

## Energy Efficient Computing with Composable Accelerators

Jason Cong

Director, Center for Domain-Specific Computing  
[www.cdsc.ucla.edu](http://www.cdsc.ucla.edu)

Chancellor's Professor, UCLA Computer Science Department  
[cong@cs.ucla.edu](mailto:cong@cs.ucla.edu)

1

## Center for Domain-Specific Computing (CDSC) Funded by 2009 NSF Expeditions in Computing Program



Aberle  
(UCLA)



Baraniuk  
(Rice)



Bui  
(UCLA)



Chang  
(UCLA)



Cheng  
(UCSB)



Cong (Director)  
(UCLA)



Palsberg



Potkonjak



Reinman



Sadayappan



Sarkar



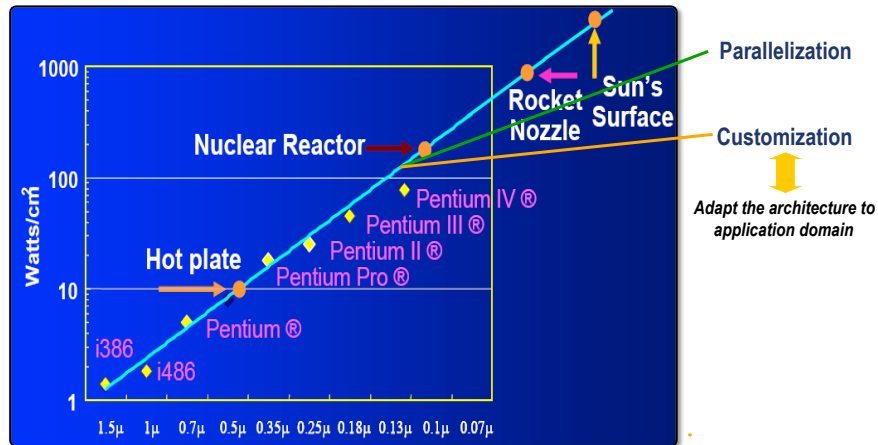
Vese

	UCLA	Rice	UCSB	Ohio State
Domain-specific modeling	Bui, Reinman, Potkonjak	Sarkar, Baraniuk		Sadayappan
CHP creation	Chang, Cong, Reinman		Cheng	
CHP mapping	Cong, Palsberg, Potkonjak	Sarkar	Cheng	Sadayappan
Application drivers	Aberle, Bui, Chien, Vese	Baraniuk		
Experimental systems	All (led by Cong & Bui)	All	All	All

2

## CDSC Focus: New Transformative Approach to Power/ Energy Efficient Computing

- ◆ Current solution: *Parallelization*
- ◆ Next significant opportunity – *Customization*

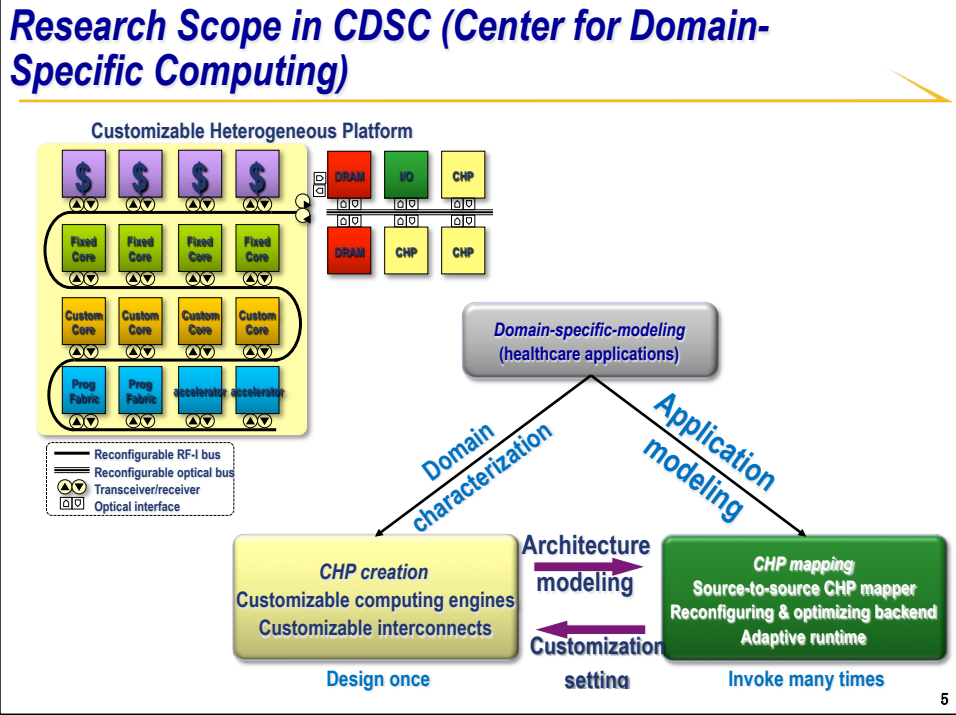


3

## Project Goals

- ◆ A general, customizable platform for the given domain(s)
  - Can be customized to a wide-range of applications in the domain
  - Can be massively produced with cost efficiency
  - Can be programmed efficiently with novel compilation and runtime systems
- ◆ Metric of success
  - A “supercomputer-in-a-box” with +100x performance/power improvement via customization for the intended domain(s)

4



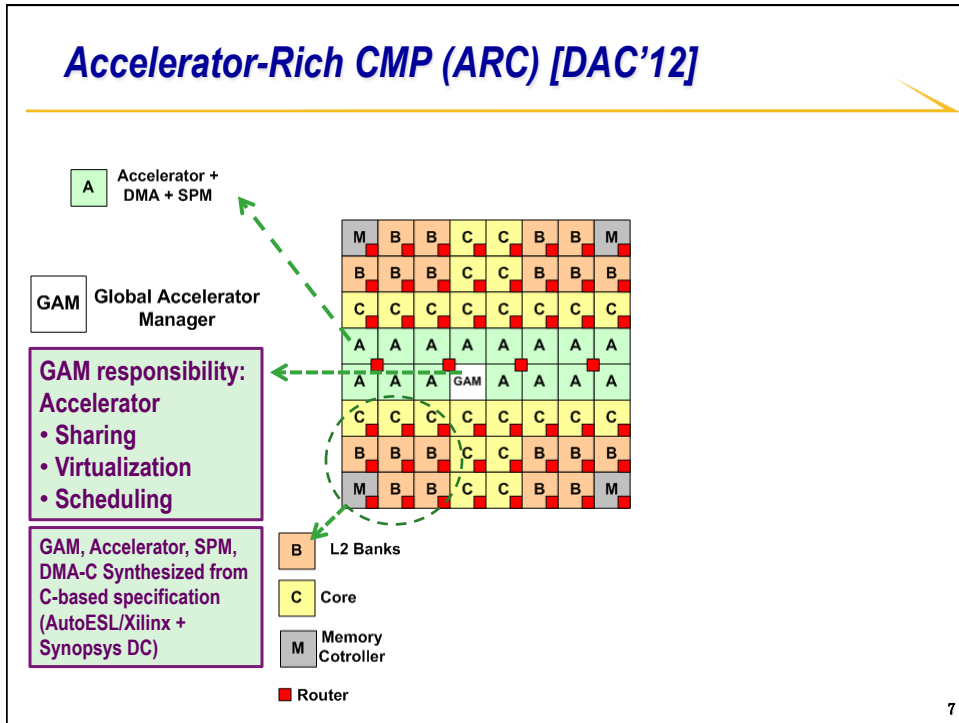
### Performance/Energy Efficiency of Accelerators

AES 128bit key 128bit data	Throughput	Power	Figure of Merit (Gb/s/W)
0.18mm CMOS	3.84 Gbits/sec	350 mW	11 (1/1)
FPGA [1]	1.32 Gbit/sec	490 mW	2.7 (1/4)
ASM StrongARM [2]	31 Mbit/sec	240 mW	0.13 (1/85)
ASM Pentium III [3]	648 Mbits/sec	41.4 W	0.015 (1/300)
C Emb. Sparc [4]	133 Kbits/sec	120 mW	0.0011 (1/10,000)
Java [5] Emb. Sparc	450 bits/sec	120 mW	0.0000037 (1/3,000,000)

[1] Amphion CS5230 on Virtex2 + Xilinx Virtex2 Power Estimator  
 [2] Dag Arne Osvik: 544 cycles AES – ECB on StrongArm SA-1110  
 [3] Helger Lipmaa PIII assembly handcoded + Intel Pentium III (1.13 GHz) Datasheet  
 [4] gcc, 1 mW/MHz @ 120 Mhz Sparc – assumes 0.25 u CMOS  
 [5] Java on KVM (Sun J2ME, non-JIT) on 1 mW/MHz @ 120 MHz Sparc – assumes 0.25 u CMOS

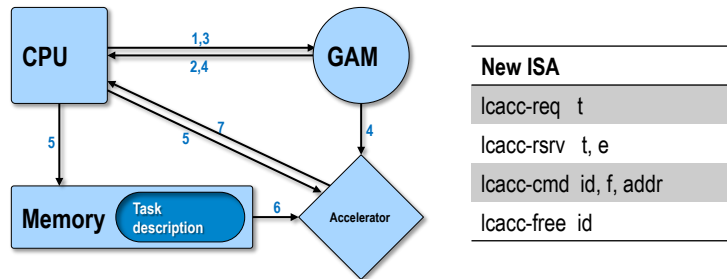
Source: P Schaumont and I Verbauwhede, "Domain specific codesign for embedded security," IEEE Computer 36(4), 2003

## Accelerator-Rich CMP (ARC) [DAC'12]



7

## Overall Communication Scheme in AXR-CMP



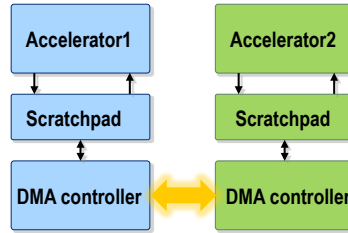
1. The core requests for a given type of accelerator (lcacc-req).
2. The GAM responds with a "list + waiting time" or NACK
3. The core reserves (lcacc-rsv) and waits.
4. The GAM ACK the reservation and send the core ID to accelerator
5. The core shares a task description with the accelerator through memory and starts it (lcacc-cmd).
6. The accelerator reads the task description, and begins working
7. When the accelerator finishes its current task it notifies the core.
8. The core then sends a message to the GAM freeing the accelerator (lcacc-free).

8

## Accelerator Virtualization

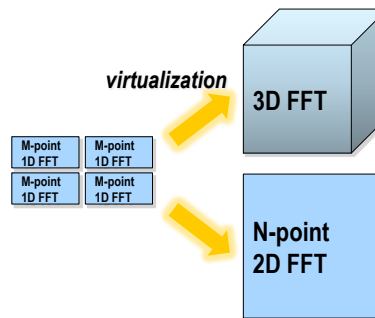
◆ **Chaining**

- Efficient accelerator to accelerator communication



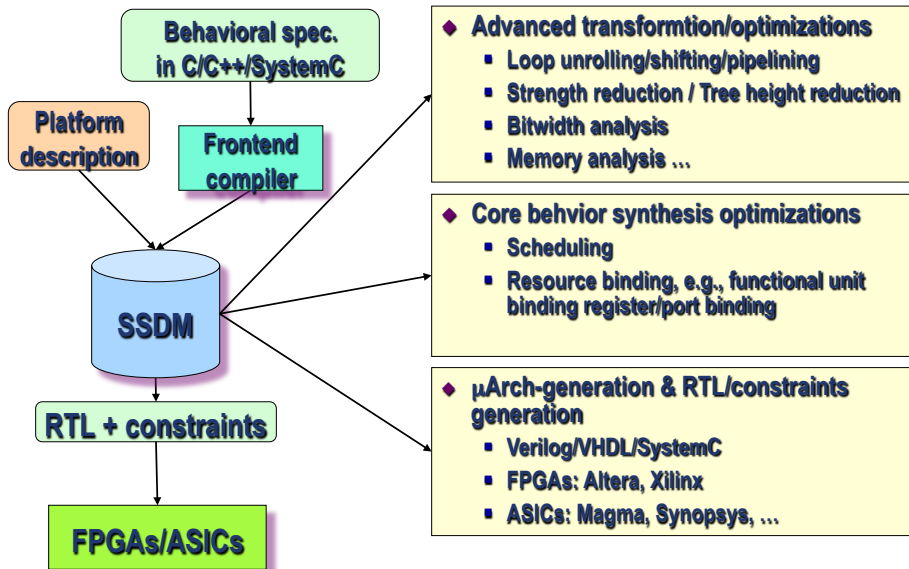
◆ **Composition**

- Constructing virtual accelerators



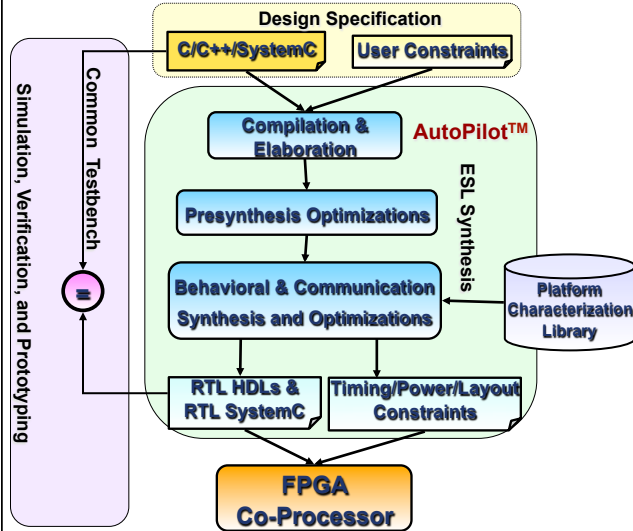
9

## xPilot: Behavioral-to-RTL Synthesis Flow [SOCC'2006] (with GSRC and NSF supports from 2001 – 2006)



10

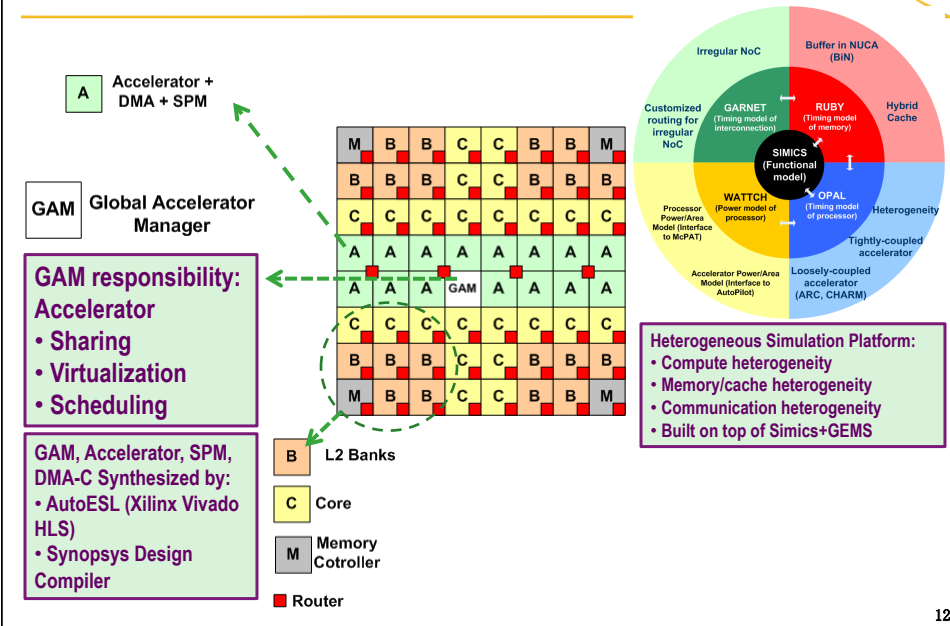
## AutoPilot Compilation Tool (based UCLA xPilot system)



- ◆ Platform-based C to FPGA 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

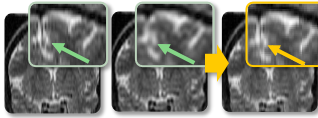
Developed by AutoESL, acquired by Xilinx in Jan. 2011

## Accelerator-Rich CMP (ARC) [DAC'12]



## Application Domain: Medical Image Processing

reconstruction



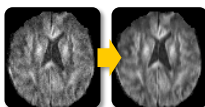
Medical images exhibit sparsity, and can be sampled at a rate  $\ll$  classical Shannon - Nyquist theory :

$$\min_u \sum_{\text{sampled points}} \|ARu - S\|^2 + \lambda \sum_{\text{voxels}} \|grad(u)\|$$

compressive sensing

---

denoising




$$\forall \text{voxel} : u(i) = \sqrt{\left( \sum_{\text{voxel} \in \text{volume}} w_{ij} f(j)^2 \right) - 2\sigma^2}, w_{ij} = \frac{1}{Z(i)} e^{-\frac{\sqrt{\sum_{k=1}^3 \|v_k - z_k\|^2}}{h}}$$

total variational algorithm

---

registration



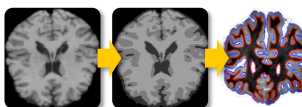
$$v = \frac{\partial u}{\partial t} + v \cdot \nabla u$$

$$\mu \Delta v + (u + \eta) \nabla (\nabla \cdot v) = -[T(x - u) - R(x)] \nabla T(x - u)$$

fluid registration

---

segmentation



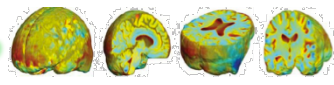
$$\frac{\partial \phi}{\partial t} = |\nabla \phi| \left[ F(\text{data}, \phi) + \lambda \text{div} \left( \frac{\nabla \phi}{|\nabla \phi|} \right) \right]$$

$$\text{surface}(t) = \{ \text{voxels } x : \phi(x, t) = 0 \}$$

level set methods

---

analysis



$$\frac{\partial v}{\partial t} + (v \cdot \nabla)v = -\nabla p + \nu \Delta v + f(x, t)$$

$$\frac{\partial v_i}{\partial t} + \sum_{j=1}^3 v_j \frac{\partial v_i}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \nu \sum_{j=1}^3 v_j \frac{\partial^2 v_i}{\partial x_j^2} + f_i(x, t)$$

Navier-Stokes equations

13

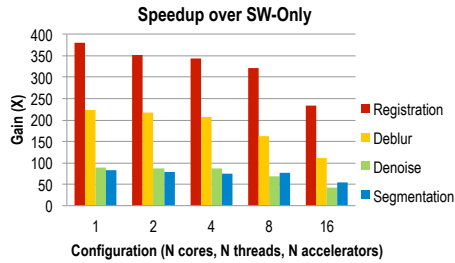
## Area Overhead

	Core	NoC	L2	GAM	Deblur	Denoise	Segmentation	Registration	SPM Banks
Number of instance/Size	1	1	8MB	1	1	1	1	1	39 x 2KB
Area(mm^2)	10.8	0.3	39.8	0.012	2.01	0.49	0.69	3.85	1.44
Percentage (%)	18.2	0.5	67.0	0.02	3.4	0.8	1.2	6.5	2.4
Total Accelerators + GAM + SPMs: 14.3 %									

- ◆ AutoESL (from Xilinx) for C to RTL synthesis
- ◆ Synopsys for ASIC synthesis
  - 32 nm Synopsys Educational library
- ◆ CACTI for L2
- ◆ Orion for NoC
- ◆ One UltraSparc III core (area scaled to 32 nm)
  - 178.5 mm^2 in 0.13 um ([http://en.wikipedia.org/wiki/UltraSPARC\\_III](http://en.wikipedia.org/wiki/UltraSPARC_III))

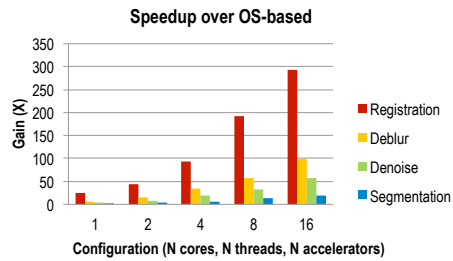
14

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



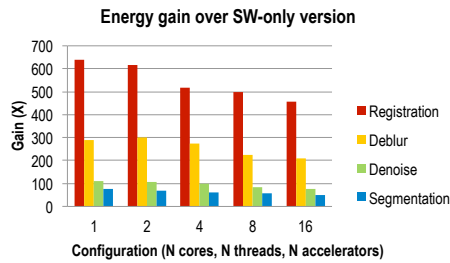
**Performance improvement over SW only approaches: on average 168x, up to 380x**

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



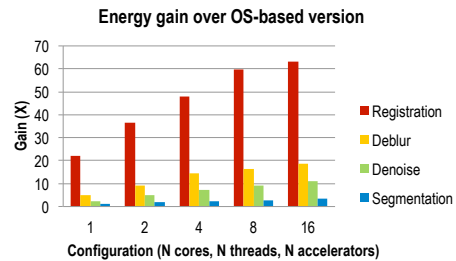
15

## 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**



16



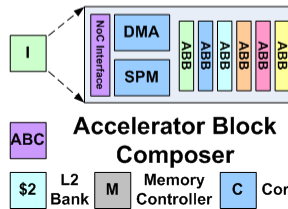
## What are the Problems with ARC?

- ◆ **Dedicated accelerators are inflexible**
  - An LCA may be useless for new algorithms or new domains
  - Often under-utilized
  - LCAs contain many replicated structures
    - Things like fp-ALUs, DMA engines, SPM
    - Unused when the accelerator is unused
- ◆ **We want flexibility and better resource utilization**
  - Solution: CHARM
- ◆ **Private SPM is wasteful**
  - Solution: BiN

17

## Fine-grain Accelerator Composition + Globally-managed Buffer in NUCA [ISLPED'12]

- ◆ **ABB**
  - Accelerator building blocks (ABB)
  - Primitive components that can be composed into accelerators
- ◆ **ABB islands**
  - Multiple ABBs
  - Shared DMA controller, SPM and NoC interface
- ◆ **ABC**
  - Accelerator Block Composer (ABC)
  - Runtime composition of virtual accelerators from ABBs
  - Arbitrate requests from cores
- ◆ **Other components**
  - Cores
  - L2 Banks
  - Memory controllers

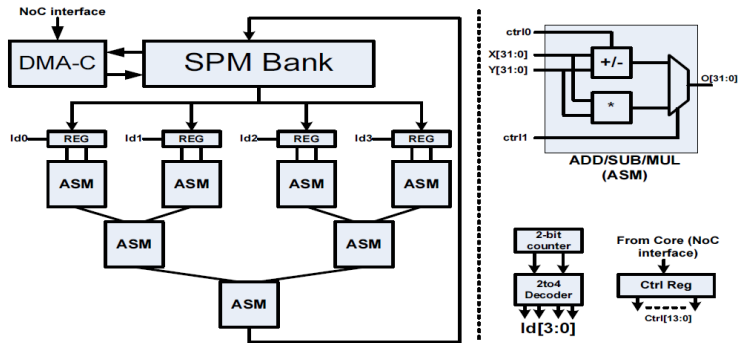


18

## An Example of ABB Library (for Medical Imaging)

ABBs	Denoise	Deblur	Registration	Segmentation
Float Reciprocal (FInv)	✓	✓		✓
Float Square-Root (FSqrt)	✓	✓	✓	✓
Float Polynomial-16 (Poly16)	✓	✓	✓	✓
Float Divide (FDiv)	✓	✓	✓	✓

Internal of Poly



19

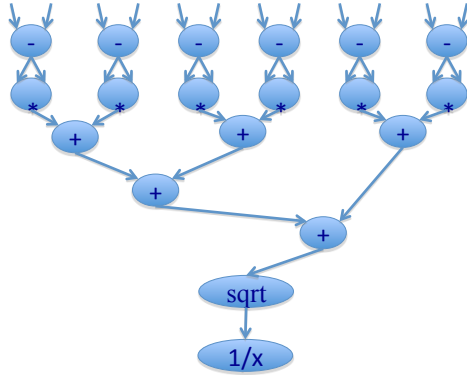
## Example of ABB Flow-Graph (Denoise)

$$1 / \sqrt{\sum_{i=0}^6 (X_i - Y)^2}$$

20

**Example of ABB Flow-Graph (Denoise)**

$$1 / \sqrt{\sum_{i=0}^6 (Xi - Y)^2}$$



21

**Example of ABB Flow-Graph (Denoise)**

$$1 / \sqrt{\sum_{i=0}^6 (Xi - Y)^2}$$

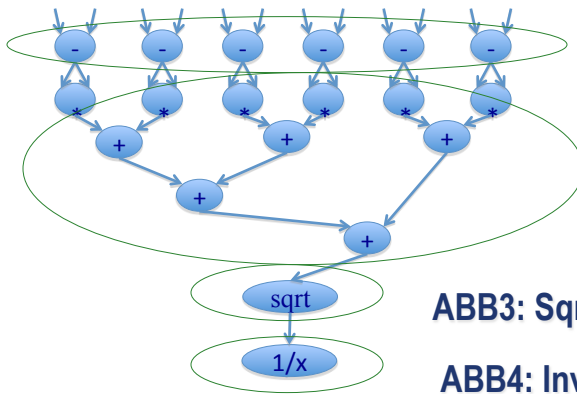


ABB1: Poly

ABB2: Poly

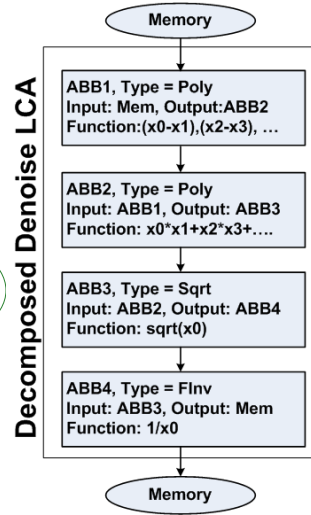
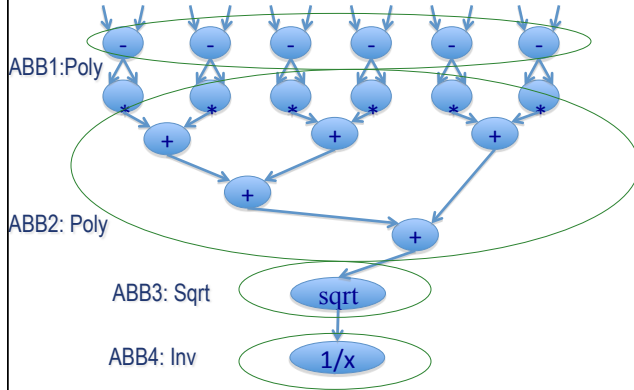
ABB3: Sqrt

ABB4: Inv

22

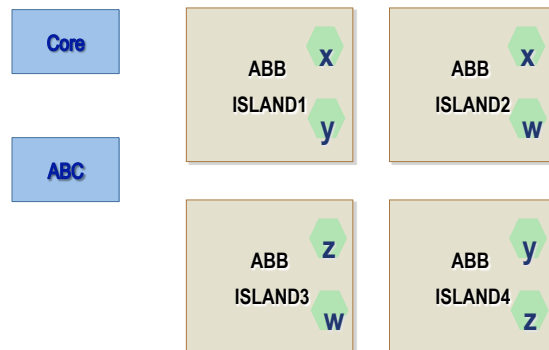
### Example of ABB Flow-Graph (Denoise)

$$1 / \sqrt{\sum_{i=0}^6 (Xi - Y)^2}$$



23

### LCA Composition Process

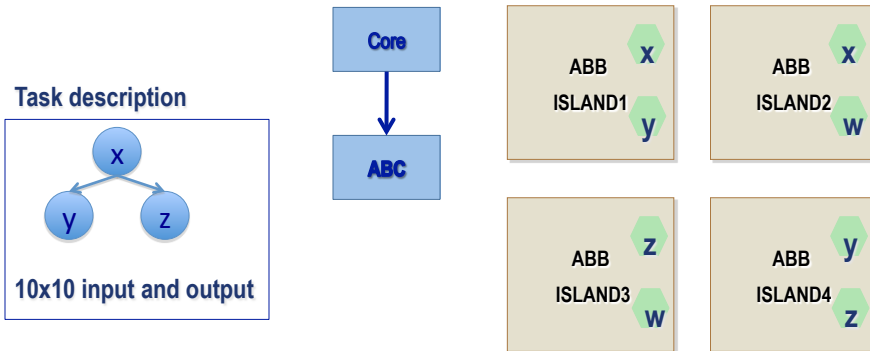


24

## LCA Composition Process

### 1. Core initiation

- Core sends the task description: task flow-graph of the desired LCA to ABC together with polyhedral space for input and output

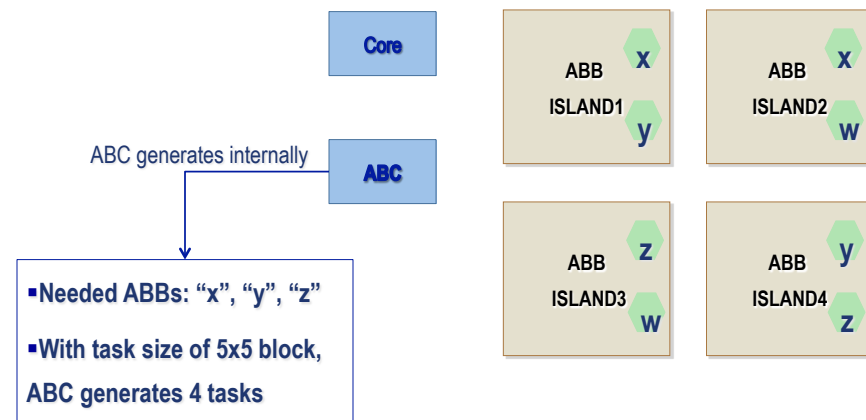


25

## LCA Composition Process

### 2. Task-flow parsing and task-list creation

- ABC parses the task-flow graph and breaks the request into a set of tasks with smaller data size and fills the task list

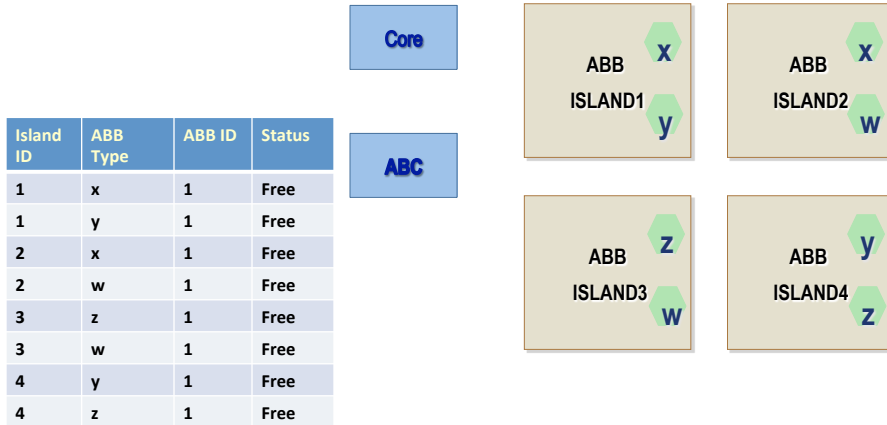


26

## LCA Composition Process

### 3. Dynamic ABB mapping

- ABC uses a pattern matching algorithm to assign ABBs to islands
- Fills the composed LCA table and resource allocation table

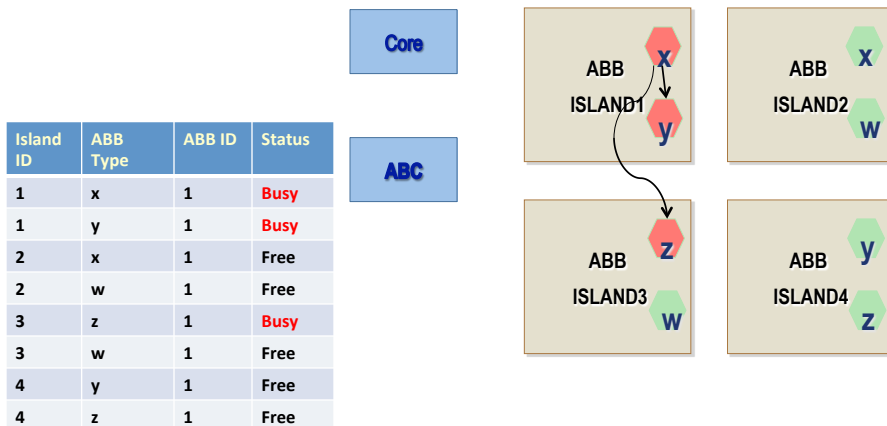


27

## LCA Composition Process

### 3. Dynamic ABB mapping

- ABC uses a pattern matching algorithm to assign ABBs to islands
- Fills the composed LCA table and resource allocation table

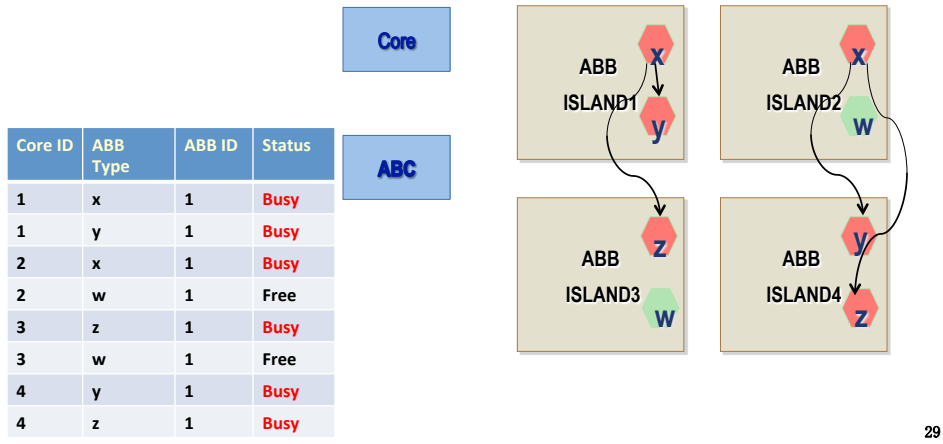


28

## LCA Composition Process

### 4. LCA cloning

- Repeat to generate more LCAs if ABBs are available

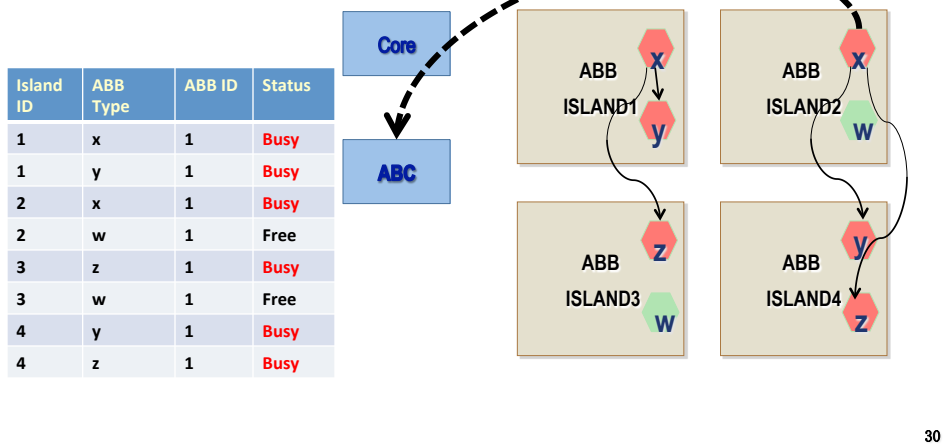


29

## LCA Composition Process

### 5. ABBs finishing task

- When ABBs finish, they signal the ABC. If ABC has another task it sends otherwise it frees the ABBs

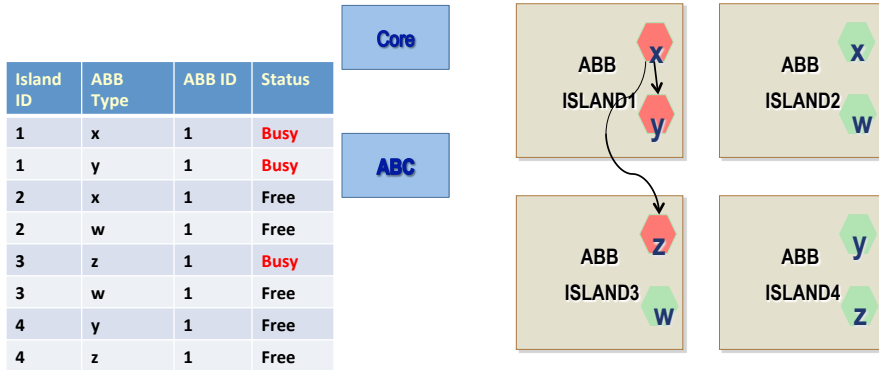


30

## LCA Composition Process

### 5. ABBs being freed

- When an ABB finishes, it signals the ABC. If ABC has another task it sends otherwise it frees the ABBs

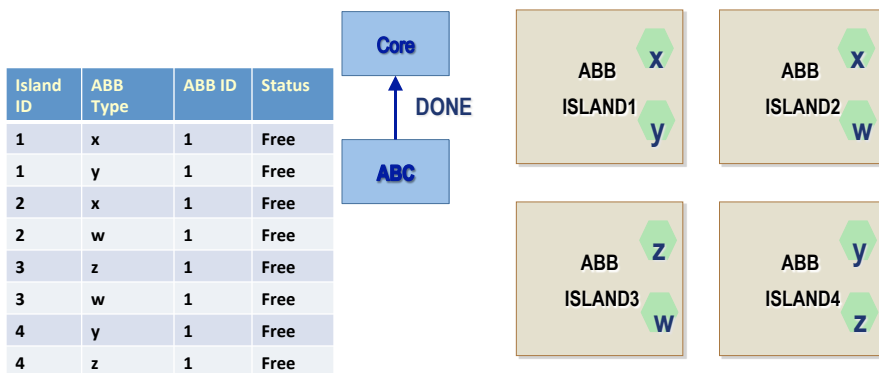


31

## LCA Composition Process

### 6. Core notified of end of task

- When the LCA finishes ABC signals the core



32



## CHARM Software Infrastructure

### ◆ ABB type extraction

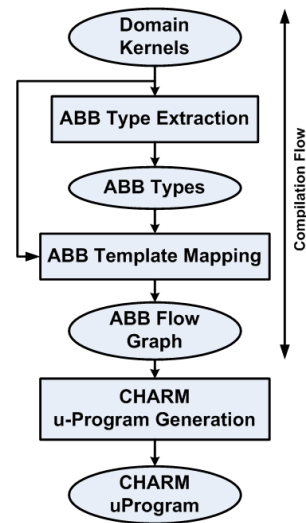
- Input: compute-intensive kernels from different application
- Output: ABB Super-patterns
- Currently semi-automatic

### ◆ ABB template mapping

- Input: Kernels + ABB types
- Output: Covered kernels as an ABB flow-graph

### ◆ CHARM uProgram generation

- Input: ABB flow-graph
- Output:



33

## ABB Template Mapping

### ◆ Scalability problem

- NP-complete
  - The flow in [S. Mahlke, et.al. CASE'06] takes ~ 30 min for *segmentation* (1147 connected ABB candidates)
- Solution: only consider maximal ABB candidates + efficient pruning techniques

### ◆ Maximal ABB candidate identification

- Input: Kernels + ABB types
- Output: Maximal ABB candidates

### ◆ Maximal ABB mapping

- Input: Kernels + maximal ABB candidates
- Output: Covered kernels as an ABB flow-graph

### ◆ Mapping efficiency

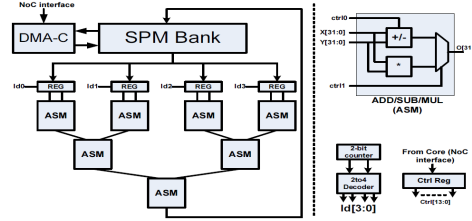
- Up to 26X reduction on total number of ABB candidates in the mapping phase
- Obtain the optimal solution in less than 6 sec. with ~ 1200 ABB candidates

34

## Area Overhead Analysis

### ◆ Area-equivalent

- The total area consumed by the ABBs equals the total area of all LCAs required to run a single instance of each benchmark



### ◆ Total CHARM area is 14% of the 1cmx1cm chip

- A bit less than LCA-based design

Name	A(u <sup>2</sup> )	P(mW)	Total#
FDiv	4949	0.264	12
Poly16	38276	1.608	96
FInv	3503	0.141	12
FSqrt	58683	1.83	8
SPM-4KB 1R/W	13591	17.6	288
SPM-768B 1R/W	2545	7	72
ABC	8383	0.066	1

Core	NoC	Cache & Dir	CHARM-HW	CHARM-Total	LCA HW	LCA Total
10.8mm <sup>2</sup> (scaled to 32nm)	0.3mm <sup>2</sup> (Orion)	39.8mm <sup>2</sup> (Cacti)	8.3mm <sup>2</sup> (14%)	59.2mm <sup>2</sup>	8.5mm <sup>2</sup> (14.3%)	59.4mm <sup>2</sup>

35

## Results: Improvement Over LCA-based Design

### ◆ Use same area as LCA (loosely coupled accelerators) based design

### ◆ Performance

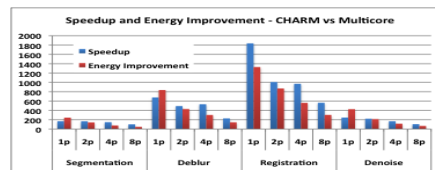
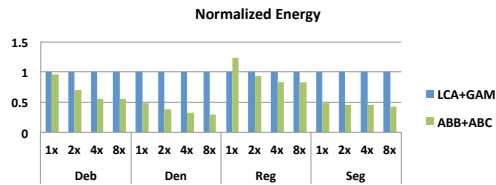
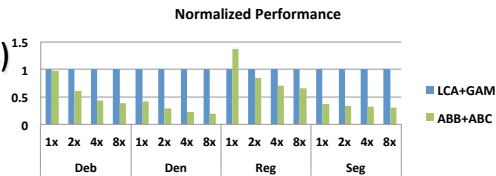
- 2.5X vs. LCA+GAM (max 5X)
- 476X vs Multi-core (max 1800 X)

### ◆ Energy

- 1.9X vs. LCA+GAM (max 3.4X)
- 381X vs. Multi-core (max 1300X)

### ◆ ABB+ABC has better energy and performance

- ABC starts composing ABBs to create new LCAs
- Creates more parallelism



36

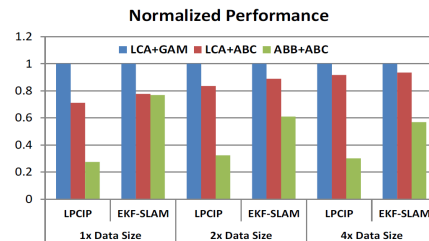
## Results: Platform Flexibility

### ◆ Two applications from two unrelated domains to MI

- Computer vision
  - Log-Polar Coordinate Image Patches (LPCIP)
- Navigation
  - Extended Kalman Filter-based Simultaneous Localization and Mapping (EKF-SLAM)

### ◆ Only one ABB is added

- Indexed Vector Load



MAX Benefit over LCA+GAM	3.64X
AVG Benefit over LCA+GAM	2.46X

37

## Memory Management for Accelerator-Rich Architectures [ISLPED'2012]

### ◆ Providing a private buffer for each accelerator is very inefficient.

- Large private buffers: occupy a considerable amount of chip area
- Small private buffers: less effective for reducing off-chip bandwidth

### ◆ Not all accelerators are powered-on at the same time

- Shared buffer [Lyonsy et al. TACO'12]
- Allocate the buffers in the cache on-demand [Fajardo et al. DAC'11][Cong et al. ISLPED'11]

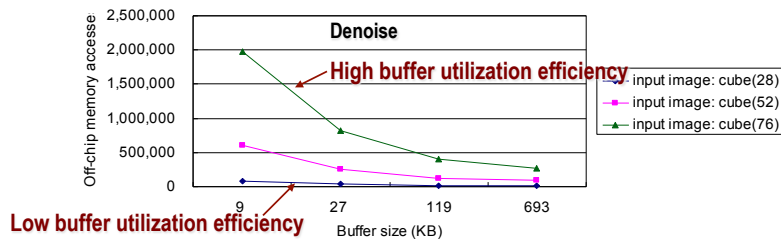
### ◆ Our solution

- BiN: A Buffer-in-NUCA Scheme for Accelerator-Rich CMPs

38

## Buffer Size vs. Off-chip Memory Access Bandwidth

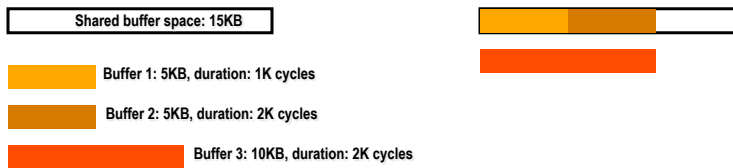
- ◆ Buffer size ↑ - off-chip memory bandwidth ↓: covering longer reuse distance [Cong et al. ICCAD'11]
- ◆ Buffer size vs. bandwidth curve: BB-Curve
- ◆ Buffer utilization efficiency
  - Different for various accelerators
  - Different for various inputs for one accelerator
- ◆ Prior work: no consideration of global allocation at runtime
  - Accept fixed-size buffer allocation requests
  - Rely on the compiler to select a single, 'best' point in the BB-Curve



39

## Resource Fragmentation

- ◆ Prior work allocates a **contiguous** space to each buffer to simplify buffer access
- ◆ Requested buffers have unpredictable space demand and come in dynamically: **resource fragmentation**
- ◆ NUCA complicates buffer allocations in cache
  - The distance of the cache bank to the accelerator also matters
- ◆ To support fragmented resources: **paged allocation**
  - Analogous to a typical OS-managed virtual memory
- ◆ Challenges:
  - Large private page tables have high energy and area overhead
  - Indirect access to a shared page table has high latency overhead



40

## BiN: Buffer-in-NUCA

### ◆ Goals of Buffer-in-NUCA (BiN)

- Towards optimal on-chip storage utilization
- Dynamically allocate buffer space in the NUCA among a large number of competing accelerators

### ◆ Contributions of BiN:

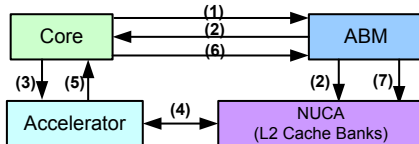
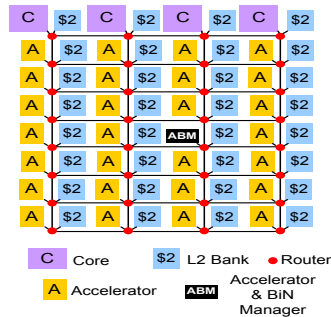
- Dynamic interval-based global (DIG) buffer allocation: address the buffer resource contention
- Flexible paged buffer allocation: address the buffer resource fragmentation

41

## Accelerator-Rich CMP with BiN

### Overall architecture of ARC [Cong et al. DAC 2011] with BiN

- Cores (with private L1 caches)
- Accelerators
  - Accelerator logic
  - DMA-controller
  - A small storage for the control structure
- The accelerator and BiN manager (ABM)
  - Arbitration over accelerator resources
  - Allocates buffers in the shared cache (BiN management)
- NUCA (shared L2 cache) banks



- (1) The core sends the accelerator and buffer allocation request with the BB-Curve to ABM.
- (2) ABM performs accelerator allocation, buffer allocation in NUCA, and acknowledges the core.
- (3) The core sends the control structure to the accelerator.
- (4) The accelerator starts working with its allocated buffer.
- (5) The accelerator signals to the core when it finishes.
- (6) The core sends the free-resource message to ABM.
- (7) ABM frees the accelerator and buffer in NUCA.

42

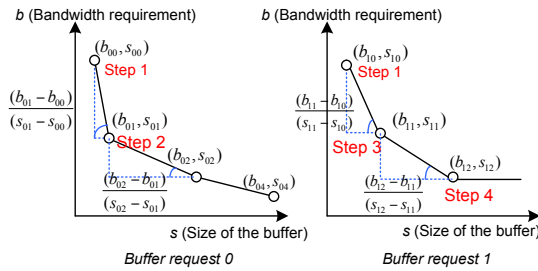
## Dynamic Interval-based Global (DIG) Allocation

◆ Perform global allocation for buffer allocation requests in an interval

- Keep the interval short (10K cycles): Minimize waiting-in-interval
- If 8 or more buffer requests, the DIG allocation will start immediately

◆ An example: 2 buffer allocation requests

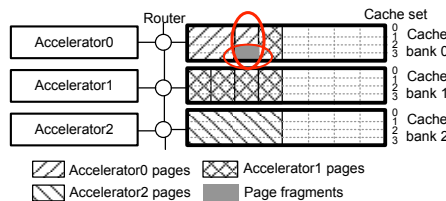
- Each point (b, s)
  - s: buffer size
  - b: corresponding bandwidth requirement at s
  - Buffer utilization efficiency at each point:  $(b_j - b_{i(j-1)}) / (s_j - s_{i(j-1)})$
- The points are in non-decreasing order of buffer size



43

## Flexible Paged Allocation

- ◆ Set the page size according to buffer size: Fixed total number of pages for each buffer
- ◆ BiN manager locally keep the information of the current contiguous buffer space in each L2 bank
  - Since all of the buffer allocation and free operations are performed by BiN manager
- ◆ Allocation: starting from the nearest L2 bank to this accelerator, to the farthest
- ◆ We allow the last page (source of page fragments) of a buffer to be smaller than the other pages of this buffer
  - No impact on the page table lookup
  - The max page fragment will be smaller than the min-page
  - The page fragments do not waste capacity since they can be used by cache



44

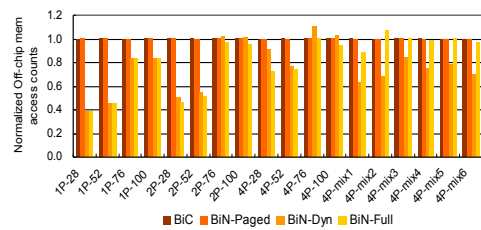
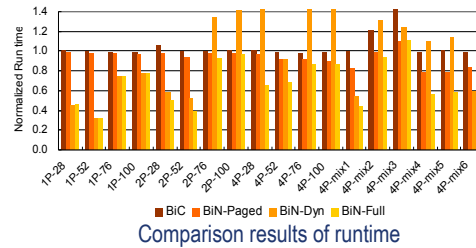
## Reference Design Schemes

- ◆ **Accelerator Store (AS) [Lyonsy, et al. TACO'12]**
  - Separate cache and shared buffer module
  - Set the buffer size 32% larger than maximum buffer size in BiN: overhead of buffer-in-cache
  - Partition the shared buffer into 32 banks distributed them to the 32 NoC nodes
- ◆ **BiC [Fajardo, et al. DAC'11]**
  - BiC dynamically allocates contiguous cache space to a buffer
  - Upper bound: limiting buffer allocation to at most half of each cache bank
  - Buffers can span multiple cache banks
- ◆ **BiN-Paged**
  - Only has the proposed paged allocation scheme
- ◆ **BiN-Dyn**
  - Based on BiN-Paged, it also performs dynamic allocation without consideration of near future buffer requests
  - It responds to a request immediately by greedily satisfying the request with the current available resources
- ◆ **BiN-Full**
  - This is the entire proposed BiN scheme

45

## Impact of Dynamic Interval-based Global Allocation

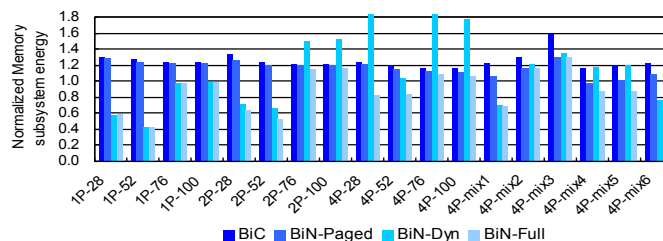
- ◆ **BiN-Full consistently outperforms the other schemes**
  - The only exception: 4P-mix3
    - 1.32X larger capacity of the AS can accommodate all buffer requests
- ◆ **Overall, compared to the accelerator store and BiC, BiN-Full reduces the runtime reduction by 32% and 35%, respectively**



46

## Impact on Energy

- ◆ **AS consumes the least per-cache/buffer access energy and the least unit leakage**
  - Because in the accelerator store the buffer and cache are two separate units
- ◆ **BiN-Dyn**
  - Saves energy in cases where it can reduce the off-chip memory accesses and runtime
  - Results in a large energy overhead in cases where it significantly increases the runtime
- ◆ **Compared with the AS, BiN-Full reduces the energy by 12% on average**
  - Exception: 4P-mix-{2,3}
    - The 1.32X capacity of AS can better satisfy buffer requests
- ◆ **Compared with BiC, Bin-Full reduces the energy by 29% on average**



47

## Concluding Remarks

- ◆ **Platform-based design methodology for accelerator-rich CMPs (ARCs)**
  - Enabled by GSRC support over past 15 years
- ◆ **ARCs provide huge opportunity for performance/energy improvement**
- ◆ **Runtime accelerator composition offers**
  - Flexibility in application adaptation
  - Better resource utilization for scalable parallelism support
- ◆ **Compilation and runtime supports are critical**

48



## Acknowledgements

- ◆ Supports from GSRC and NSF
- ◆ Collaboration with faculty in Center for Domain-Specific Computing in the past 3 years.



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)



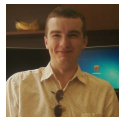
Vese  
(UCLA)

49

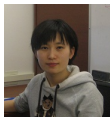
## More Acknowledgements



Mohammad Ali Ghodrat



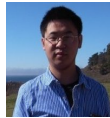
Michael Gill



Hui Huang



Chunyue  
Liu



Yi Zou



Beayna  
Grigorian

- ◆ This research is partially supported by the Center for Domain-Specific Computing (CDSC) funded by the NSF Expedition in Computing Award CCF-0926127, GSRC under contract 2009-TJ-1984.

50