Maximizing Data Reuse for Minimizing Memory Space Requirements and Execution Cycles

M. Kandemir, G. Chen and F. Li
Pennsylvania State University

Outline

- Introduction
- Determine minimum on-chip memory
  - Data-centric view for reuse
  - Data Block Graph (DBG)
  - Scheduling
  - Code generation
- Scheduling under capacity constraints
- Experiments
- Summary

Introduction

- Constraints for embedded system design
  - Power consumption
  - Memory size
- Memory management of embedded devices
  - Design suitable on-chip memory configuration
  - Reconfigure application codes to make better use of available on-chip memory
- Our proposal
  - A very aggressive one in extracting and employing data reuse under data dependence constraints

Determine minimum on-chip memory

- Array-based data-intensive applications
- Memory space unit
  - Data block
  - On-chip block
- Problem
  - Determine the minimum on-chip memory size (number of on-chip blocks) so that increasing its size cannot bring additional performance benefits

Data-Centric View for Reuse

```
do i = 1, N
  do j = 1, N
    ... = U[i][j]
end do
do k = 1, N
  do l = 1, N
    ... = U[k][l]
end do
```

(a)

```
do t1 = 1, N
  do t2 = 1, N
    ... = U[t1][t2]
    ... = U[t1][t2]
end do
```

(b)

N^2/K on-chip block

1 on-chip block

Challenging Issues

- Intro-loop and inter-loop data dependences
- Multiple arrays accessed by the application
- Complex array access patterns
- More sophisticated loop transformations required than simple fusion-like transformation

Data Block Graph

- Array indices and loop bounds are affine functions of enclosing loop indices
- *v* loop nests and their iteration spaces I_1, I_2, I_3, …, I_v
- Computation domain $\mathcal{I}_D = \bigcup_{1 \le i \le v} \mathcal{I}_I$
- The set of iterations from loop nest i to access data block j of array U: $\mathcal{I}_{U,i,j}$

$$\exists R \text{ and } \exists d \in \text{ data block } j \text{ of } U \text{ such that } R(l) = d$$

$$\bigcup_i \bigcup_U \bigcup_j \mathcal{I}_{U,i,j} = \mathcal{I}_D$$

Data Block Graph (cont'd)

- Nodes: all $\mathcal{I}_{U,i,j}$s
- Edges: data dependence relationships between two nodes
- Execution of computation domain: visiting each node of DBG
- Legal Execution: traversal of DBG respecting all data dependences

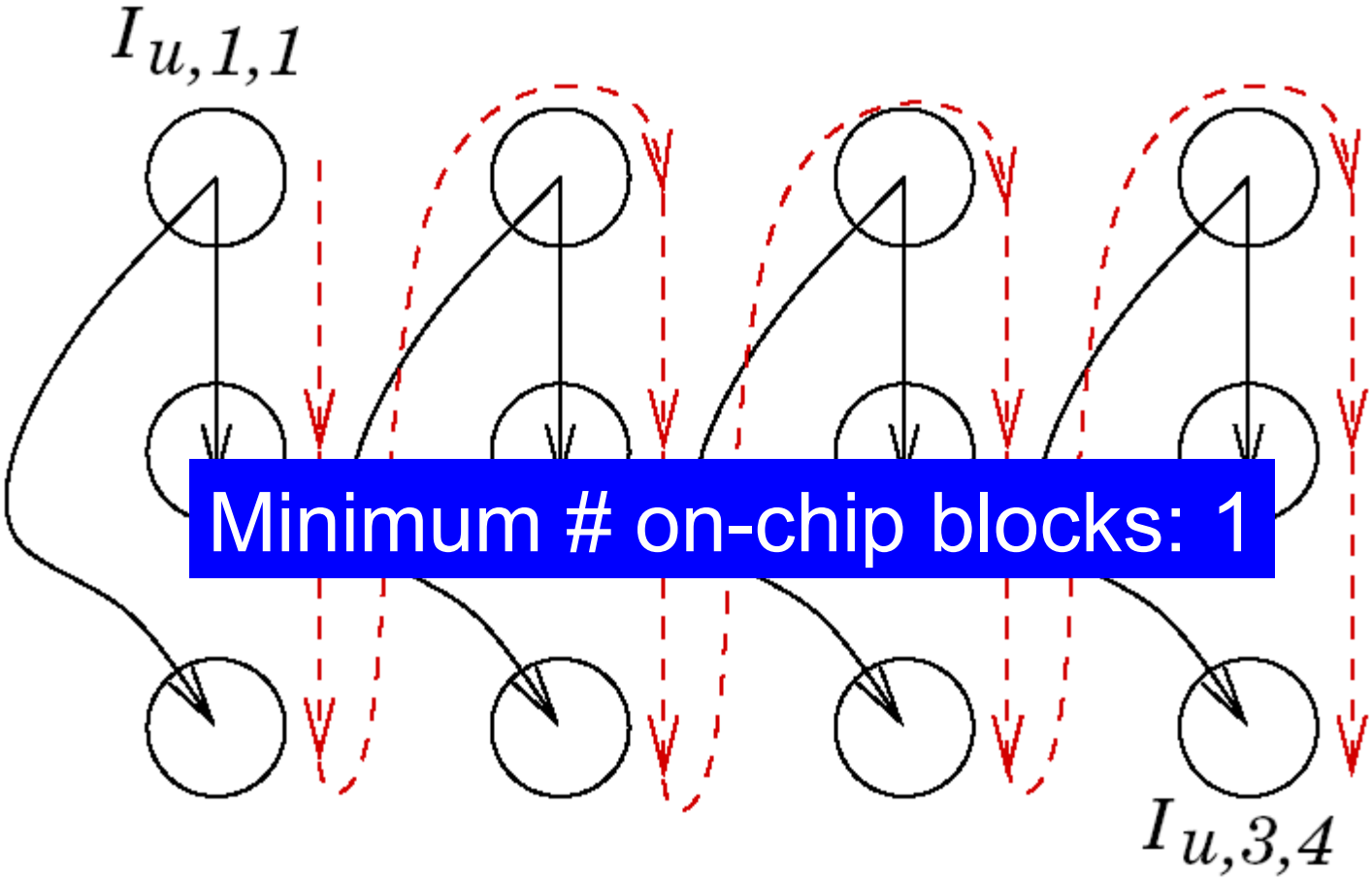
Scheduling

- Maximize data reuse for a DBG
  - Schedule two nodes: $\mathcal{I}_{U,i,j}$ and $\mathcal{I}_{U,i',j}$ one after another by observing the data dependences
- Conventional loop transformation achieve data reuse by transforming each loop nest individually
- Our approach considers the entire computation domain to extract more reuse than existing locality-enhancing techniques such as loop tiling and loop fusion

Scheduling for Single Array Case

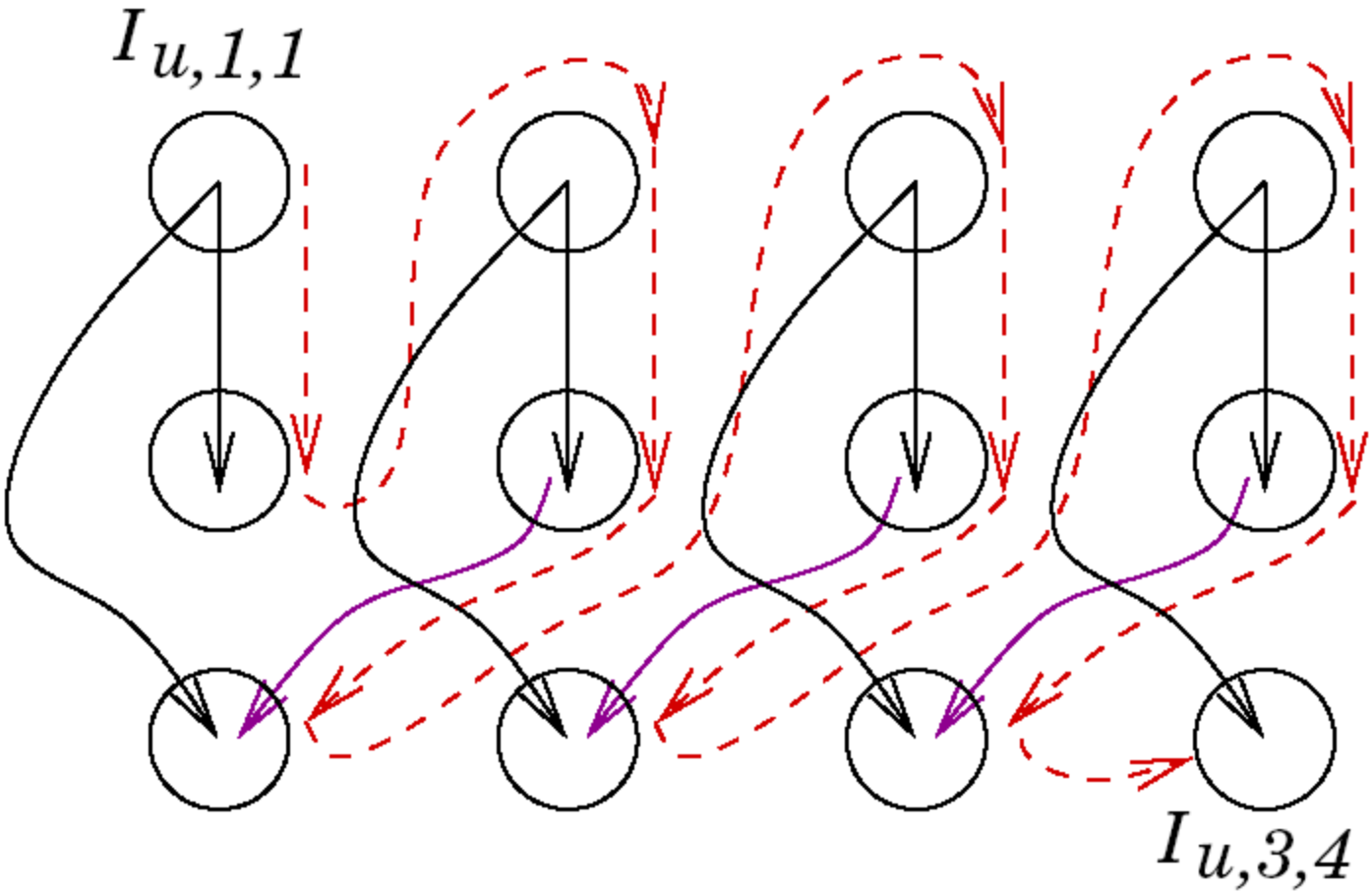
- Heuristic scheduling based on list scheduling
  - Select one node from DBG at one time and schedule it
  - Data dependence constraints: All DBG nodes on which this node depends must be already scheduled
  - Data reuse constraints: This node should access the same data block as the previous node
    - All nodes accessing the current block have been scheduled – recycle on-chip block for another data block
    - Other unscheduled nodes still need this block – increase the number of on-chip blocks by 1

```
do i = 1, N
  U[i] = ...
end do
do j = 1, N
  ... = U[j]
end do
do k = 1, N
  ... = U[k]
end do
```



Array U:

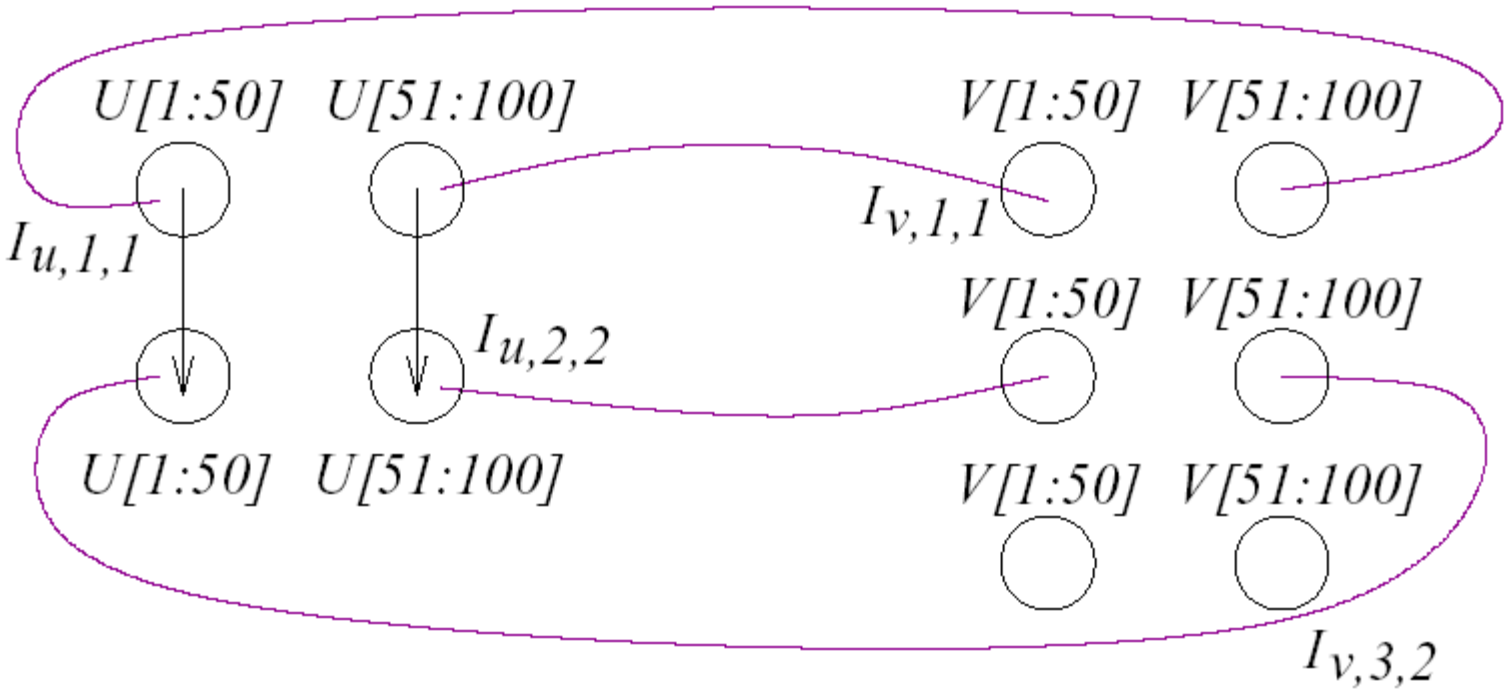
Block 1	Block 2	Block 3	Block 4



Minimum # on-chip blocks: 2

Scheduling for Multiple Arrays Case

- Single array centric
  - Accesses to one or two arrays are dominating
  - Determine the number of on-chip blocks required to minimize the number of on-chip block updates
  - Perform each array in turn and choose the best
  - Drawback: fail to capture the coupling between references to different arrays
- Global method
  - Combined DBG (CDBG) by adding locality edges



Minimum # on-chip blocks: 2

Code Generation

- Polyhedral tool (Omega Library)
  - Given the iteration space, generate the loop code that traverse the iteration space

$$U[i+j-1][j+k+3]$$

$$IS = \{[i,j,k] : (LB_i \leq i \leq UB_i) \wedge (LB_j \leq j \leq UB_j) \wedge (LB_k \leq k \leq UB_k)\}.$$

$$U[G][F], U[G][F+1], U[G][F+2], \ldots, U[G][F+K-1]$$

$$DB = \{[a,b] : (a = G) \wedge (F \leq b \leq F+K-1) \wedge ([a:b] \in DS)\}.$$

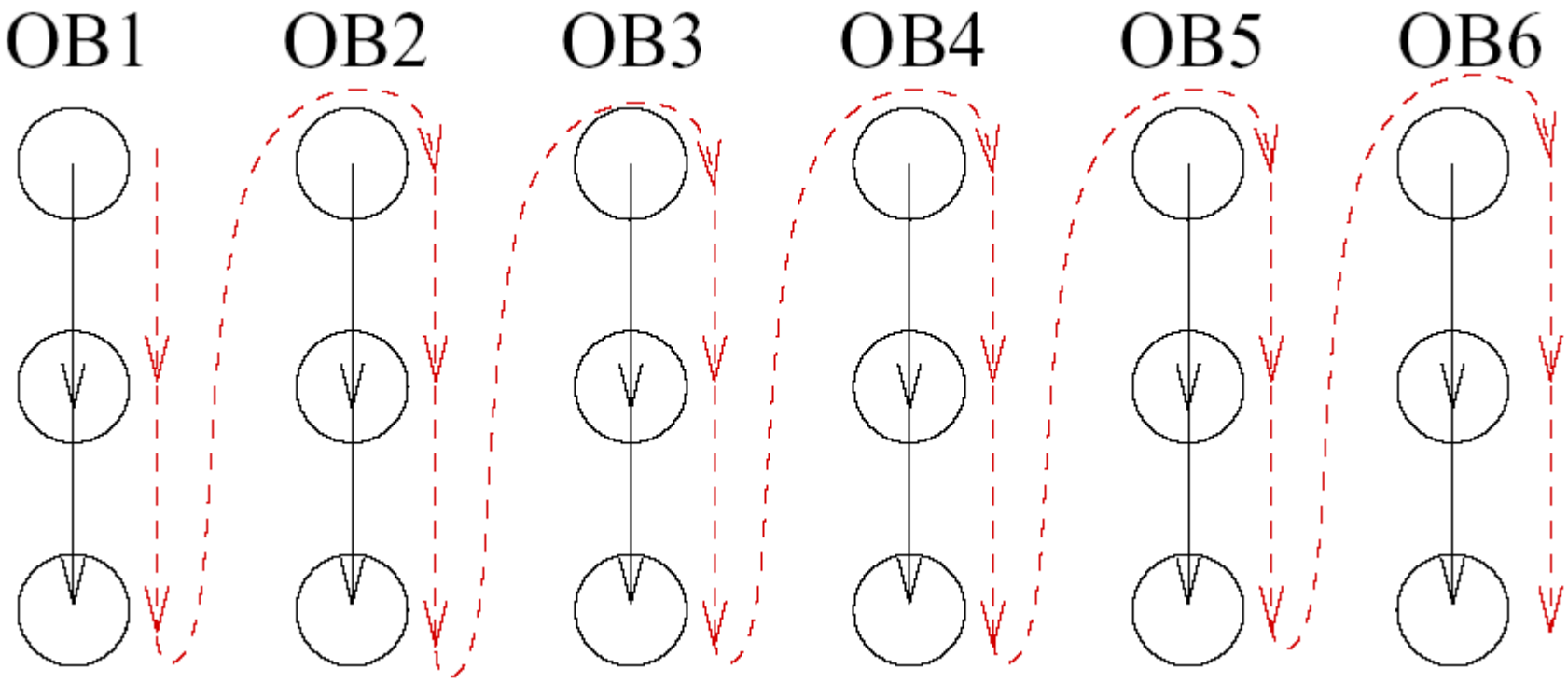
$$ND = \{[i,j,k] : \exists a \exists b \text{ such that } (a = i+j-1) \wedge (b = j+k+3) \wedge ([i,j,k] \in IS) \wedge ([a,b] \in DB)\}.$$

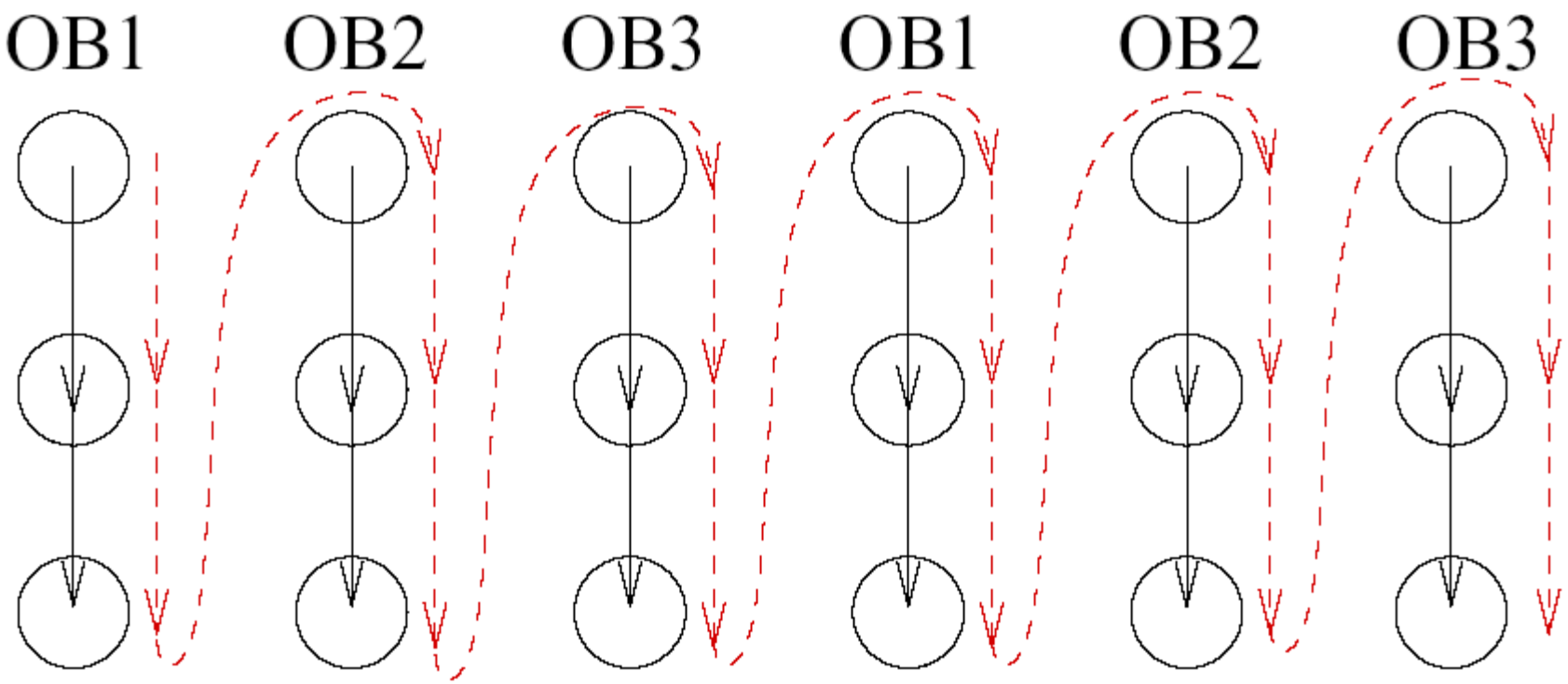
Code Generation (Cont'd)

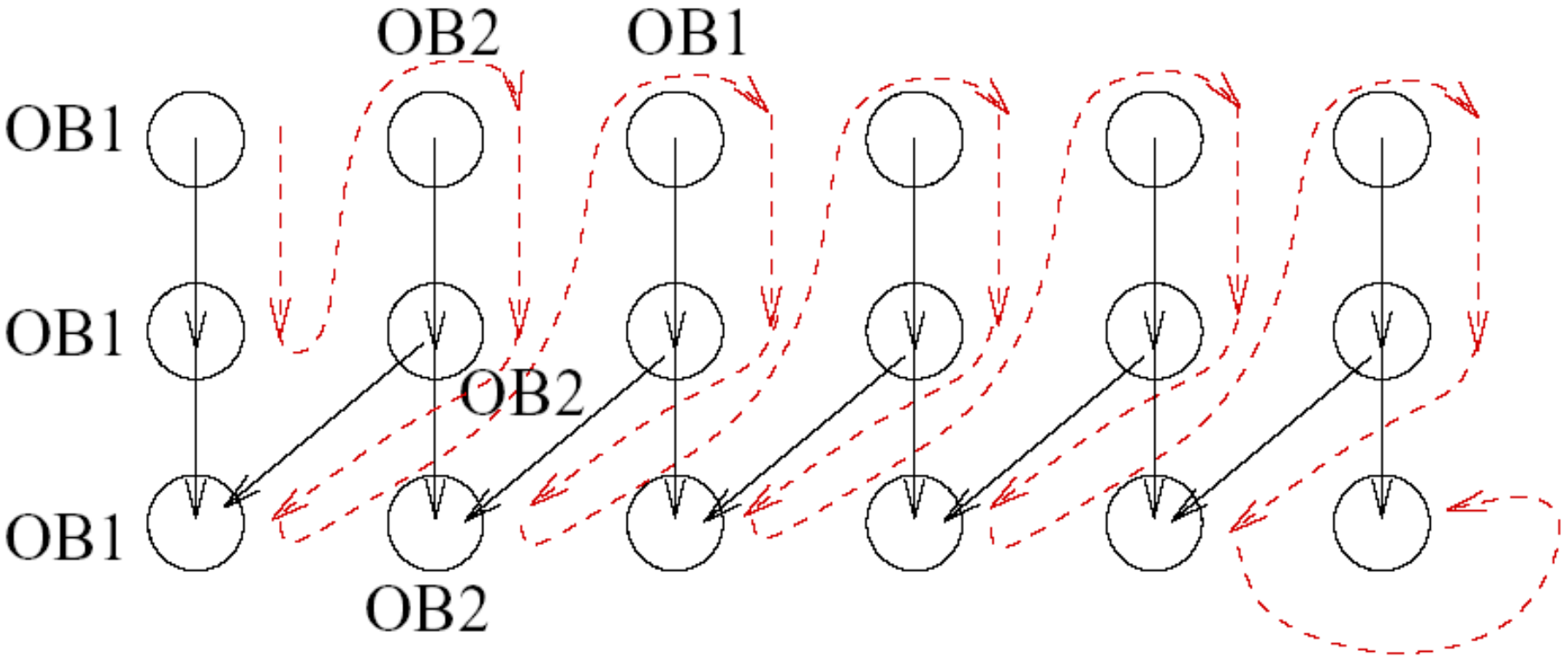
```
do G = 1, N
 do F = 1, N, K
  if (K ≥ 1 && UB3 ≥ LB3) {
   do t1 = max(G-F-K+LB3+5,LB1,G-UB2+1),
               min(G+UB3-F+4,UB1,G-LB2+1)
    do t3 = max(LB3,-G+t1+F-4),
               min(UB3,-G+t1+F+K-5)
     ...U[G,G-t1+t3+4]...
    end do
   end do
  }
 end do
end do
```

Scheduling under capacity constraints

- How we schedule a DBG with a fixed number of on-chip blocks?
- Objective: to minimize the number of on-chip block loads/updates as much as possible
- Given on-chip blocks number: r and the number of node chains in DBG: t
  - r≥t: assign a private on-chip block per chain
  - r<t: LRU-based victim selection


OB1
OB2
OB3
OB4
OB5
OB6


OB1
OB2
OB3
OB1
OB2
OB3


OB2
OB1
OB1
OB1
OB2
OB1
OB2

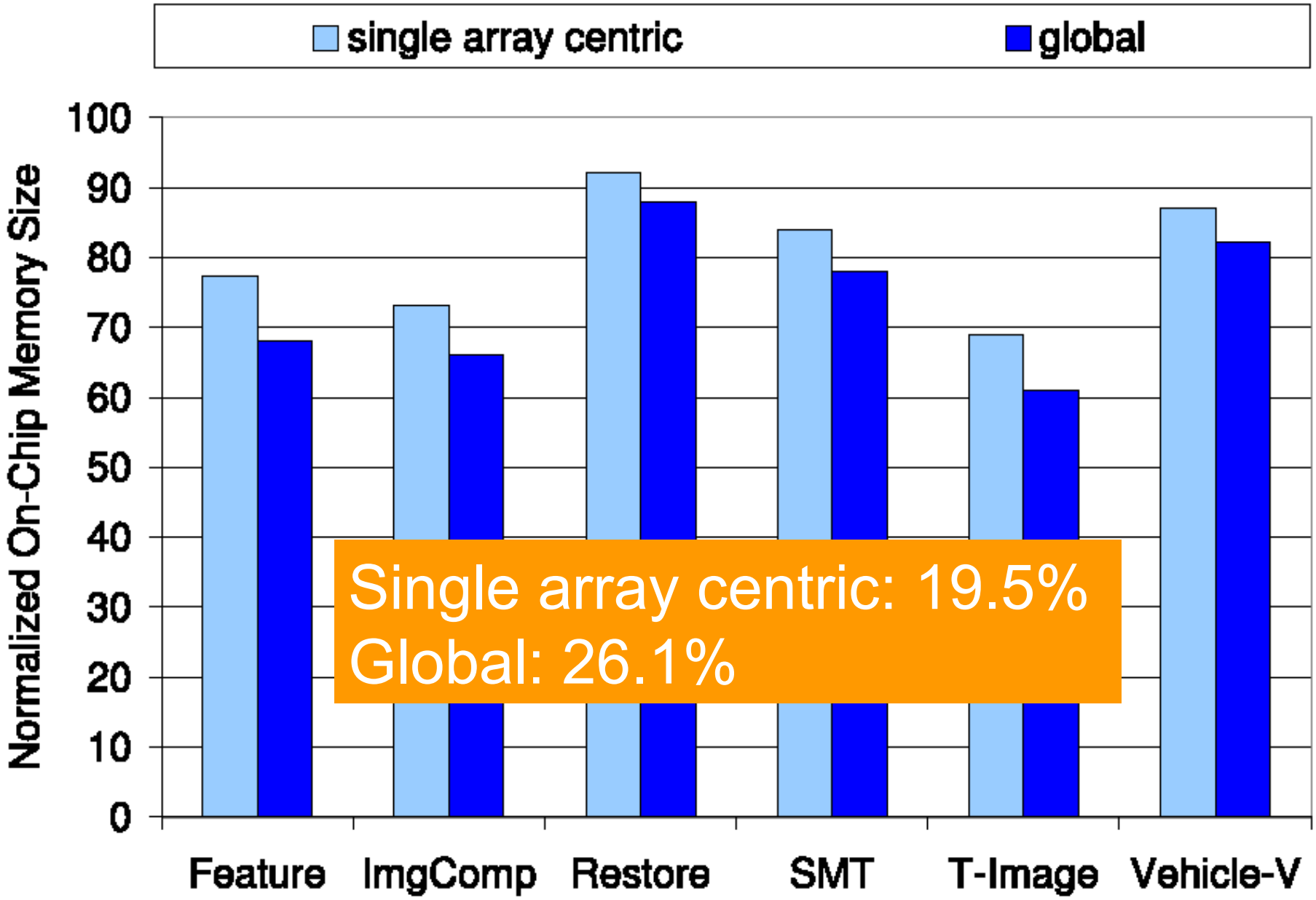
Setup

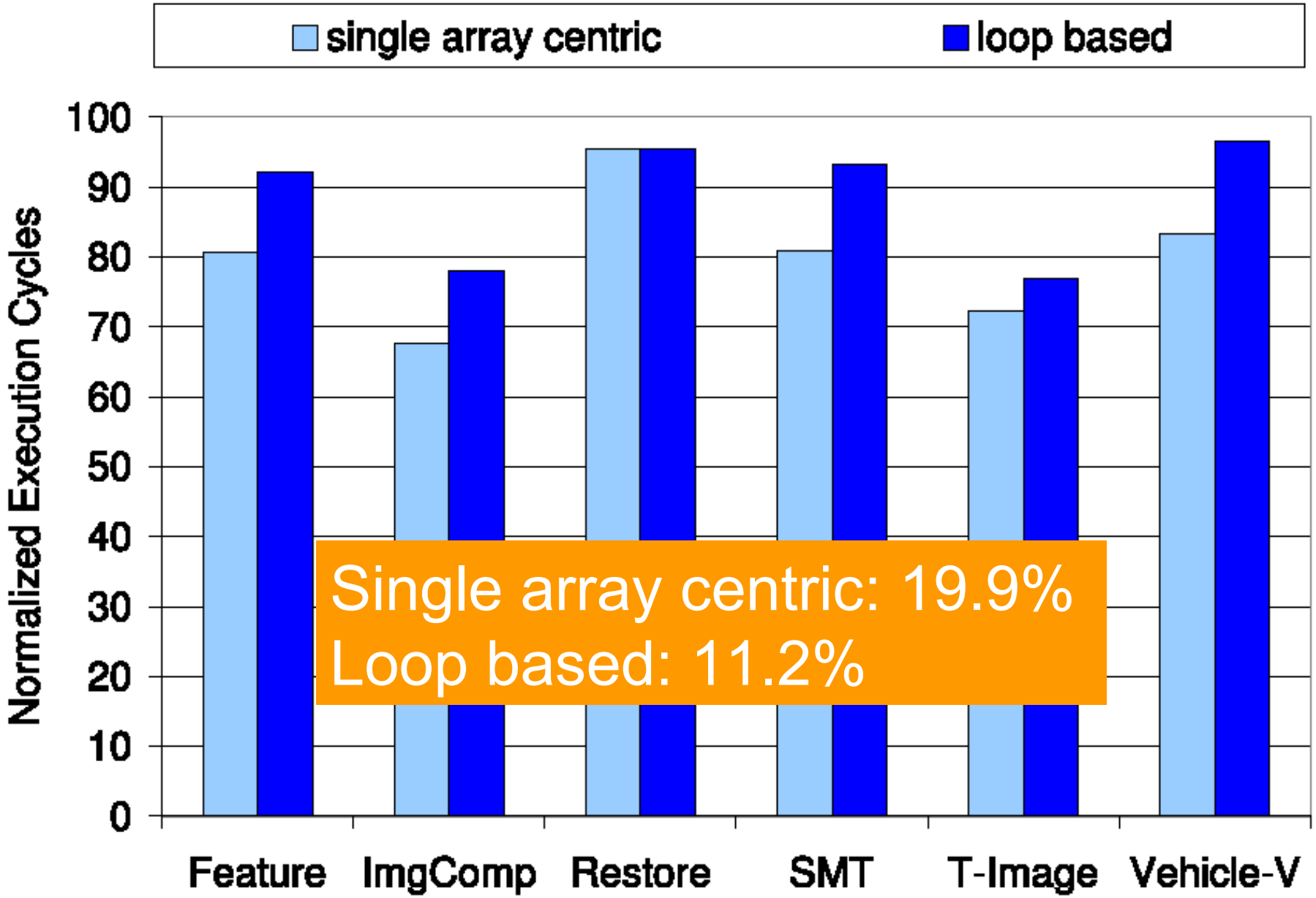
Parameter	Value
CPU	
Functional Units	4 integer ALUs 4 FP ALUs
LSQ Size	8 Instructions
RUU Size	16 Instructions
Fetch Width	4 instructions/cycle
Decode Width	4 instructions/cycle
Issue Width	4 instructions/cycle
Commit Width	4 instructions/cycle
Fetch Queue Size	4 instructions
Clock	400 MHz
Memory Hierarchy	
On-Chip SRAM	8KB, software-managed, 1 cycle latency
Data Block Size	32 bytes
Off-Chip DRAM	32MB, 100 cycle latency

Benchmarks

Benchmark Name	Brief Description	Number of Cycles (M)	On-Chip Storage
Feature	Feature Extraction	2,015.11	366.41
ImgComp	Image Compression	760.16	110.63
Restore	Image Restoration	1,862.40	278.26
SMT	Video Smoothing	95.08	34.97
T-Image	Crowd management with	914.24	182.55
Vehicle-V	Vehicle Tracking and Classification	618.27	102.79

Benchmark Name	Number of Nodes	Number of Edges	Ave. Chain Length	Eff. Chain Length
Feature	822	2,776	11.3	9.6
ImgComp	241	995	14.0	12.3
Restore	720	2,315	13.7	9.1
SMT	103	392	9.6	6.7
T-Image	508	1,733	13.6	11.4
Vehicle-V	211	789	10.1	6.6

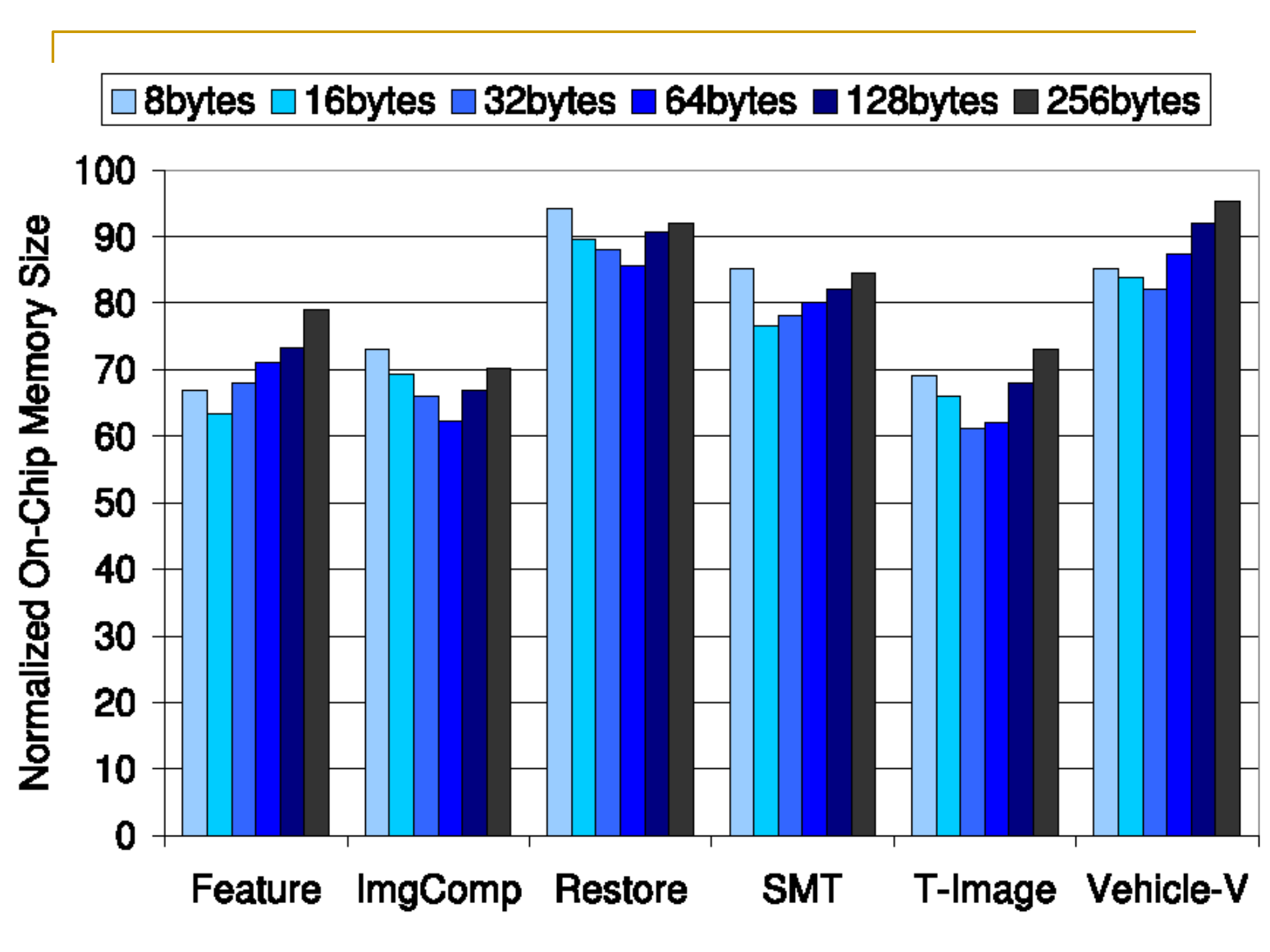

single array centric
global
Normalized On-Chip Memory Size
100
90
80
70
60
50
40
30
20
10
0
Feature
ImgComp
Restore
SMT
T-Image
Vehicle-V
Single array centric: 19.5%
Global: 26.1%


single array centric
loop based
Normalized Execution Cycles
100
90
80
70
60
50
40
30
20
10
0
Feature
ImgComp
Restore
SMT
T-Image
Vehicle-V
Single array centric: 19.9%
Loop based: 11.2%

Summary

- Targeting the optimization for constrained on-chip memory space in embedded systems
  - Determine the minimum on-chip memory capacity that minimize the frequency and volume of data transfers between off-chip and on-chip memories
  - Restructure application code to make better use of the available memory hierarchy and obtain better performance

Thank you!


8bytes
16bytes
32bytes
64bytes
128bytes
256bytes
Normalized On-Chip Memory Size
100
90
80
70
60
50
40
30
20
10
0
Feature
ImgComp
Restore
SMT
T-Image
Vehicle-V