory references from input source code
- ◆ Build a stand-alone memory system with streaming interface to communicate with computation parts
- ◆ Pros
 - Don't need source code of operation scheduling
 - Can be co-operate with (different) commercial HLS tools
- ◆ Cons
 - Compromised QoR for possible unnecessary registers
 - Hard to handle complex dependencies
- ◆ ICCAD 2012

34

Horizontal Algorithm Overview *



* J. Cong, W. Jiang, B. Liu, and Y. Zou, "Automatic Memory Partitioning and Scheduling for Throughput and Power Optimization", ICCAD 2009

35

Candidates Generation

◆ *Cyclic partitioning*

- Easy to implement
- Very effective in practice

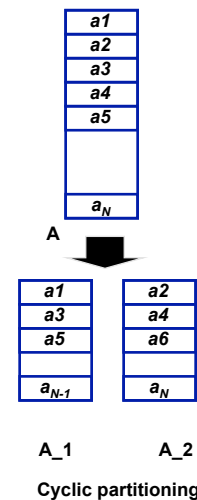
◆ *Theorem*

$$\forall i, a_1 * i + b_1 \neq a_2 * i + b_2 \pmod N$$

$$\Leftrightarrow \gcd(a_1 - a_2, N) \nmid (b_1 - b_2)$$

◆ *Example*

- i and $(3*i+1)$ will always be distributed to different memory banks for 2-way cyclic partitioning



36

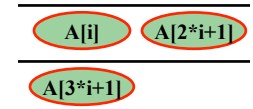
Port Limitation Checking

◆ Conflict graph G

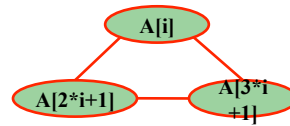
- Node: array references
- Edges: there is an edge $\langle i, j \rangle$ in G if array reference R_i and R_j may access the same memory bank at certain loop iteration

◆ Theorem

- If $\max_clique(G) < port_limit$, then at each time step, number of concurrent memory accesses $< port_limit$



Memory accesses and scheduling



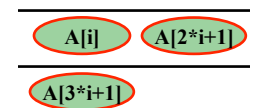
Conflict Graph

37

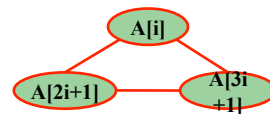
Memory Scheduling

◆ Iterative Algorithm

- Select a partitioning candidate from candidates group
- Construct conflict graph G
- Check $|\max_clique(G)| < port_limit$
 - If yes, return the final partitioning solution
 - Otherwise, reschedule and test again



Memory accesses and scheduling



Conflict Graph

Max_clique = 3

38

Experimental Results (for lithography simulation acceleration)

◆ Platform: Xilinx Virtex-4 FPGA

	II	latency	BRAM	LUT	FF	SLICE	CP(ns)
Original	16	1603	164	1667	584	1220	9.996
Manual	1	113	184	2839	1822	2027	9.939
AMP (1)	1	112	176	2749	1941	2066	9.843
AMP (2)	2	211	168	2018	714	1583	9.984
AMP (4)	4	407	164	1890	727	1435	9.811

- ◆ AMP: Automatic memory portioning (this work)
- ◆ Design exploration with 4X difference in performance and 60% difference in area
- ◆ Manual design with > 1000 lines of code

39

Experimental Results (Throughput)

◆ Platform: Xilinx Virtex-4 FPGA

	Original II	AMP II	Original Slices	AMP Slices	COMP
fir	3	1	241	510	2.12
idct	4	1	354	359	1.01
litho	16	1	1220	2066	1.69
matmul	4	1	211	406	1.92
motionEst	5	1	832	961	1.16
palindrome	2	1	84	65	0.77
avg		5.67x			1.45

- ◆ Average 6x performance improvement with 45% area overhead

40

Limitations

◆ Non-affine memory references

- Data reuse will introduce modulo operations to memory references
- $(a_1 * i + b_1) \% m$
- m: reuse buffer size
- ASP-DAC 2012

Example	
R_1	R_2
$2 * i + 1$	$9 * i + 1$
$2 * i$	$3 * i + 1$
$i + 4$	$4 * i + 1$

◆ What if $(a_2 - a_1) \nmid (b_2 - b_1)$?

- Can't find suitable N
- Solution here, increase $//$
- ICCAD 2012

41

Memory Partitioning for Data Reuse

◆ On-chip reuse buffer: Reduce the repeated off-chip accesses

[Cong, DAC'11, ICCAD'11]

- Temporarily store the fetched data on-chip
- Then repeatedly access them on-chip

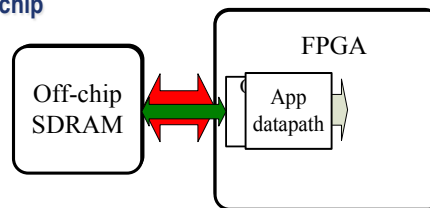
◆ Example [Cong, ICCAD'11]

Data reuse will introduce modulo operations to memory references

```
for i=0 to 199
  for j=0 to 299
    for k=0 to 399
      ...=f(A[i, j, k], A[i-3, j, k],
           A[i, j-2, k], A[i, j, k-1]);
```



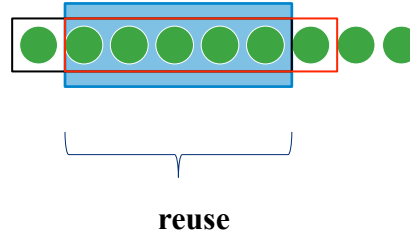
```
data_type buf[2][400];
for i=0 to 199
  for j=0 to 299
    for k=0 to 399 {
      read_in = A[i, j, k];
      ...=f(read_in, A[i-3, j, k],
           buf[j%2][k], A[i, j, k-1]);
      buf[j%2][k] = read_in;
    }
```



An Motivational Example*

H.264 motion compensation

```
//access from off-chip memory
for (j = 0; j < 16; j++)
    for (i = 0; i < 16; i++) {
        pp0 = lumabuffer[j][i];
        pp1 = lumabuffer[j][i+1];
        pp2 = lumabuffer[j][i+2];
        pp3 = lumabuffer[j][i+3];
        pp4 = lumabuffer[j][i+4];
        pp5 = lumabuffer[j][i+5];
        filter(pp0,pp1,pp2,pp3,pp4,pp5,&out,...);
    }
```



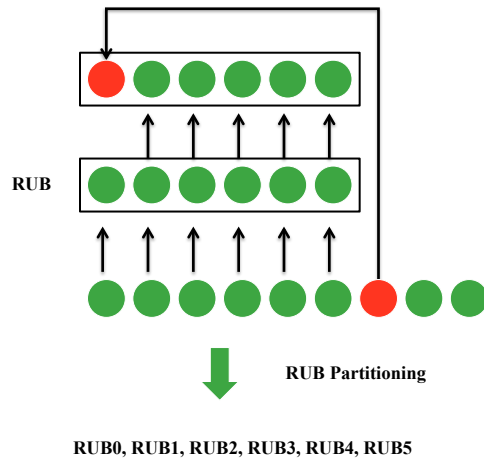
* Y. Wang, P. Zhang, X. Cheng and J. Cong, "An Integrated and Automated Memory Optimization Flow for FPGA Behavioral Synthesis", ASP-DAC2012

An Motivational Example

H.264 motion compensation

```
imgpel RUB[6];
for (j = 0; j < 16; j++)
{ //Reuse buffer pre-fetch
    RUB[0] = lumabuffer[j][0]; RUB[1].....
    for (i = 0; i < 16; i++) {
        pp0 = RUB[i%6];
        pp1 = RUB[(i+1)%6];
        pp2 = RUB[(i+2)%6];
        pp3 = RUB[(i+3)%6];
        pp4 = RUB[(i+4)%6];
        pp5 = lumabuffer[j][i+5];

        filter(pp0,pp1,pp2,pp3,pp4,pp5,&out,...);
        RUB[(i+5)%6] = pp5; } }
```



Limitation of AMP Algorithm in ICCAD'09

◆ Partitioning for affine indices [Cong, ICCAD'09]

- $R_1: a_1 \cdot i + b_1$ and $R_2: a_2 \cdot i + b_2$
- n_1 : number of banks
- Solution1: $\gcd(a_1 - a_2, n_1) \nmid (b_2 - b_1)$

◆ Example for array with modulo indices

- Array RUB, size=99
- Accesses to RUB[i%99] and RUB [(7*i+1)% 99]
- Bank partitioning : 2
- i=14, accesses to 0,14, conflict!

0	1		13	14				97	98
---	---	--	----	----	--	--	--	----	----

Extension to Support Modulo Operations [ASPAC'2012]

◆ Extended approach for indices with modulo operation

- $R_1: (a_1 \cdot i + b_1) \% m$ and $R_2: (a_2 \cdot i + b_2) \% m$
- m : size of the reuse buffer
- n_2 : number of banks
- Solution2: $\gcd(a_1 - a_2, m, n_2) \nmid (b_2 - b_1)$

◆ Example

- Array RUB, size=99
- Accesses to RUB[i%99] and RUB [(7*i+1)% 99]
- $n_2=3$, because $\gcd(6, 99, 3) \nmid 1$
- i=14, accesses to 0,14, **no conflict!**

$$n_1 \leq n_2$$

0	1	2		12	13	14		96	97	98
---	---	---	--	----	----	----	--	----	----	----

0	1		13	14				97	98	99
---	---	--	----	----	--	--	--	----	----	----

Performance improve

BRAM reduction

Area overhead

Frequency influence

Padding:

Vertical and Mixed Memory Partitioning*

◆ Horizontal Scheduling

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

■ What if $(a_2 - a_1) \mid (b_2 - b_1)$?

■ Can't find suitable N , increase II

Example	
R_1	R_2
$2 * i + 1$	$9 * i + 1$
$2 * i$	$3 * i + 1$
$i + 4$	$4 * i + 1$

◆ Vertical Memory Partitioning and Scheduling

• Schedule accesses of the *same* memory reference in *different* loop iterations to non-conflicting memory banks

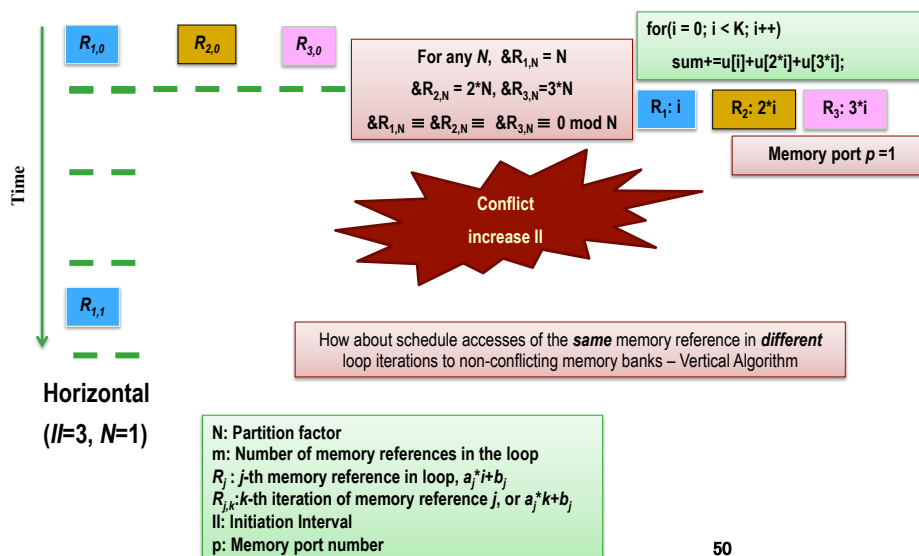
• Can generate valid partitions for any affine memory references

◆ Mixed Memory Partitioning and Scheduling

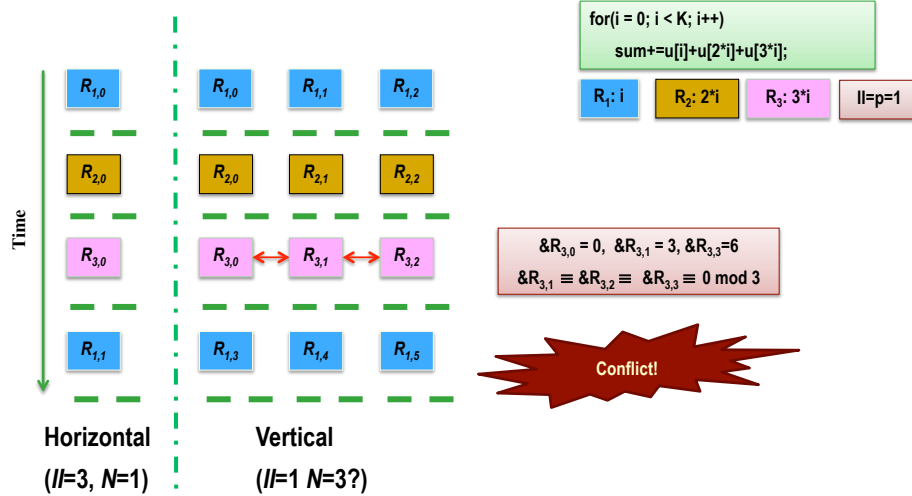
• Relax constraints to allow *different* memory references in *different* loop iterations to be scheduled in one cycle to non-conflicting banks

* P. Li, Y. Wang, P. Zhang, G. Luo, T. Wang and J. Cong, "Memory Partitioning and Scheduling Co-optimization in Behavioral Synthesis", to appear in ICCAD 2012

Motivational Example

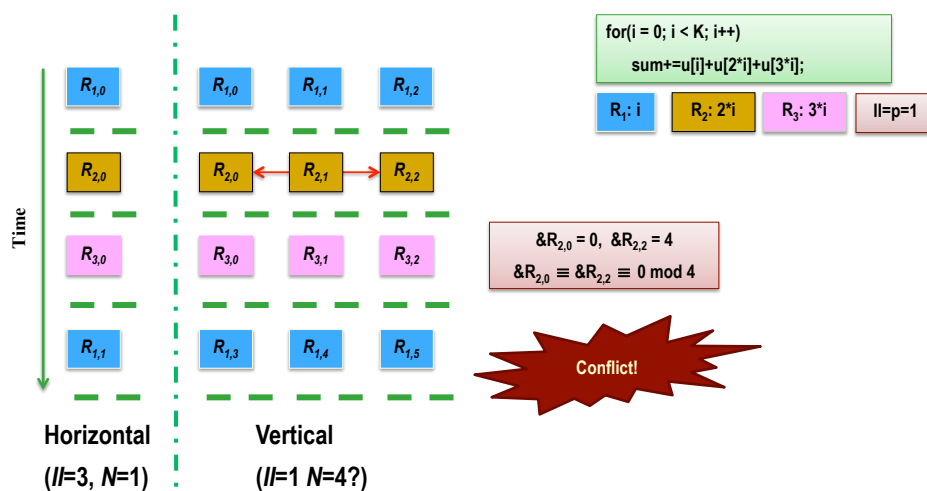


An Motivational Example



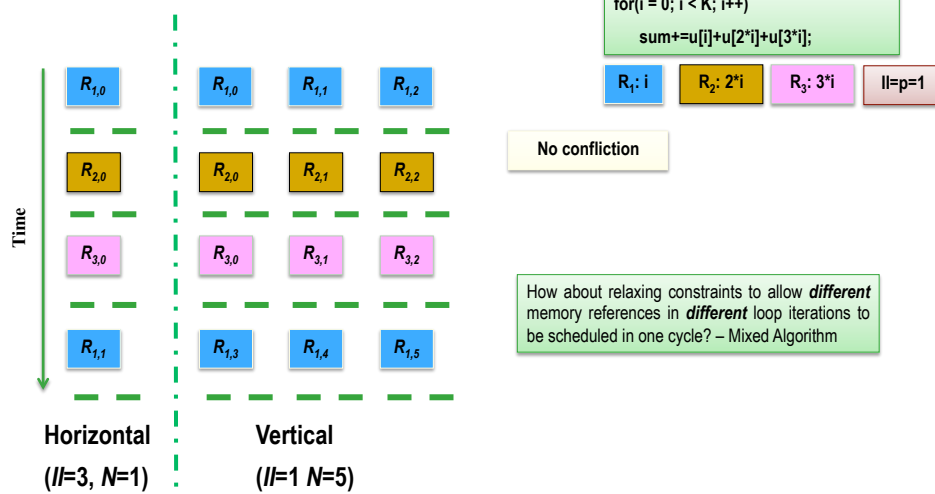
51

Motivational Example



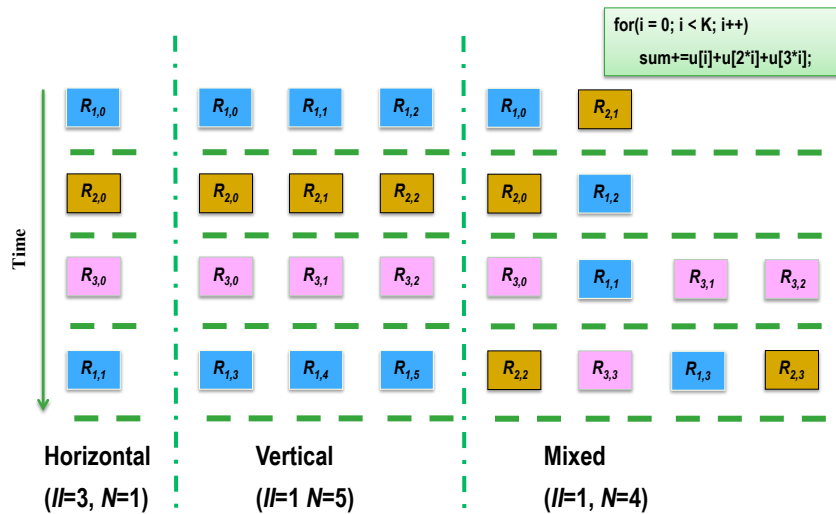
52

Motivational Example



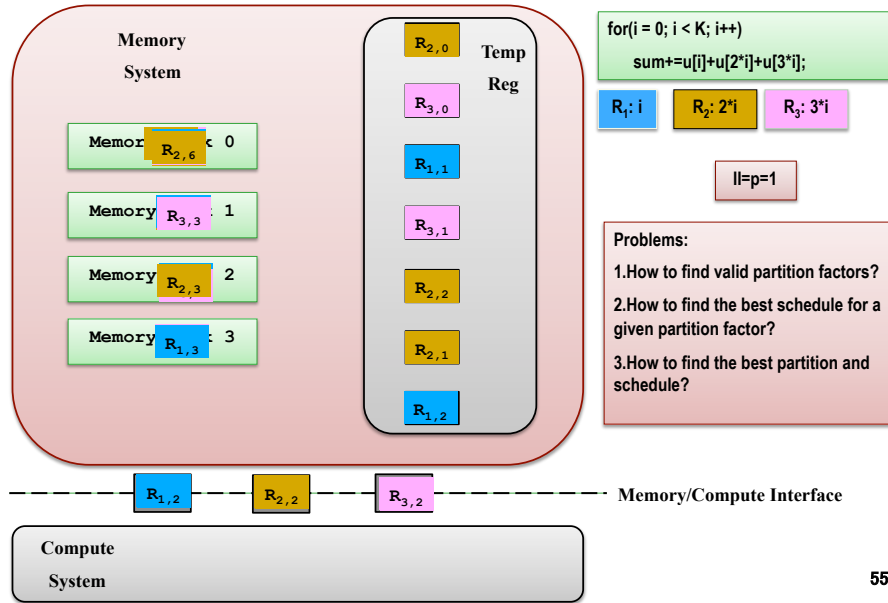
53

Motivational Example



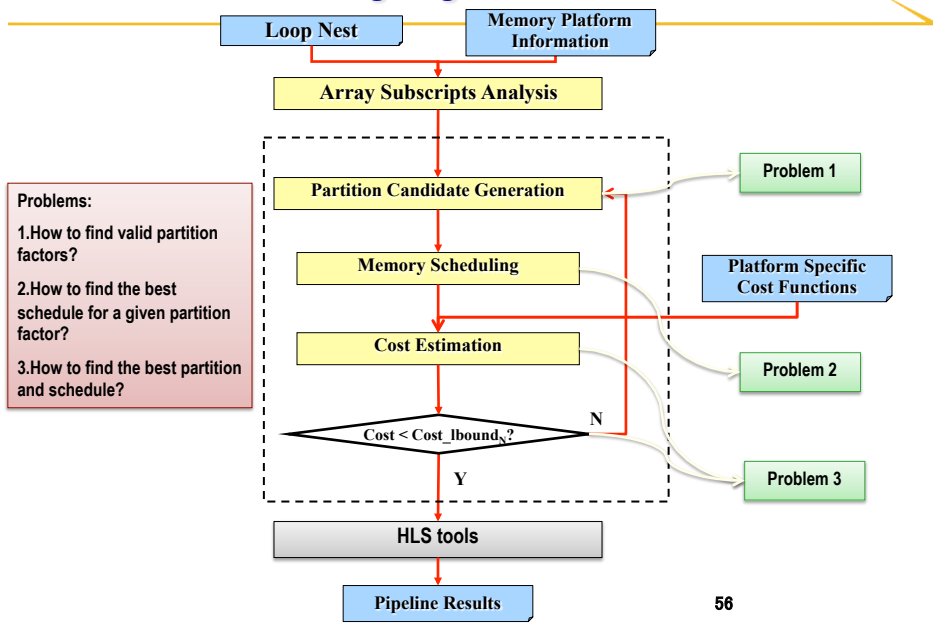
54

Proposed Memory Architecture: Decoupling Memory System and Compute System (Delaying Responses with Required Throughout)



55

Vertical Scheduling Algorithm Overview



56

Vertical Memory Scheduling (Problem 1)

◆ Theorem: N is a valid partition factor

$$\Leftrightarrow \begin{cases} N \geq m * \gcd(N, a_1) / (II * p) \\ N \geq m * \gcd(N, a_2) / (II * p) \\ \dots \\ N \geq m * \gcd(N, a_m) / (II * p) \end{cases}$$

N : Partition factor
 m : Number of memory references in the loop
 a_j : Coefficient of the j -th mem refs $R_j: a_j * i + b_j$
 II : Initiation Interval
 p : Memory port number

◆ Corollary:

- Vertical algorithm can generate valid partitions for any affine memory references
- $m * \max(a_1, a_2, \dots, a_m) / (II * p)$ is a valid partition factor

57

Vertical Scheduling for Motivational Example

$$\begin{cases} N \geq 3 * \gcd(N, 1) & (1) \\ N \geq 3 * \gcd(N, 2) & (2) \\ N \geq 3 * \gcd(N, 3) & (3) \end{cases}$$

```
for(i = 0; i < K; i++)
    sum += u[i] + u[2*i] + u[3*i];
```

$R_1: i$ $R_2: 2*i$ $R_3: 3*i$ $II=p=1$

- ◆ $N=3$ fails on Eq. (3), so 3 is not a valid partition factor
- ◆ $N=4$ fails on Eq. (2), so 4 is not a valid partition factor
- ◆ $N=5$ satisfy all the Eqs, so 5 is a valid partition factor

58

Mixed Memory Scheduling (Problem 1)

◆ Theorem: N is a valid partition factor $\Leftrightarrow$

$$\begin{cases} N \geq (II * p) * (L_{11} * \gcd(a_1, N) + L_{12} * \gcd(a_2, N) + \dots + L_{1m} * \gcd(a_m, N)) \\ N \geq (II * p) * (L_{21} * \gcd(a_1, N) + L_{22} * \gcd(a_2, N) + \dots + L_{2m} * \gcd(a_m, N)) \\ \dots \dots \\ N \geq (II * p) * (L_{N1} * \gcd(a_1, N) + L_{N2} * \gcd(a_2, N) + \dots + L_{Nm} * \gcd(a_m, N)) \end{cases}$$

$$L_{jk} = \begin{cases} 1 & \text{if } \gcd(a_j, N) \mid (b_j - k) \\ 0 & \text{otherwise} \end{cases}$$

N: Partition factor
m: Number of memory references in the loop
 a_j : Coefficient of the j-th mem refs R_j ; $a_j * i + b_j$
 b_j : Constant of the j-th mem refs R_j ; $a_j * i + b_j$
II: Initiation Interval
p: Memory port number

59

Mixed Scheduling for Motivational Example

◆ If $N=3$,

- $L_{11}=L_{12}=L_{13}=L_{21}=L_{22}=L_{23}=L_{33}=1$
- $L_{31}=L_{32}=0$

$$\begin{cases} N \geq \gcd(N, 1) + \gcd(N, 2) & (1) \\ N \geq \gcd(N, 1) + \gcd(N, 2) & (2) \\ N \geq \gcd(N, 1) + \gcd(N, 2) + \gcd(N, 3) & (3) \end{cases}$$

◆ Eq (3) fails, so 3 is not a valid partition factor

◆ If $N=4$,

- $L_{11}=L_{12}=L_{13}=L_{14}=L_{22}=L_{24}=L_{31}=L_{32}=L_{33}=L_{34}=1$
- $L_{21}=L_{23}=0$

$$\begin{cases} N \geq \gcd(N, 1) + \gcd(N, 3) & (1) \\ N \geq \gcd(N, 1) + \gcd(N, 2) + \gcd(N, 3) & (2) \\ N \geq \gcd(N, 1) + \gcd(N, 3) & (3) \\ N \geq \gcd(N, 1) + \gcd(N, 2) + \gcd(N, 3) & (4) \end{cases}$$

◆ All Eqs satisfied, 4 is a valid partition factor

```
for(i = 0; i < K; i++)
    sum+=u[i]+u[2*i]+u[3*i];
```

$R_1: i$

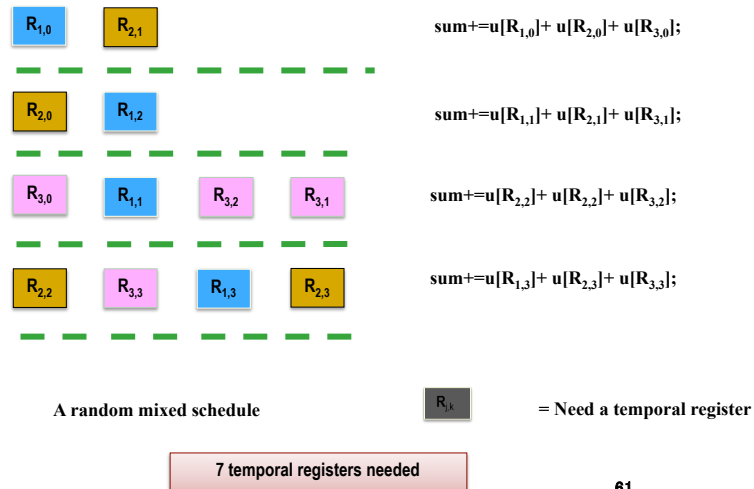
$R_2: 2*i$

$R_3: 3*i$

II=p=1

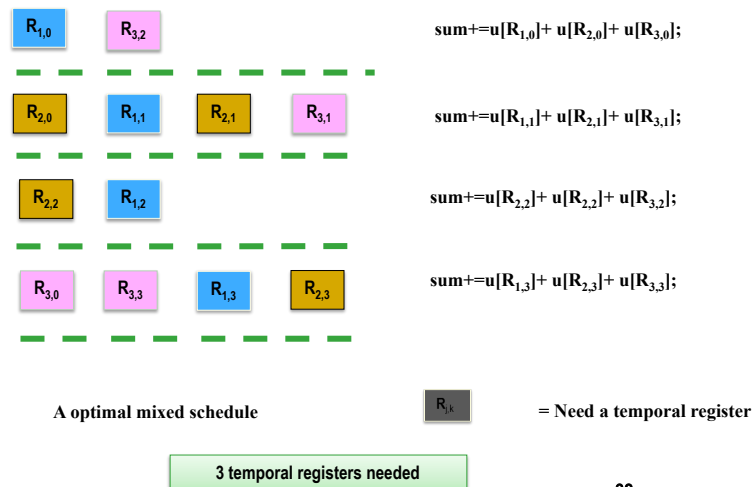
60

Memory Scheduling Algorithm (Problem 2)



61

Memory Scheduling Algorithm (Problem 2)



62

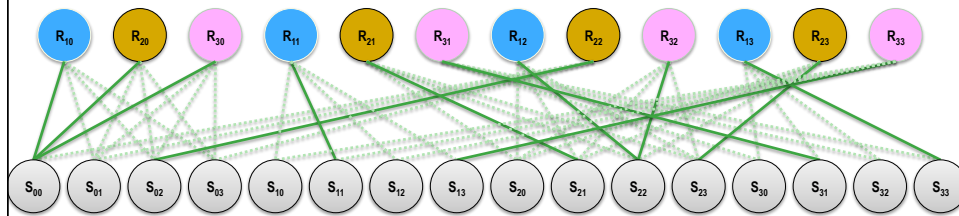
Memory Scheduling Algorithm (Problem 2)

- ◆ Define virtual memory slot $S(l, g, h)$
 - l : memory bank number
 - g : memory port number
 - h : cycle number
- ◆ A schedule is a mapping from R_{jk} to $S(l, g, h)$
- ◆ Cost is the number of temporal registers

$$\text{Cost}(R_{j,k}, S(l, g, h)) = \begin{cases} 0, & \text{if } h = k * II \\ 1, & \text{otherwise} \end{cases}$$
- ◆ Equivalent to finding the max matching with min cost on the bipartite memory scheduling graph
 - Hungarian algorithm in polynomial time

63

Memory Scheduling Graph for the Motivational Example



— Cost = 0, Without tmp register

- - - - - Cost = 1, With tmp register

..... Optimal Schedule

* R_{jk} denotes the k -th iteration of R_j

† S_{lh} denotes the h -th cycle of memory bank l ,
port number is omitted for single port memory

Limitations: Algorithm focus on minimizing temporal registers,
scheduling delay is not optimized.

```
for(i = 0; i < K; i++)
  sum += u[i] + u[2*i] + u[3*i];
```

$R_1: i$

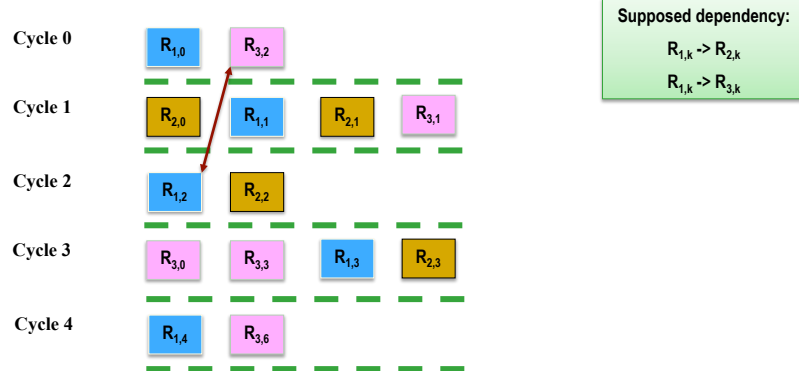
$R_2: 2*i$

$R_3: 3*i$

$II=p=1$

64

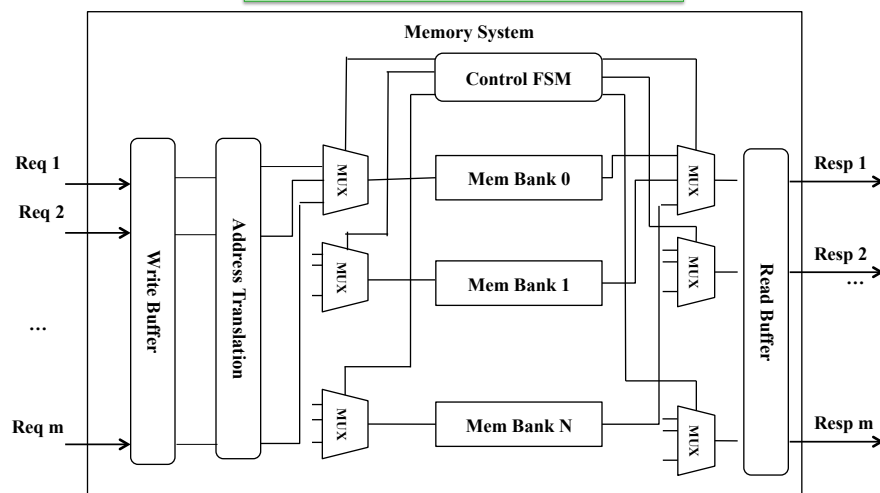
Handling Dependency



65

Cost Estimation (Problem 3)

What is a good memory partition and schedule?
Is partition factor the smaller the better? Not necessarily.



Cost Estimation

◆ Total Cost of memory system

- Memory Banks
- Input Muxes
- Output Mux
- Temporal Registers
- FSM Control
- Address Translation

$$\lambda_1 * N * \left\lceil \frac{\left\lceil \frac{Array_{size}}{N} \right\rceil}{BRAM_{Size}} \right\rceil$$

$$\sum_{k=0}^{N-1} (C_{MUX}(\sum_{j=0}^{m-1} L_{jk}))$$

$$\sum_{j=0}^{m-1} (C_{MUX}(\frac{N}{gcd(a_j, N)}))$$

Calculated by scheduling algorithm

Constant

To be discussed

67

Cost Optimization: Address Translation Optimization (ATO)

◆ Address translation for cyclic partition

- $bank_id = addr \% N$
- $offset = addr / N$
- “%” and “/” are expensive in hardware if $N \neq 2^n$

◆ In loops, addresses are regular with constant stride between adjacent iterations

$$bank_id_{i+1} = \begin{cases} bank_id_i + l, & \text{if } bank_id_i + l < N \\ bank_id_i + l - N, & \text{if } bank_id_i + l \geq N \end{cases}$$

$$offset_{i+1} = \begin{cases} offset_i + k, & \text{if } bank_id_i + l < N \\ offset_i + k + 1, & \text{if } bank_id_i + l \geq N \end{cases}$$

Simple Add/Sub/Comp operations are used to replace expensive div/mods

Experimental Results - Denoise

	Slices	RAMBs	DSPs	Power	CP(ns)
horizontal	529	10	6	679	7.002
horizontal-ATO	459	10	0	537	7.287
horizontal-2^n	3254	256	0	5239	7.335
vertical	1007	7	72	1597	7.259
vertical-ATO	701	7	0	1403	6.505
vertical-2^n	1059	8	0	2477	7.105
mixed	511	7	6	573	6.33
mixed-ATO	427	7	0	510	6.956
mixed-2^n	555	8	0	549	7.046

Xilinx AutoPilot 2011.4, ISE 13.2
Xilinx Virtex-6

In all experiments, $l = p = 1$

69

Experimental Results

		Slices	RAMBs	DSPs	Power	CP(ns)
Denoise	Horizontal	529	10	6	679	6.956
	Mixed-ATO	427	7	0	510	7.002
	Comp(%)	-19.3	-30.0	-100	-24.9	0.7
Registration	Horizontal	486	10	5	358	7.208
	Mixed-ATO	289	6	0	305	5.409
	Comp(%)	-40.5	-40.0	-100	-14.8	-25.0
Binarization	Horizontal	369	10	5	392	7.002
	Mixed-ATO	238	5	0	297	5.293
	Comp(%)	-35.5	-50.0	-100	-24.2	-24.4
Segmentation	Horizontal	671	10	9	891	7.302
	Mixed-ATO	452	7	0	620	6.132
	Comp(%)	-32.6	-30.0	-100	-30.4	-16.0
Deconvolution	Horizontal	1674	10	141	1790	7.4
	Mixed-ATO	556	7	6	581	6.339
	Comp(%)	-66.8	-30.0	-95.7	-67.5	-14.3
Average	Comp(%)	-38.9	-36.0	-99.1	-32.4	-15.8

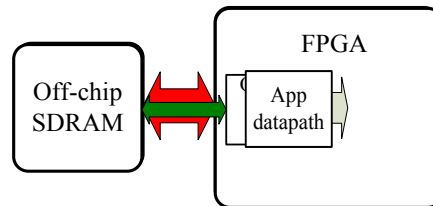
Xilinx AutoPilot 2011.4, ISE 13.2
Xilinx Virtex-6

In all experiments, $l = p = 1$

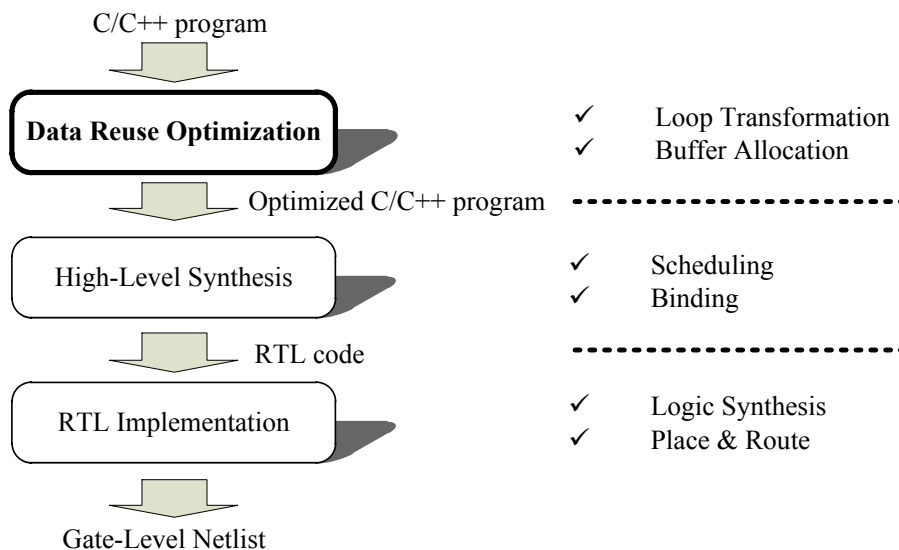
70

Ongoing Work: Code Transformation for Data Reuse [DAC'2012]

- ◆ **Off-chip memory access**
 - Performance: high cycle latency, low bandwidth efficiency
 - Power: 50~80% in data processing applications
- ◆ **On-chip memory: Reduce the repeated off-chip access**
 - Fetch once, access multiple times
 - On-chip resource vs. off-chip bandwidth
- ◆ **Needed in high-level synthesis**
 - Incorporate data reuse by source code transformation



Position in the design flow



Related Work

- ◆ **Memory hierarchy allocation (HA)** [e.g. Issenin'07, Brockmeyer'03]
 - Analyze reuse candidates
 - Select reuse candidates
 - Tradeoff between on-chip buffer size and off-chip bandwidth
- ◆ **Loop transformation (LT)** [e.g. Lim'99, Bondhugula'08]
 - Reduce buffer size by reordering the loop iterations
- ◆ **Combined LT and HA** [e.g. Cong'11]
 - LT needs HA to decide which reuse buffers to optimize
 - HA needs LT to determine and minimize the reuse buffer size
 - But this method limits to perfectly nested loops

Example of Code Transformations

```
// original
for i=0 to N
  for j=0 to N
    S1: B[i,j]=f0(A[i,j], A[i-1,j], A[i-2,j],
               A[i-3,j]);
  for i=0 to N
    for j=0 to N
      S2: C[i,j]=f1(B[i,j],B[i,j-1],B[i,j-2],
                   B[i,j-3]);
```

```
// T0 transformation
for i=0 to N
  for j=0 to N {
    S1: B[i,j]=f0(A[i,j], A[i-1,j], A[i-2,j],
               A[i-3,j]);
    S2: C[i,j]=f1(B[i,j],B[i,j-1],B[i,j-2],
                   B[i,j-3]);
```

data type buff2, N1:			
# offchip reference	buffer size		
	org	T0	T1
2	$N^2+O(N)$	$3N+O(1)$	$N^2+O(N)$
4	$3N+O(1)$	$N+O(1)$	6

```
for j=0 to N
  S2: C[i,j]=f1(B[i,j],B[i,j-1],B[i,j-2],
               B[i,j-3]);
```

```
// T1 transformation
for j=0 to N
  for i=0 to N
    S1: B[i,j]=f0(A[i,j], A[i-1,j], A[i-2,j],
               A[i-3,j]);
  for i=0 to N
    for j=0 to N
```

Optimal loop transformation is different for different off-chip bandwidth requirements!

Problem Modeling and Formulation

Loop Transformation Encoding

```
// original
for i=0 to N
  for j=0 to N
    S1;
  for i=0 to N
    for j=0 to N
      S2;
```

```
// T0
for i=0 to N
  for j=0 to N {
    S1;
    S2;
  }
```

```
// T1
for j=0 to N
  for i=0 to N
```

Loop fusion

$$\phi^{S1} = (0, i, 0, j, 0)^T$$

$$\phi^{S2} = (1, i, 0, j, 0)^T$$

Loop permutation

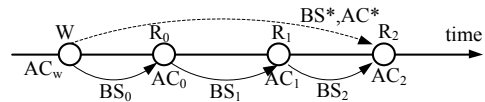
A 2d+1 vector for each statement

Data Reuse

AC: off-chip access count

BS: on-chip buffer size

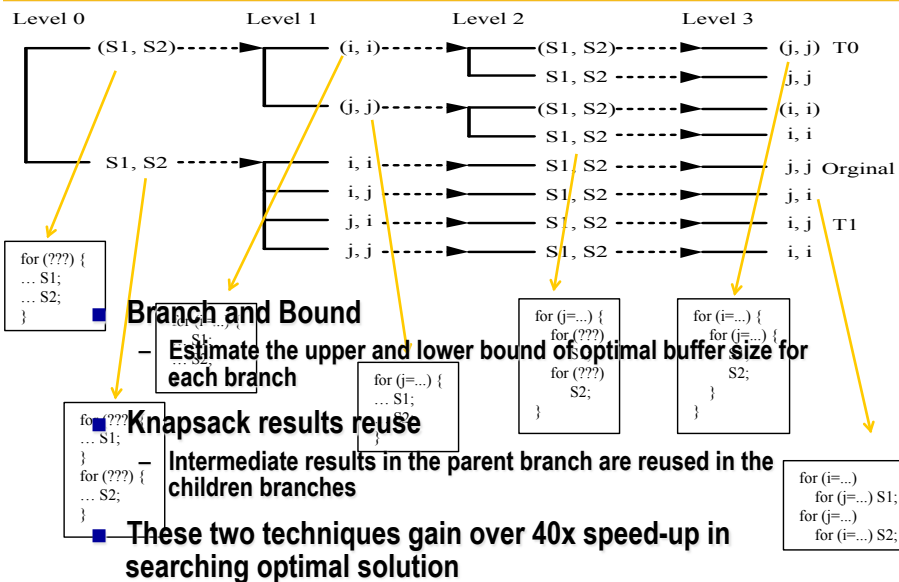
– Calculated by polyhedral model



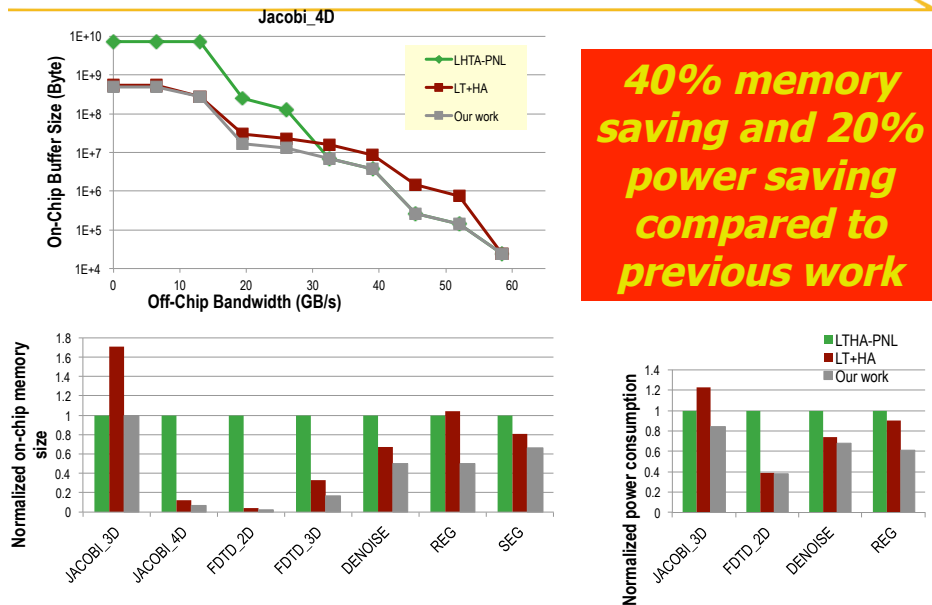
$$BS = \sum_y b_y BS_y, \quad AC = AC_{total} - \sum_y b_y AC_y$$

Knapsack problem

Efficient Design Space Exploration



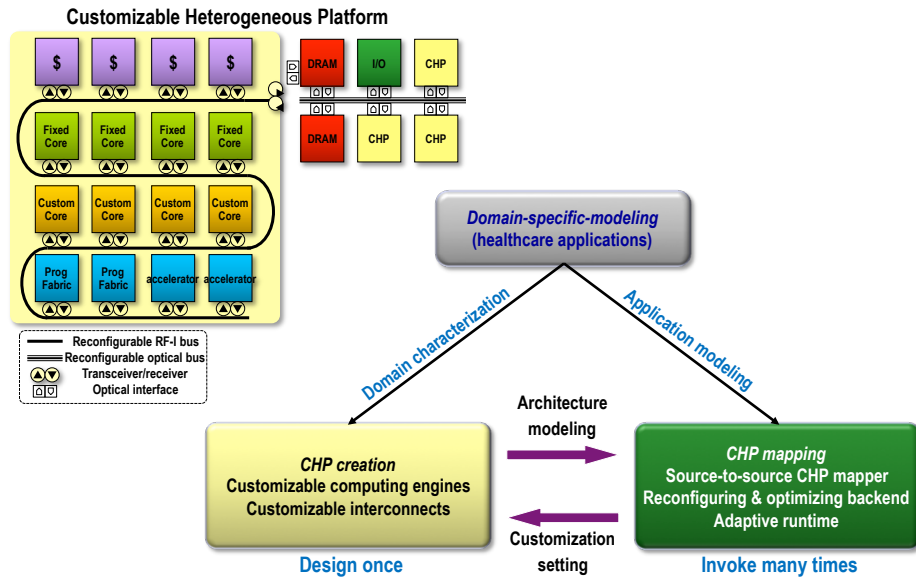
Experiment



Outline

- ◆ Brief review of history of HLS
- ◆ The UCLA effort
 - Original motivation
 - The Wave of ESL Synthesis
 - Commercialization
- ◆ Highlights of some algorithmic innovations
- ◆ Applications
 - Used in design of accelerator-centric architectures

Research Scope in CDSC (Center for Domain-Specific Computing)



Center for Domain-Specific Computing (CDSC)



Aberle
(UCLA)



Baraniuk
(Rice)



Bui
(UCLA)



Chang
(UCLA)



Cheng
(UCSB)



Cong (Director)
(UCLA)



Palsberg
(UCLA)



Potkonjak
(UCLA)



Reinman
(UCLA)



Sadayappan
(Ohio-State)



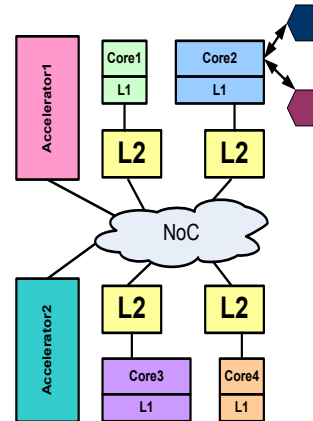
Sarkar
(Associate Dir)
(Rice)



Vese
(UCLA)

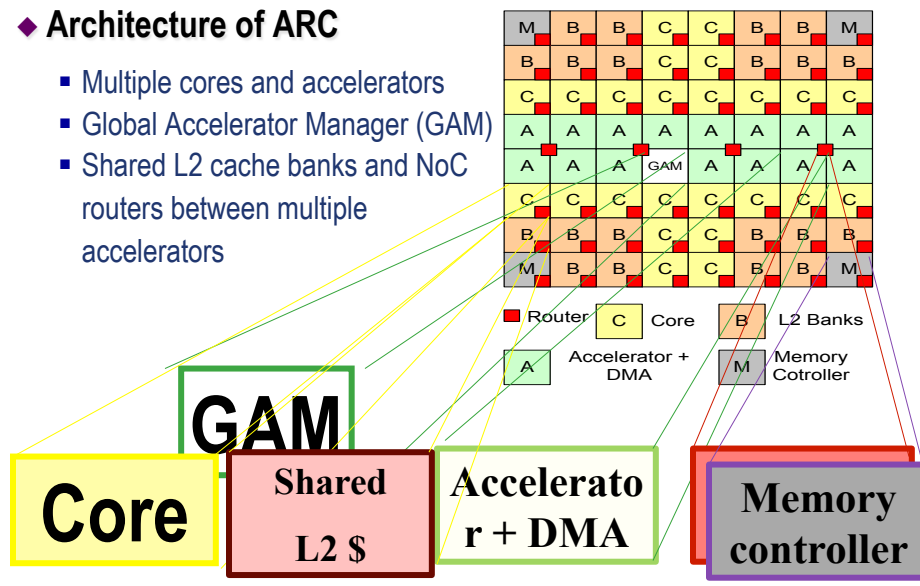
Accelerator-Centric CMP (ARC) [SAW'2011 & DAC'2012]

- ◆ Dark silicon and utilization wall
 - 6.5% utilization for a 45 nm chip filled with 64bit operators, assuming a power budget of 80 W [ASPLOS'2010]
- ◆ Our proposed solution: extensive use of accelerators (customized or implemented using programmable fabric)
 - Sea of accelerators
- ◆ Type of accelerators:
 - Tightly vs. loosely coupled
- ◆ Benefits
 - Better performance
 - Higher power-efficiency
 - It's ok to be "wasteful"
- ◆ Critical needs:
 - Efficient accelerator management
 - Scheduling
 - Sharing

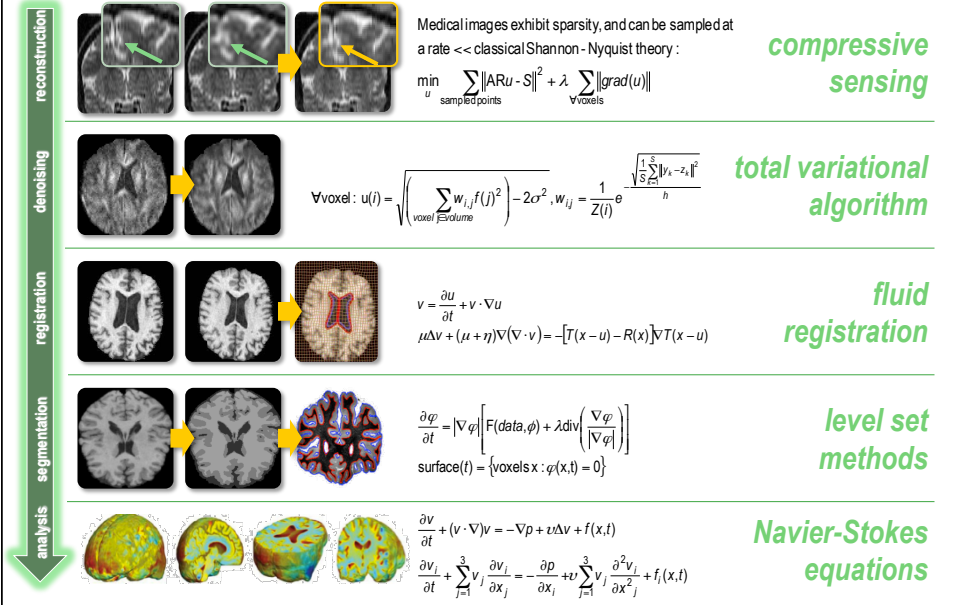


Overall Architecture of ARC

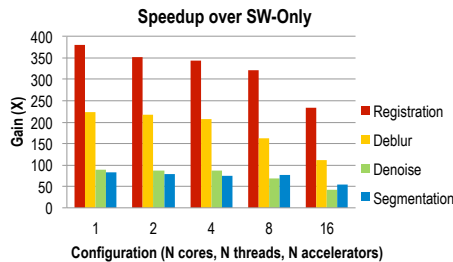
- ◆ Architecture of ARC
 - Multiple cores and accelerators
 - Global Accelerator Manager (GAM)
 - Shared L2 cache banks and NoC routers between multiple accelerators



Application Domain: Medical Image Processing

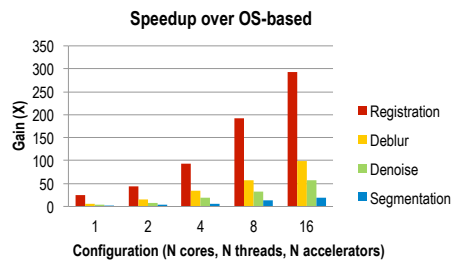


Experimental Results – Performance (N cores, N threads, N accelerators)

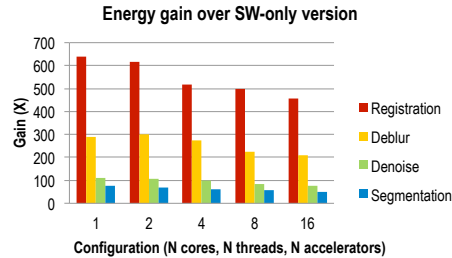


Performance improvement over SW only approaches:
on average 168x, up to 380x

Performance improvement over OS based approaches:
on average 51x, up to 292x

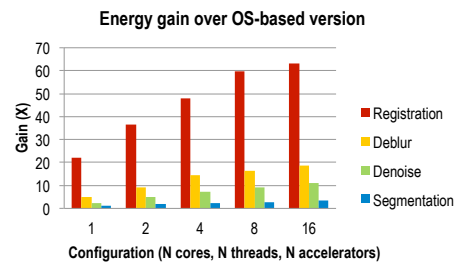


Experimental Results – Energy (N cores, N threads, N accelerators)



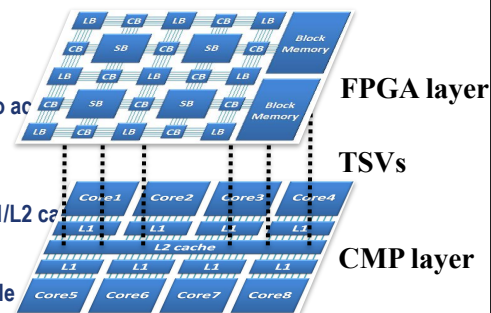
Energy improvement
over SW-only approaches:
on average 241x, up to 641x

Energy improvement
over OS-based approaches:
on average 17x, up to 63x



3D Integration for Customization or Specialization

- ◆ Vertical integration:
 - CMP layer + customization/acceleration layer
- ◆ Accelerators can directly access caches
 - L1 or L2
- ◆ Low latency
 - 1 cycle traversal across the TSV bundle
 - 2-3 cycles to get from the TSV bundle to accelerator
- ◆ Higher bandwidth
 - Almost equal to the bandwidth of the L1/L2 cache
- ◆ No single bottleneck
 - Each cache can have its own TSV bundle
 - Sharing TSV bundles possible
- ◆ Early results: medical imaging benchmarks [ASAP'2011]
 - > 7x performance gain
 - > 18x energy gain



Concluding Remarks

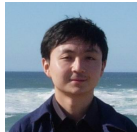
- ◆ Behavior synthesis is maturing
 - Still plenty of room for new research
 - Source-code quality is important
 - System-level integration
- ◆ Customization and specialization open many opportunities for computer architecture innovation
 - Cores
 - Accelerators
 - Memory
 - Network-on-chips
- ◆ Software support is also critical

Acknowledgements

- ◆ Publications are available from
<http://cadlab.cs.ucla.edu/~cong>
- ◆ We would like to thank the supports from
 - Gigascale Systems Research Center (GSRC)
 - National Science Foundation (NSF)
 - Semiconductor Research Corporation (SRC)
 - Various industrial sponsors (Altera, Intel, Magma/Synopsys, Mentor Graphics, Xilinx)
 - Peking University (PKU) Center for Energy-Efficient Computing and Applications
- ◆ Former and current team members from UCLA and PKU who contributed to the HLS project



Yiping Fan



Guoling Han



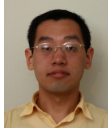
Wei Jiang



Zhiru Zhang



Peng Zhang



Bin Liu



Peng Li (PKU)



Yuxin Wang (PKU)