Empowered by Innovation NEC

IMAPCAR2: A Dynamic SIMD/MIMD Mode Switching Processor for Embedded Systems

Shorin Kyo[1], Shouhei Nomoto[1], Takuya Koga[2]
Hanno Lieske[1], Shinichiro Okazaki[1]

NEC Corporation[1]
NEC Electronics Corporation[2]

HOT CHIPS 21 (August 2009)

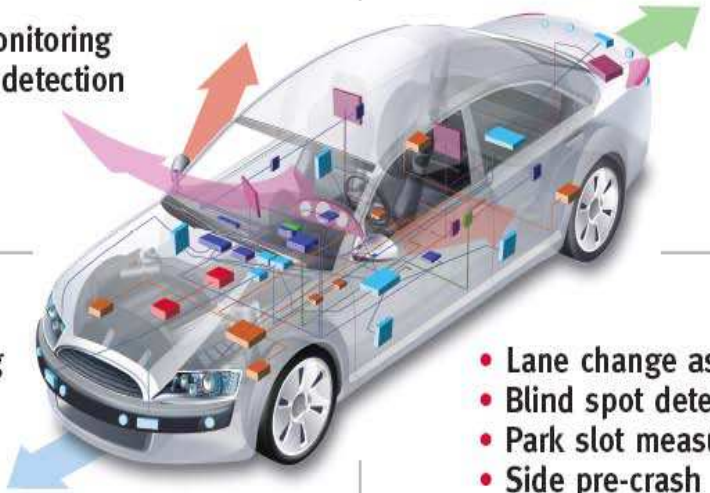
Outline

- Design trends and issues
- Our approach
- IMAPCAR2 processor core design
- Application mapping
- Benchmark results

Requirements of In-Vehicle Vision Processors

- Dynamic back up aid
- Following distance warning
- Parking assist

- Eye gaze monitoring
- Drowsiness detection
- PODS



- **High beam assist**
- Lane departure warning
- Pedestrian protection
- Traffic sign recognition
- Pre-crash
- Stop & Go

- Lane change assist
- Blind spot detection
- Park slot measurement
- Side pre-crash

High Performance

Real-time execution

High Flexibility

Application variety

Power Efficiency

Low price (small die size)
Easy cooling (<3W)

Existing Approaches and Design Trends

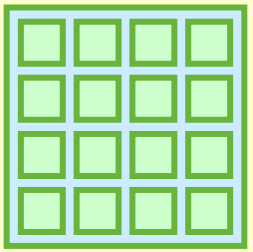
● ***Today***

- ✓ Programmable solution
- ✓ Without fan (< 3W)

DSPx1-x4
+
SIMD extensions

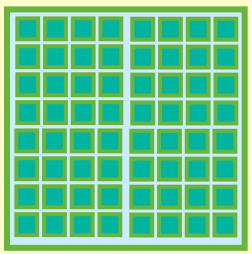
✓ **Lack of performance**

● ***Increase number of cores(MIMD)***



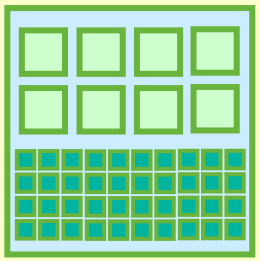
✓ **More general purpose, however, more expensive**

● ***Enhance the SIMD part***



✓ **Cheaper, however, limited applicability**

● ***Heterogeneous multi/many-cores***

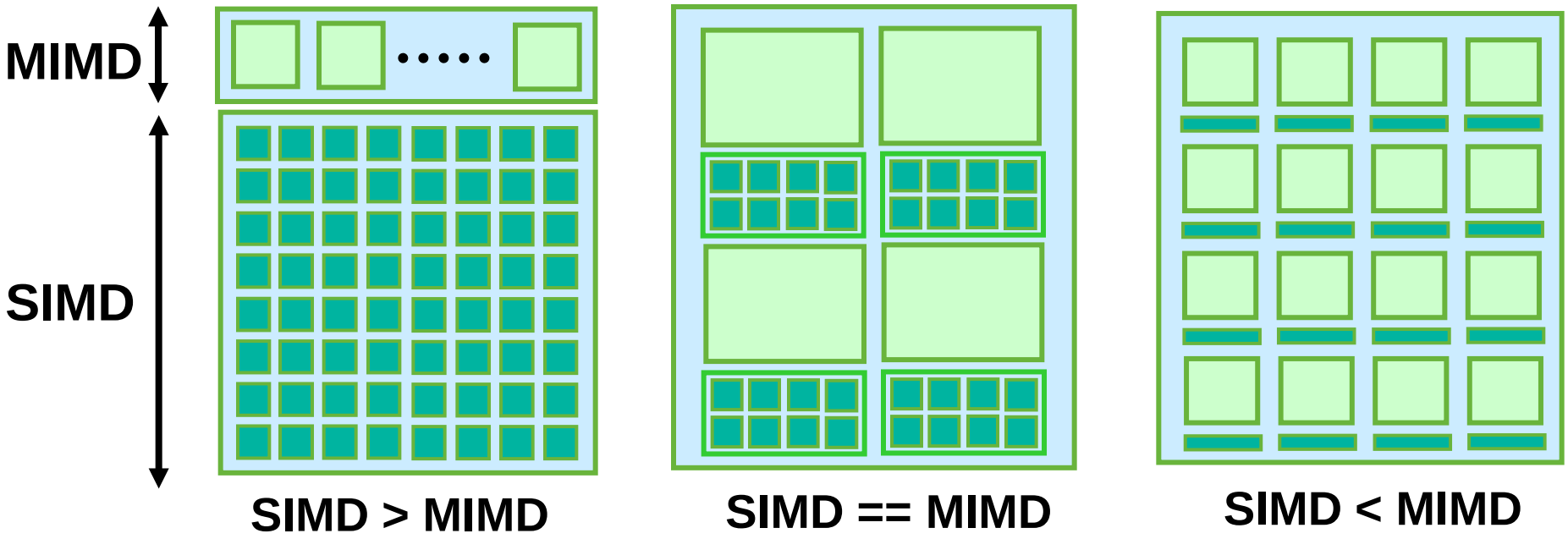


✓ **Having both features**

Recent “SIMD+MIMD” Trends

SIMD+MIMD becoming commonplace

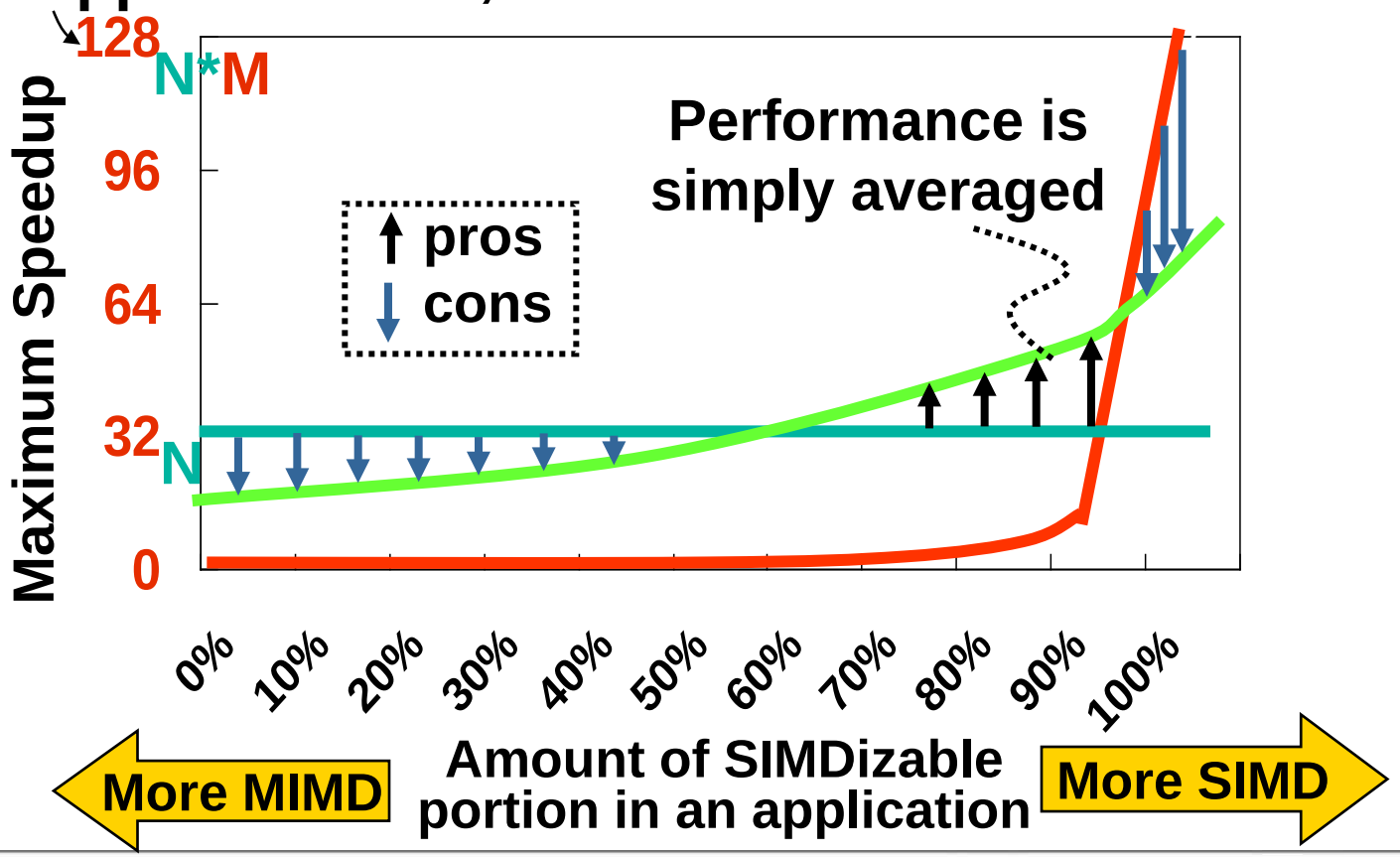
- Adding “high to medium” SIMD accelerator to multi-cores
- “*SIMD instruction*” added multi/many cores

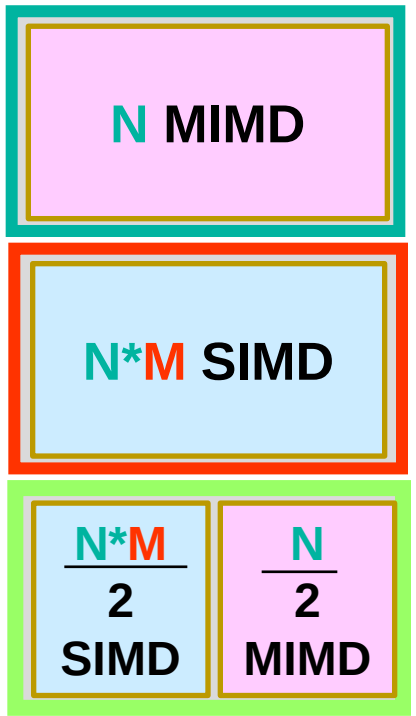


Pros and cons of “MIMD + SIMD”

- Suppose the limited (die) space fits *N* MIMD cores
- In case of *1* MIMD ≒ *M* SIMD, also fits *N*M* SIMD cores
- The same area will also fit *N/2* MIMD + *N*M/2* SIMD cores

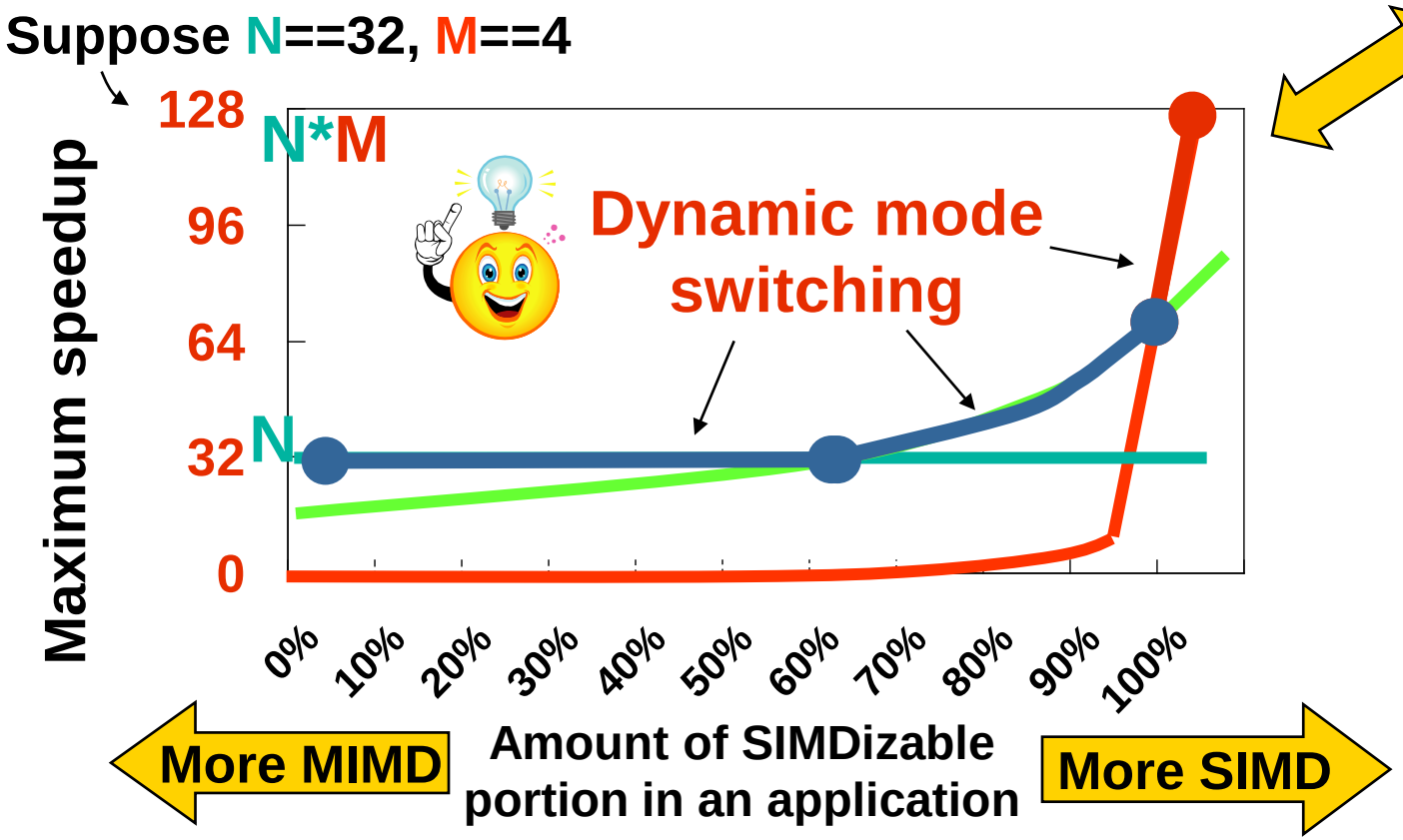
Suppose N==32, M==4

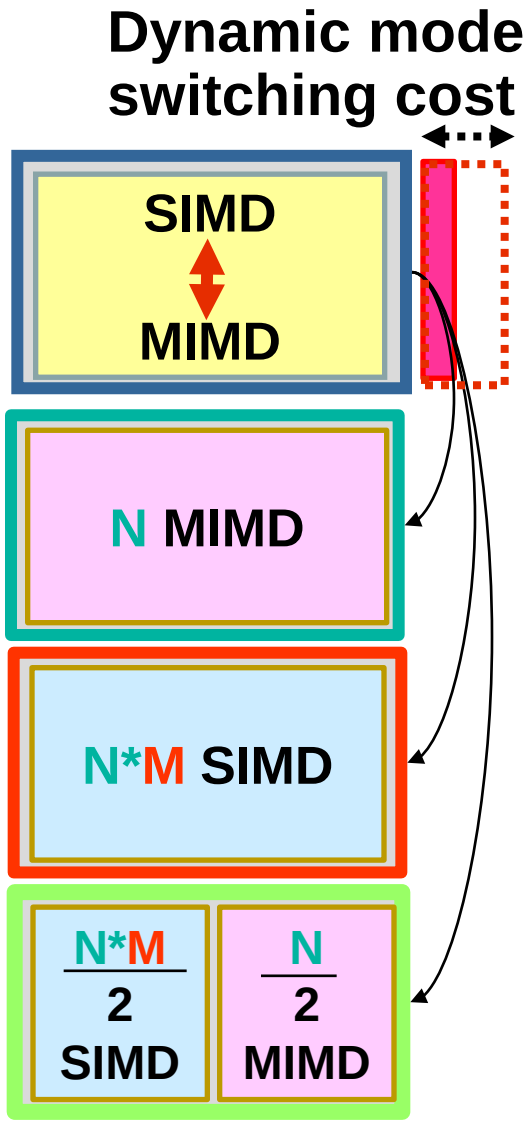




Our Approach

- Start from a highly parallel SIMD configuration
- Perform mode switching to take all the best !
- How to lower the mode switching cost ?





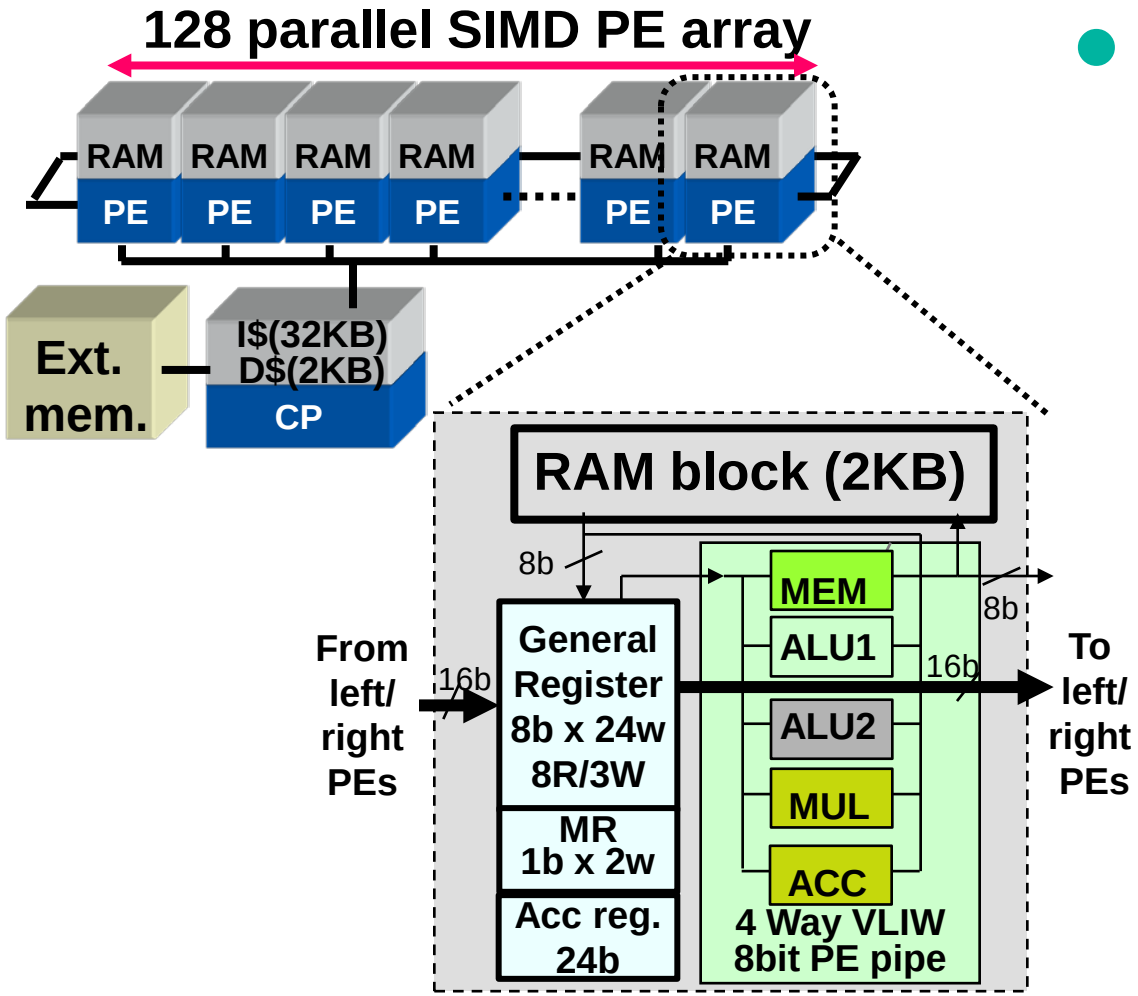
Outline

- Design trends and issues
- Our approach
- IMAPCAR2 processor core design
  - Morphing multiple SIMD PE into one MIMD PU
  - HW resource reuse strategy
- Application mapping
- Benchmark results

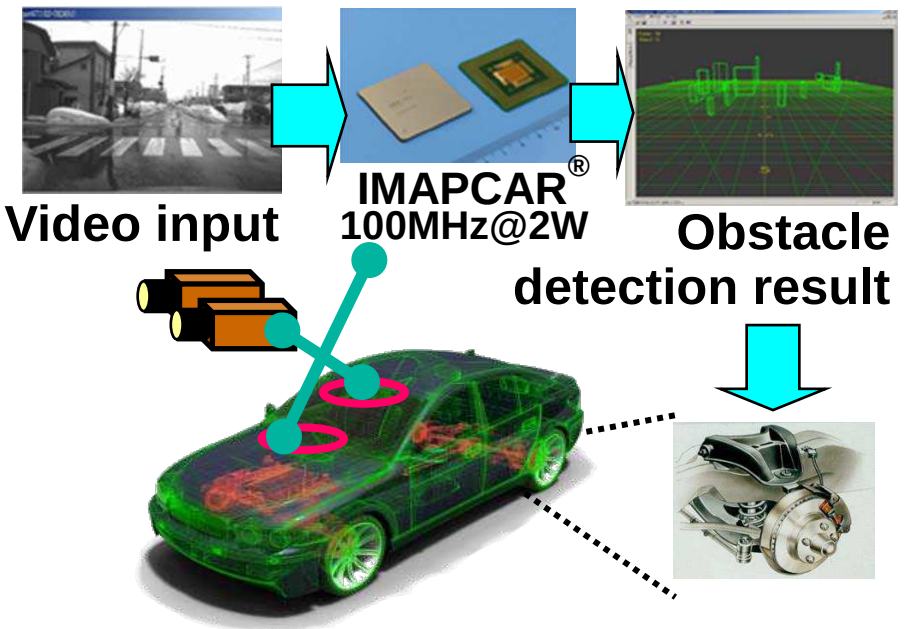
Our Previous Design : IMAPCAR

- **IMAP architecture (ISCA'05)**

IMAP: Integrated Memory Array Processor



- **IMAPCAR based products**
  - **In-vehicle safety systems (TOYOTA Lexus, etc.)**
  - **RoboCar vision system**
    http://www.zmp.co.jp/e_home.html



CP: Control Processor, I$: Instruction Cache, D$: Data Cache,
PE: SIMD Processing Element, Ext. mem.: External memory

IMAPCAR2 (XC Core) Design Goals

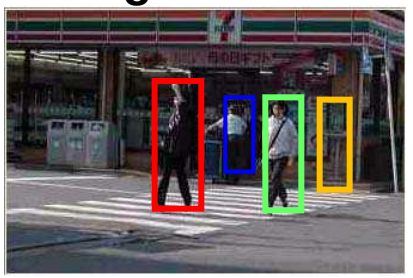
Fully adapt to the SIMD/MIMD nature of image recognition apps.

2D pixel data → Pixel processing → 2D feature data → Feature vector extraction → 1D vectors → Classifier → Decision: Pedestrian yes / no

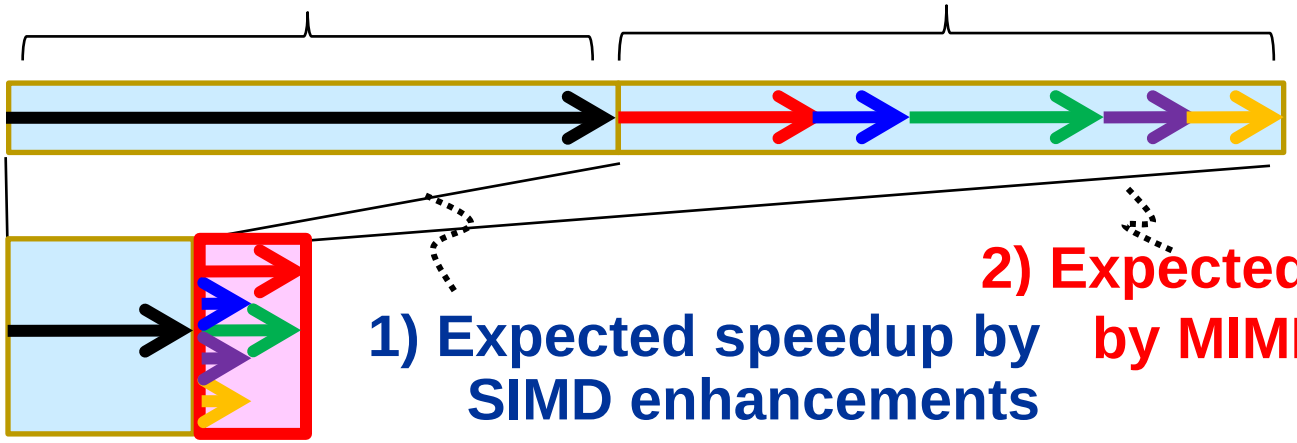
Uniform data parallel processing (Suitable in SIMD)
Ex. Candidate Search

Non-uniform ROI processing/ (Suitable in MIMD)
Ex. Candidate verification

ROI: Region of Interest



IMAPCAR
(SIMD only)



IMAPCAR2 design goals

1) Expected speedup by SIMD enhancements

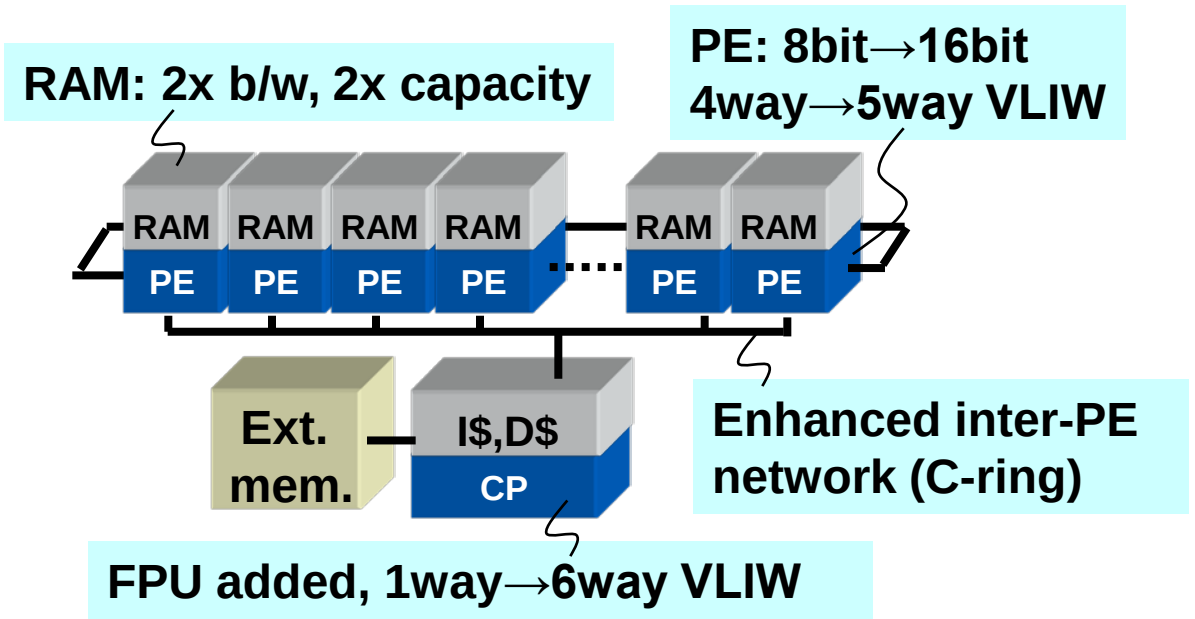
2) Expected speedup by MIMD support

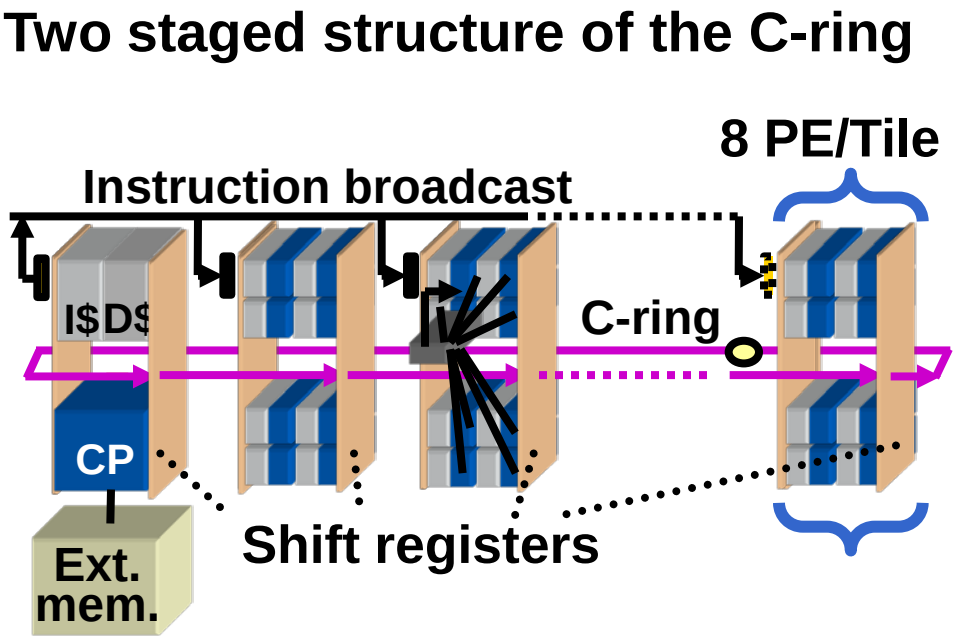
XC Core : SIMD Performance Enhancements

XC core: e**X**tensible **C**omputing core

SIMD enhancements (vs. IMAPCAR)

- 16bit (2x) architecture, 32bit/PE (2x) on-chip memory bandwidth, etc.
- 8PE/tile, circuit-switched 2-staged inter-PE ring network (C-ring)
- Power and area are maintained by process shrink (130nm→90nm)

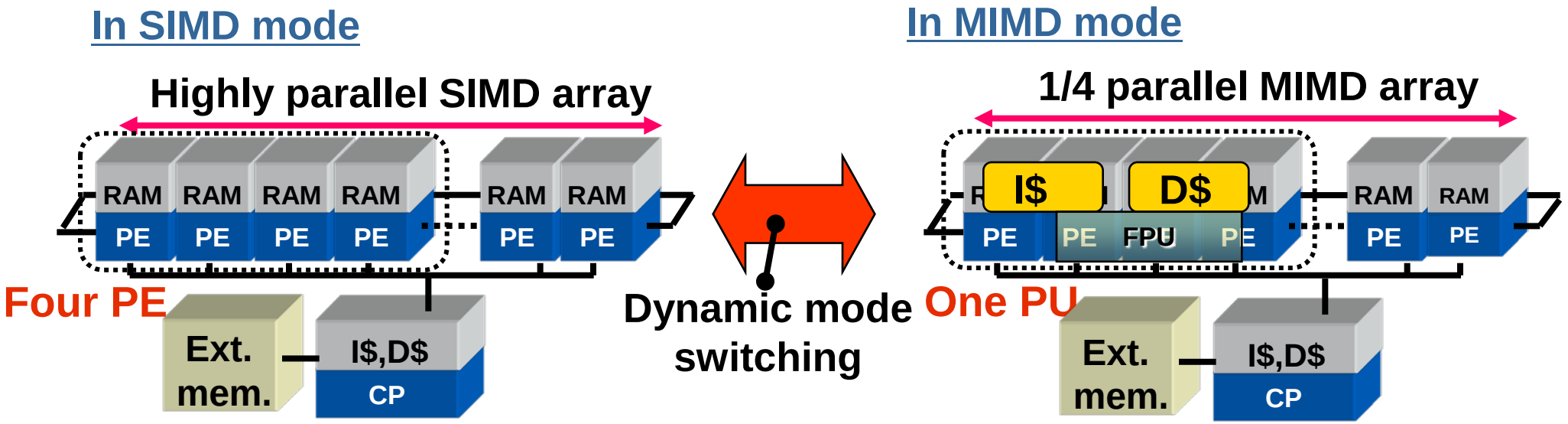




XC Core : MIMD Support

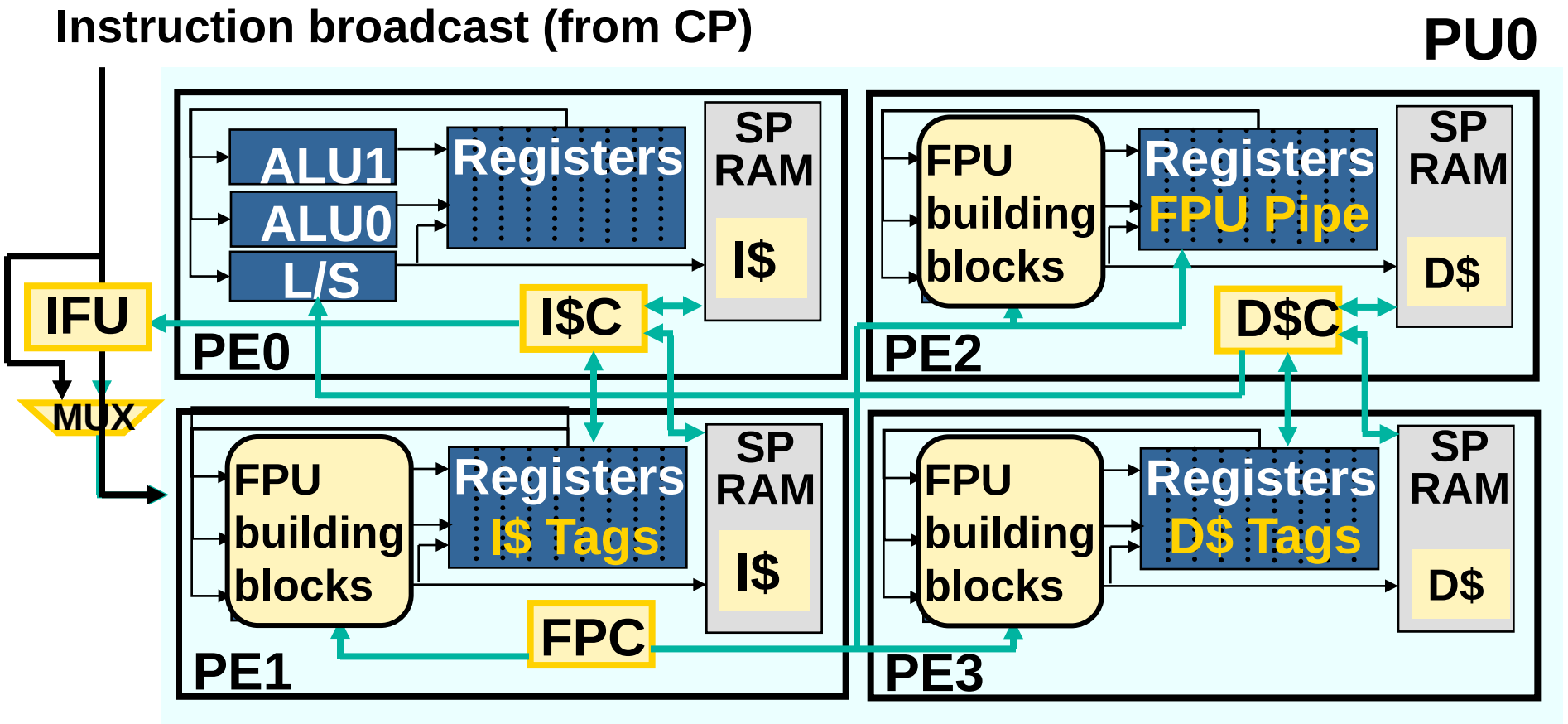
Supporting MIMD mode in very low cost

- Trading off between performance and flexibility
  - Reuse HW resources of four SIMD PEs to form one MIMD PU



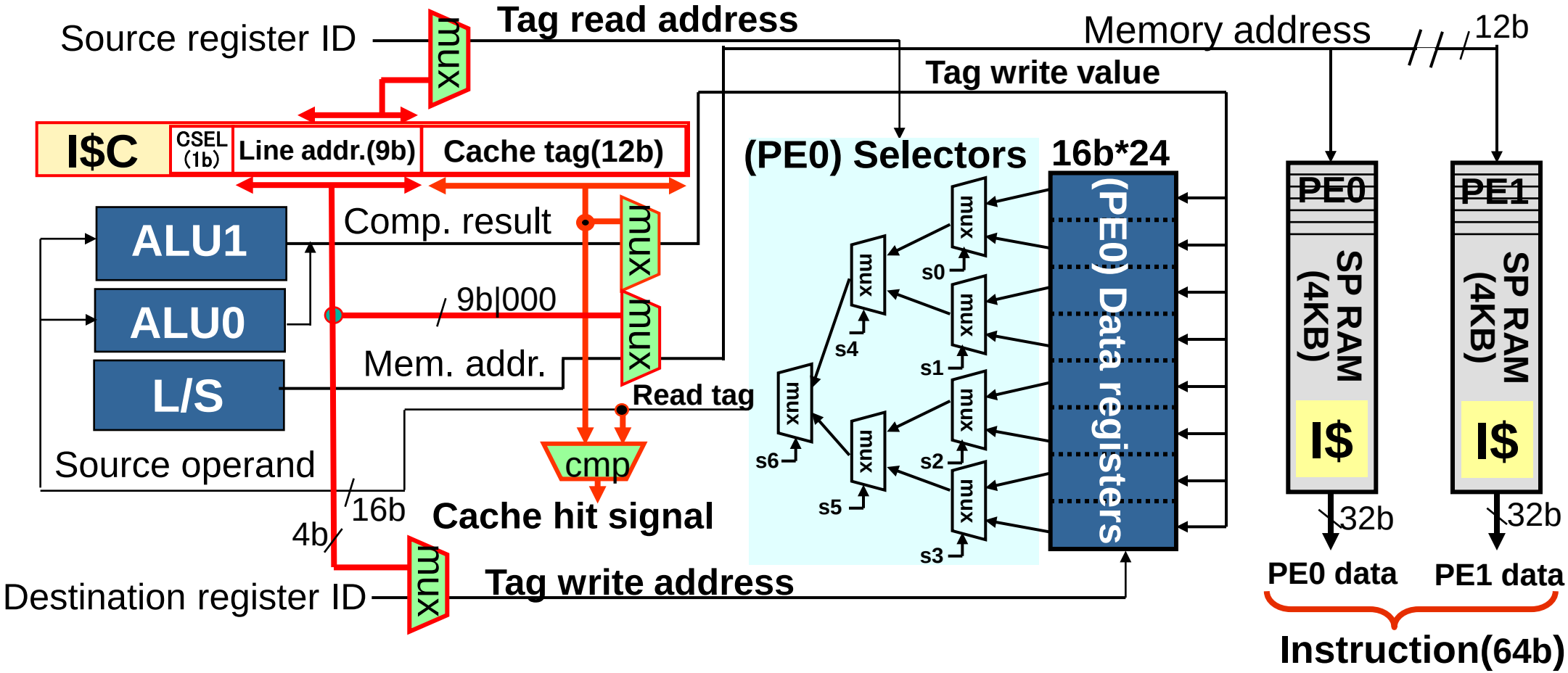
PU: MIMD Processing Unit
FPU: Floating Point Unit

Detail HW Resource Reuse Strategy



SP RAM: Scratch-pad RAM, **IFU:** Instruction Fetch Unit, **FPC:** Floating Point Control
I$C: Instruction cache Control, **D$C:** Data cache Control

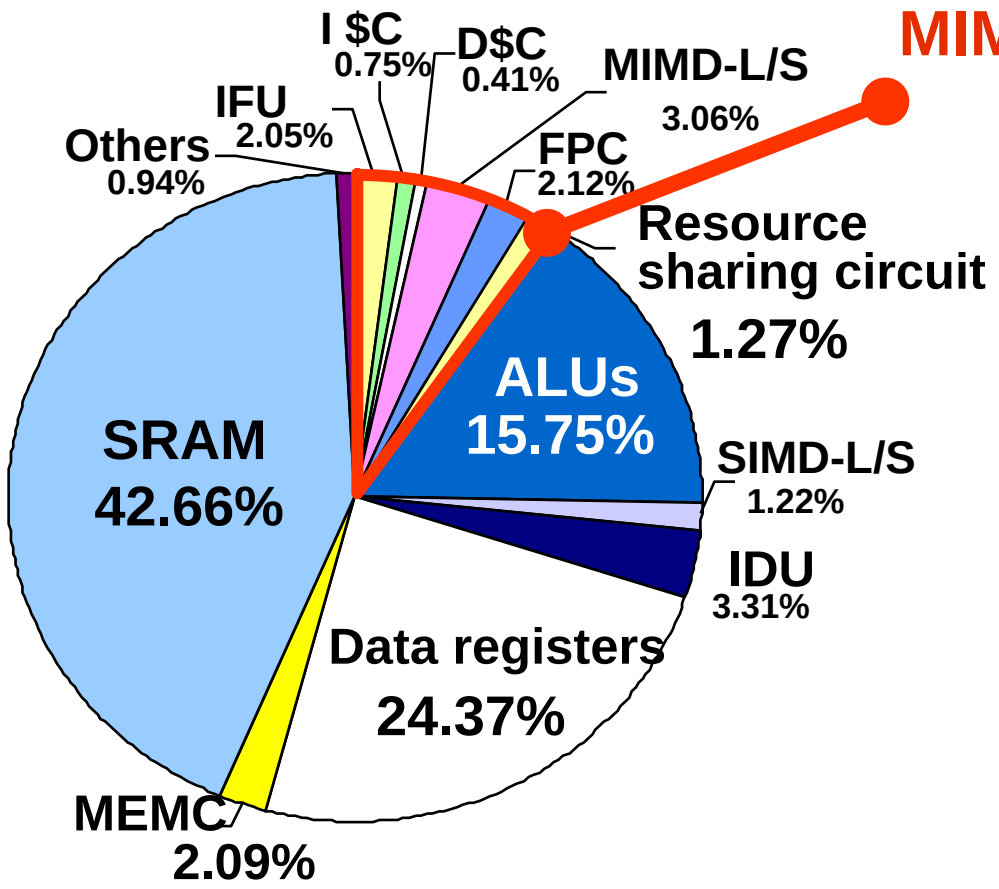
Configuring PU I$ using PE HW Resources



- SP RAM of 2 PEs and data registers of 1 PE are fully re-used
- Major overheads : several 2to1 MUXs, small I$C logics

Overall HW Overhead for Mode Switching

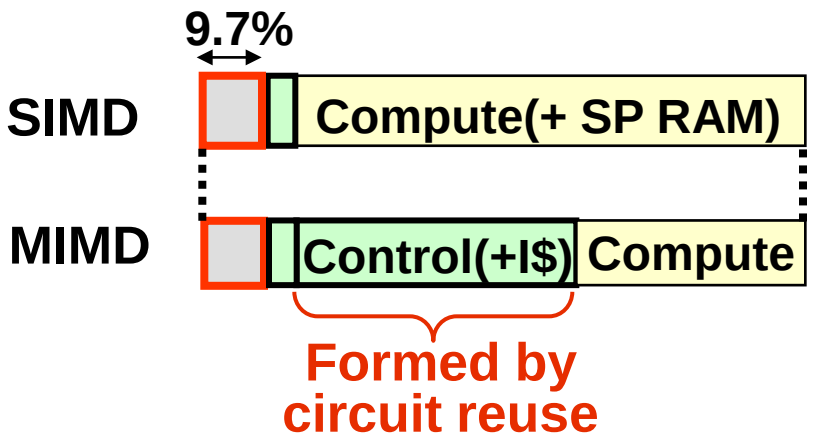
Gate count distribution of a 8 PE tile



MIMD Supporting Overheads

Summary (Logic + RAM)

MIMD core only logic	Resource sharing logic	Total
8.4%	1.3%	9.7%



Synthesis tool: Synopsys Design Compiler
Library: NEC 90nm process
Target operation frequency: 150MHz

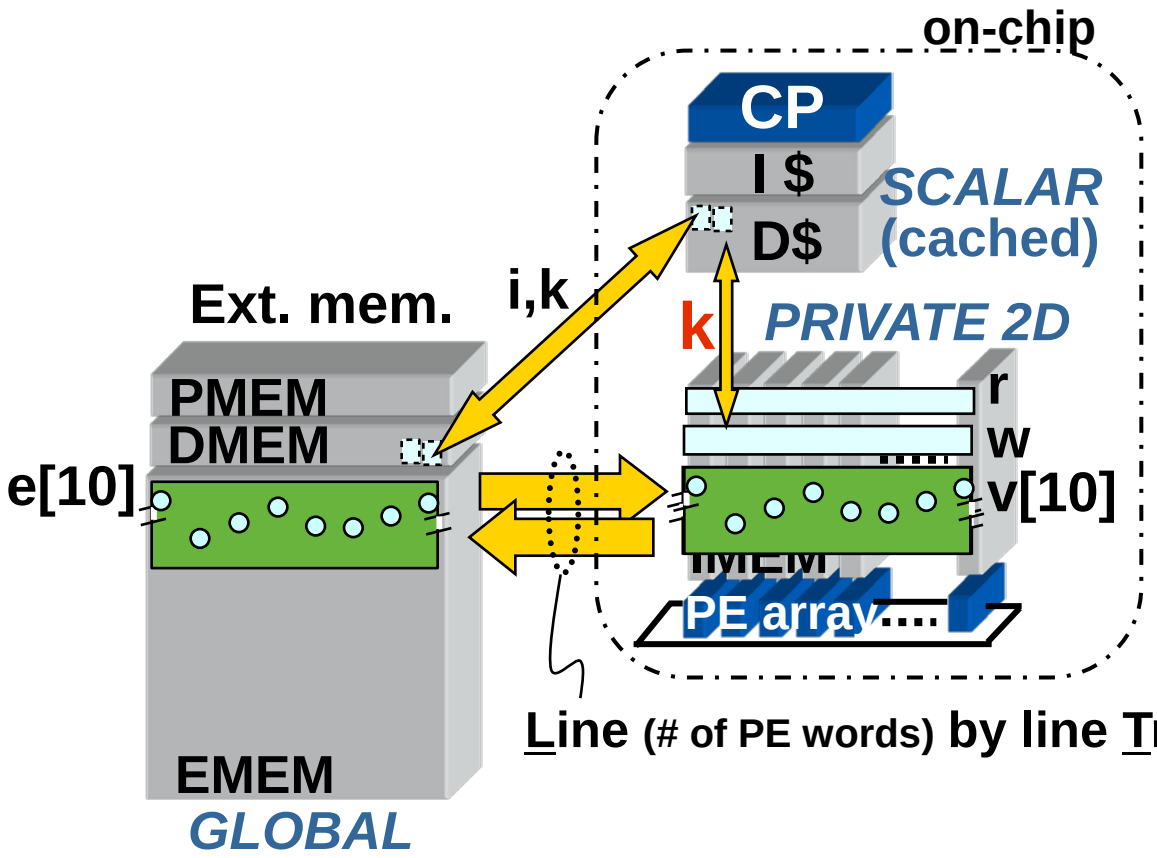
Outline

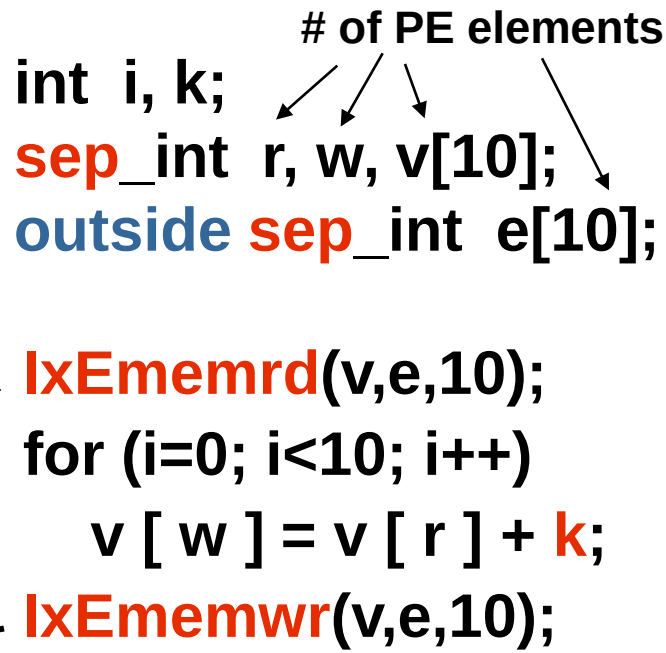
- Design trends and issues
- Our approach
- IMAPCAR2 processor core design
- Application mapping
  - SIMD programming model
  - MIMD programming model
- Benchmark results

Programming the SIMD Mode

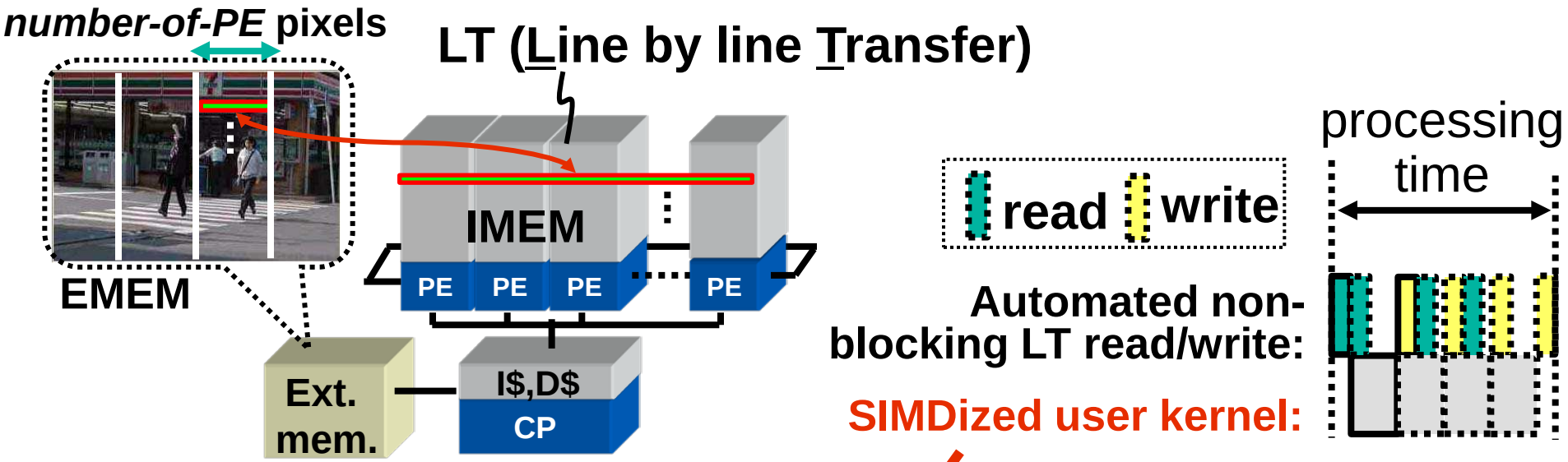
One dimensional C (1DC)

- *GLOBAL*, *SCALAR*, and *PRIVATE 2D* memory spaces
- ANSI C + straightforward vector data type enhancement
- Explicit user driven "*GLOBAL*" ⇔ "*PRIVATE 2D*" line data transfer



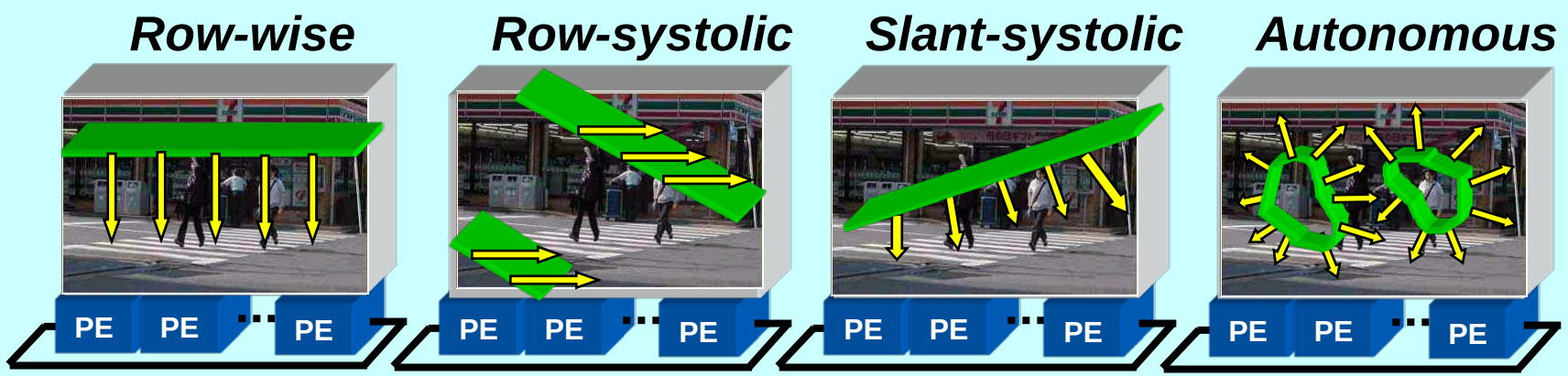


Automated LT + SIMD Kernels



SIMDizing methods: traverse the IMEM space for data analyses

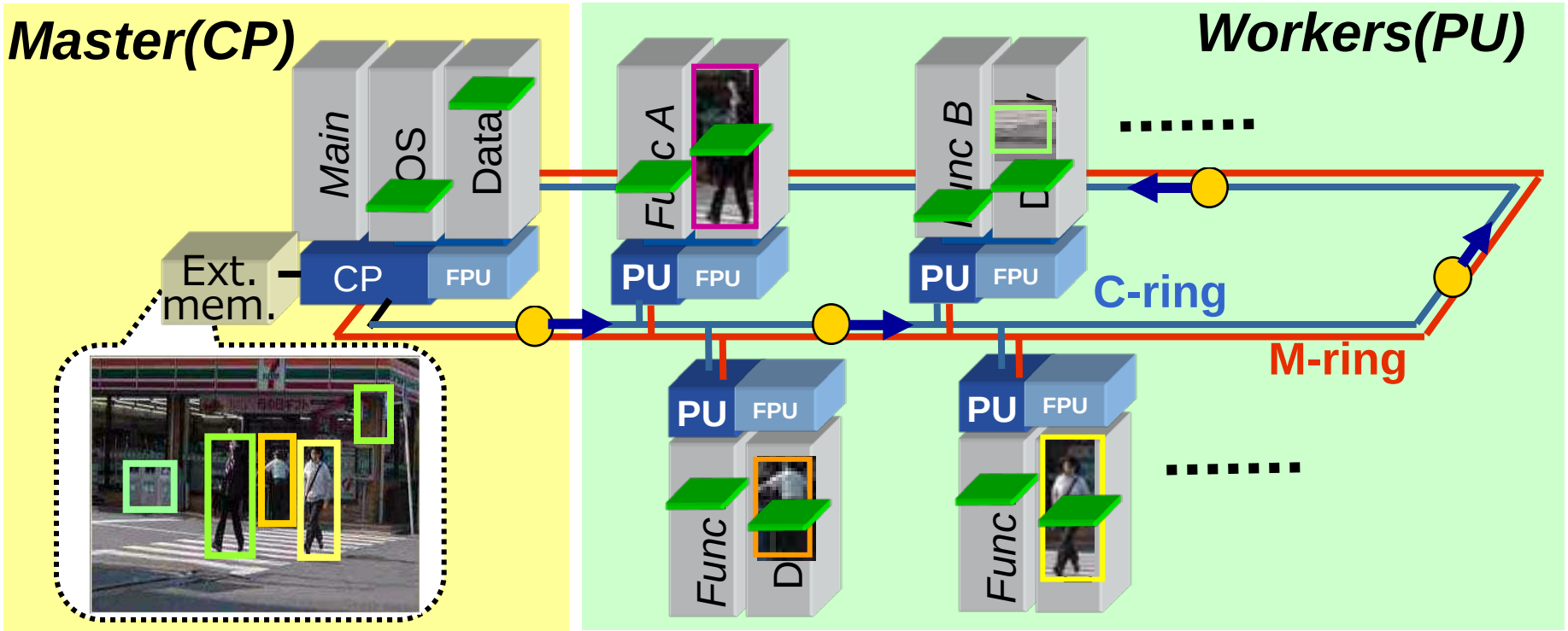
: Pixels processed in a single operation



Programming the MIMD Mode

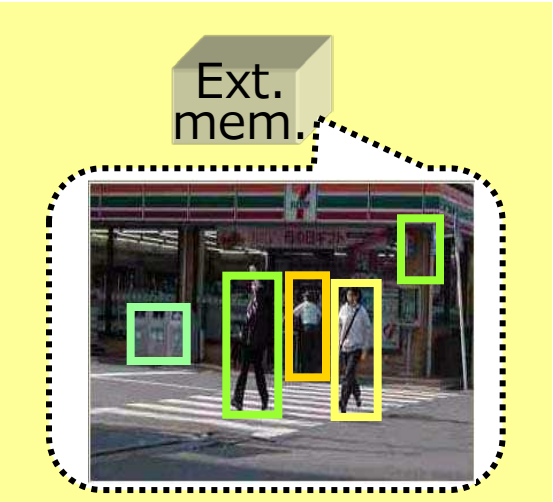
HW support of master-worker style task allocation

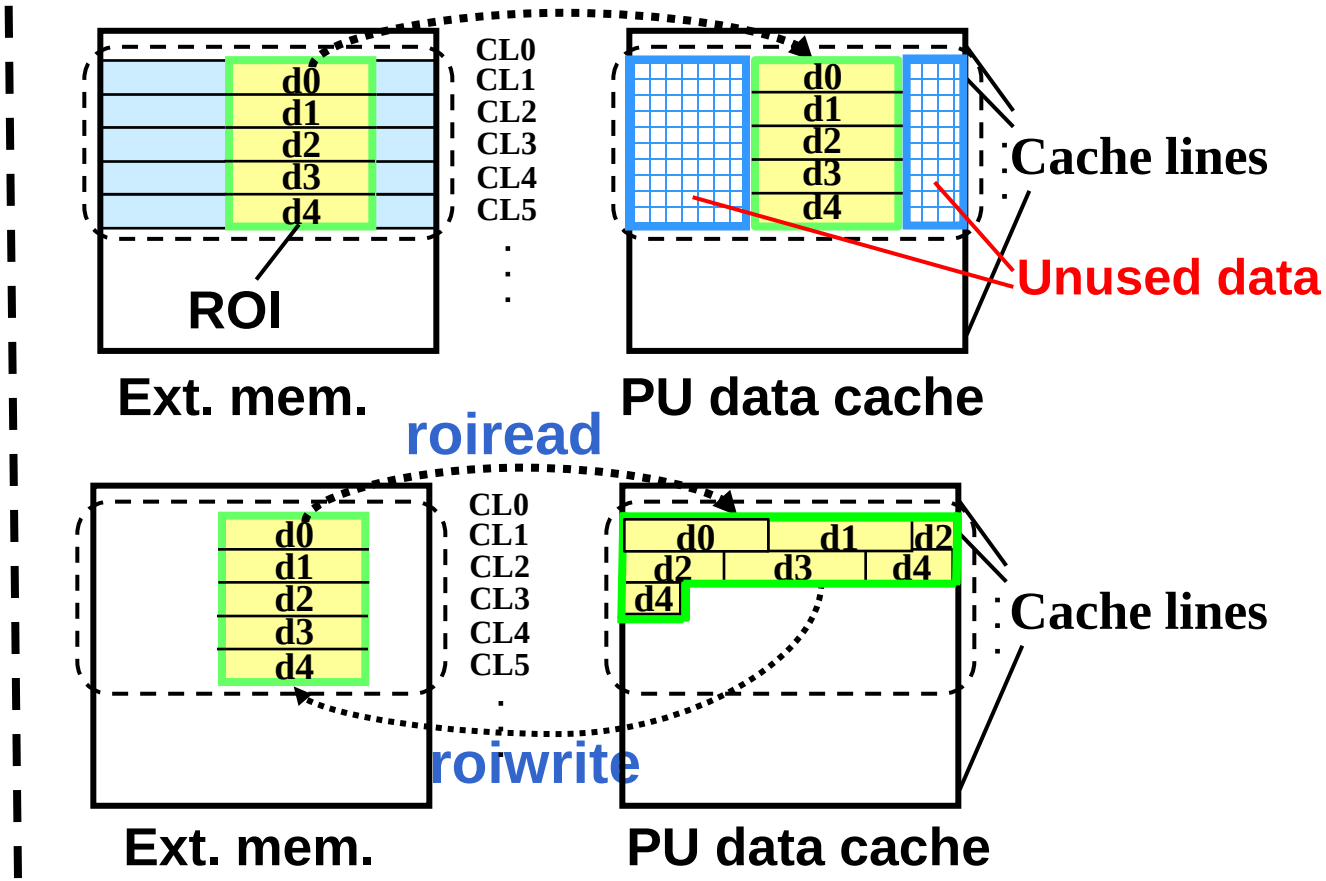
- C-ring for message communications, M-ring for ROI data transfer
  - One-to-one/ one-to-any/ interruption task messages distribution by C-ring
  - HW supported ROI data transfer using M-ring



Memory Traffic Reduction in MP Mode

HW supported ROI data transfer





HW supported instruction/data broadcast to each PU at startup

Cache-bypassed access for applications with little data locality

Programming Support Summary

In SIMD mode

ANSI C with long vector data extension
Fixed point configuration
Mem. acc. flexibility within private 2D space

In MIMD mode

ANSI C
Floating point operations
Thread programming

(PE/PU number independent) Libraries

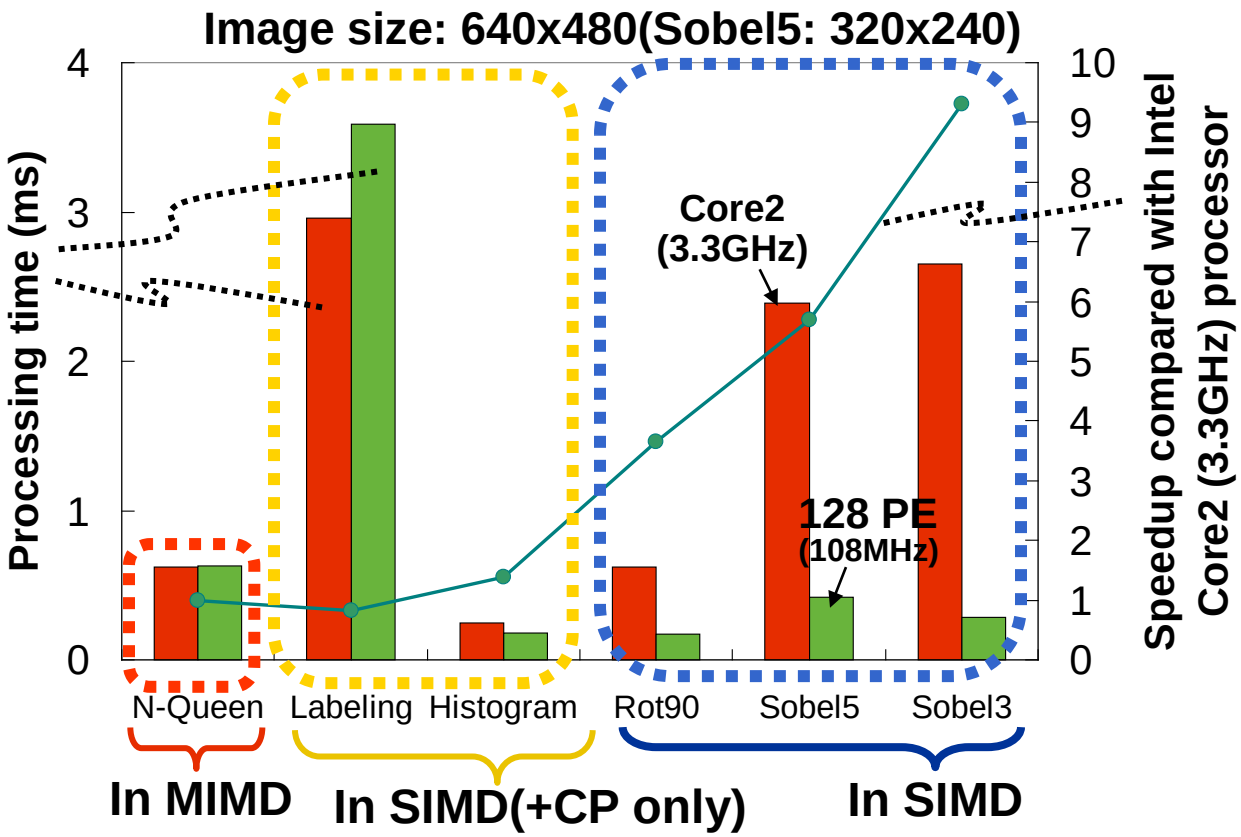
- **Standard SIMD library (Data remapping etc,...)**
- **Standard MIMD library (message system, thread basics, etc…)**
- **Basic image processing library**
  - **MATLAB, OpenCV compatible image processing library (subset)**
  - **Pthreads library**
  - **T-Kernel Operating System library**
  - **Eclipse plug-in for 1DC source level debugging**
  - **1DC class library (enable running 1DC code on desktop PCs using SSE*)**

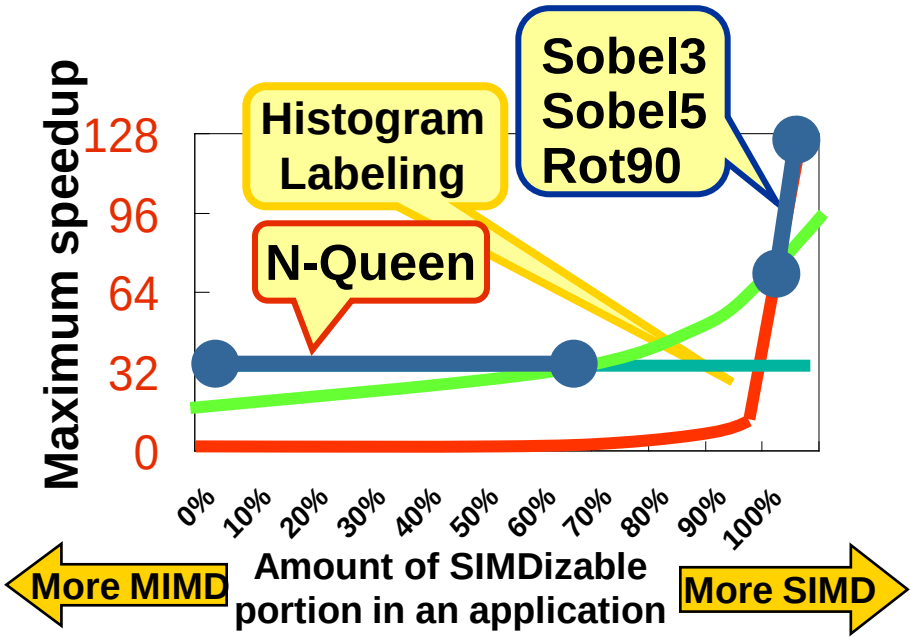
Outline

- Design trends and issues
- Our approach
- IMAPCAR2 processor core design
- Application mapping
- Benchmark results
  - SIMD and MIMD Performance
  - SIMD/MIMD performance

SIMD and MIMD Performance (vs. GPP)

- 1DC compiler code vs. C compiler code (GPP: gcc –O3, 1DC: cc1dc –O)
- Up to 9x speedup vs. GPP@3.3GHz, in SIMD mode using 128PE
- 32PU@108MHz in MIMD mode is comparative with a GPP@3.3GHz

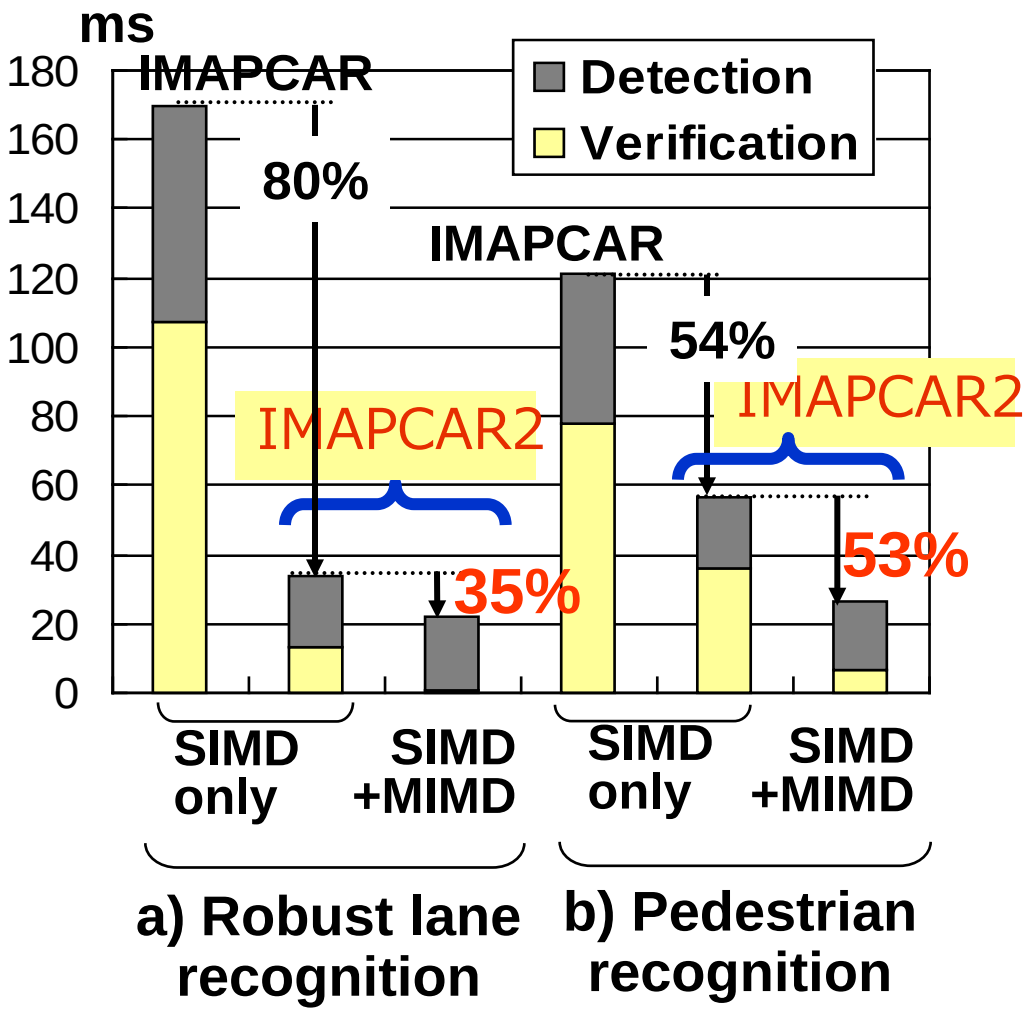




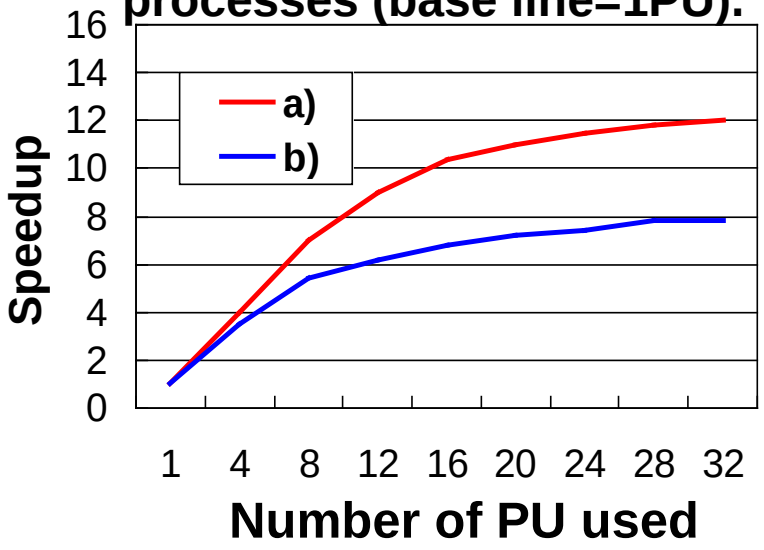
GPP: General Purpose Processor

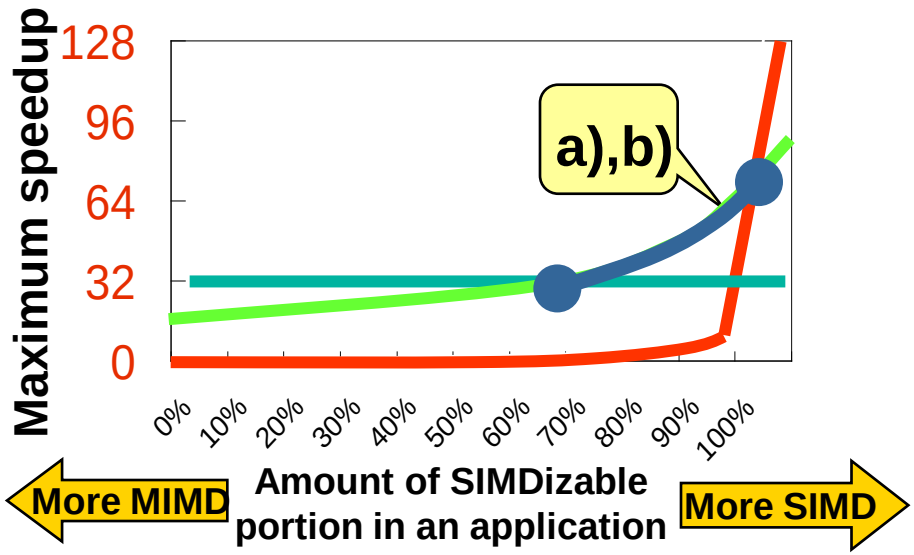
SIMD / MIMD Performance (vs. IMAPCAR)

- Up to 5x speedup in SIMD mode
- Further 2x speedup by mode switching



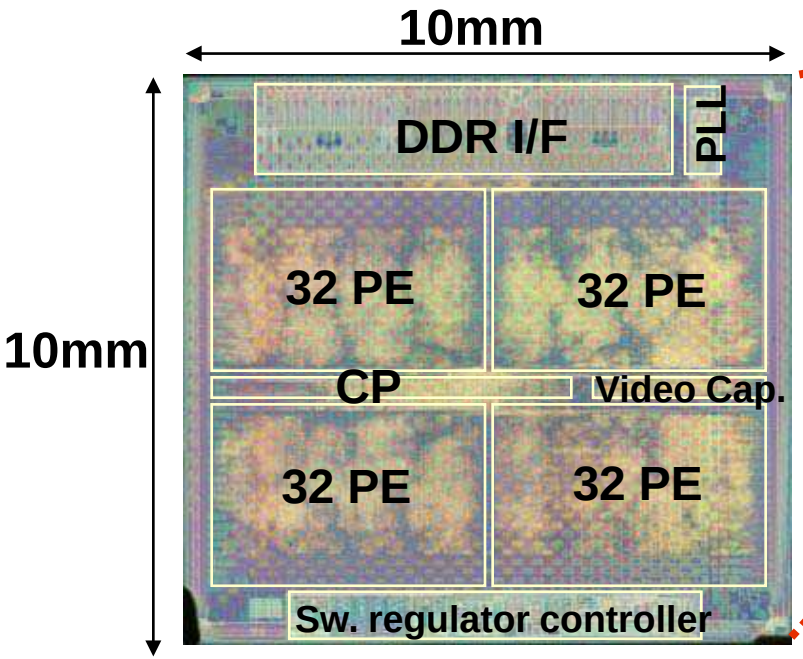
Performance scaling of verification processes (base line=1PU).





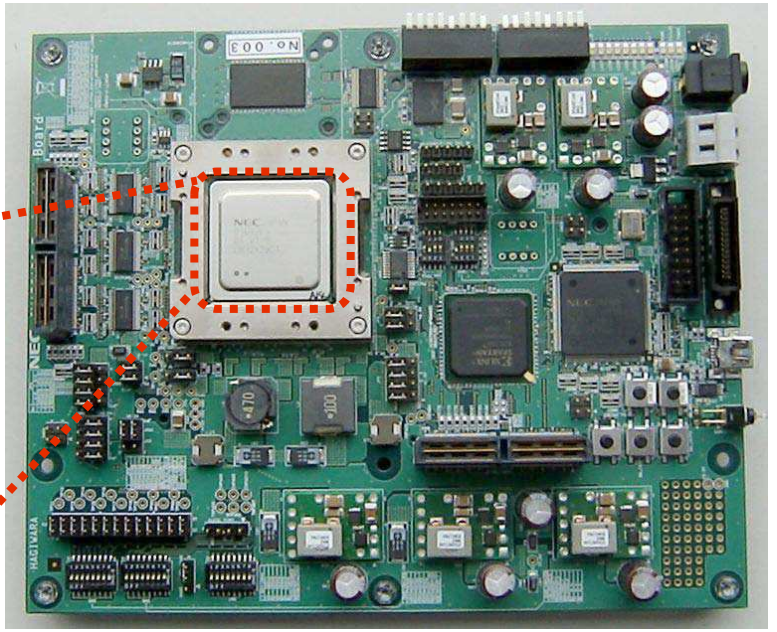
128PE ⇔ 64PE+16PU ⇔ 32PU Prototype

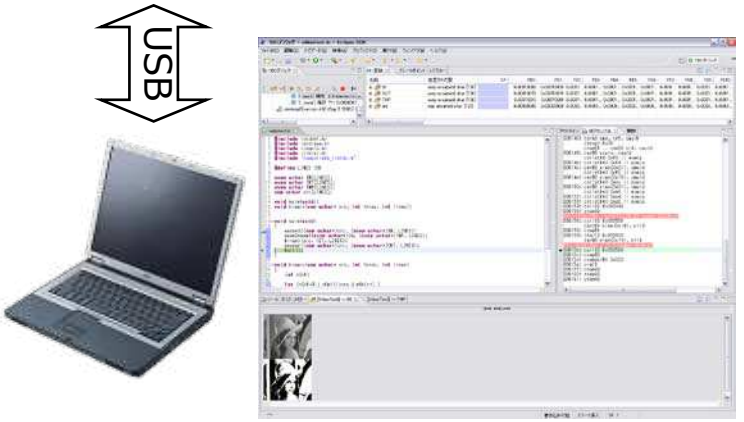
128PE/64PE+16PU/32PU
138 GOPS (in SIMD)
21GOPS+3.6GFLOPS (in MIMD)



90nm, CMOS Cu 7 Layers
529 pin FCBGA, 108MHz

Evaluation board with USB interface

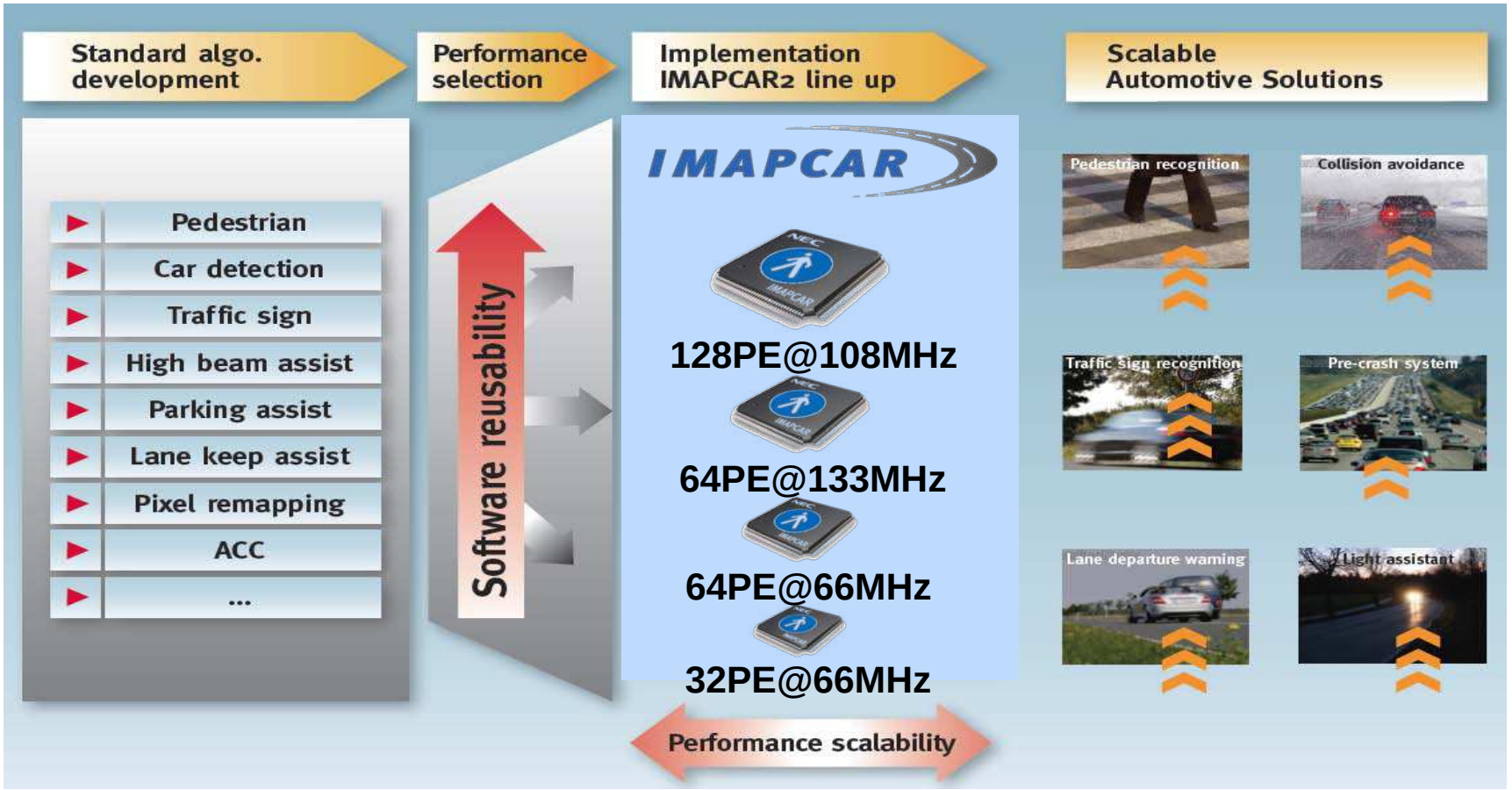




Eclipse 1DC plugin

Platform Approach

- 32PE/8PU, 64PE/16PU, and 128PE/32PU line-ups
- Software reuse from low- to high-end products



Summary

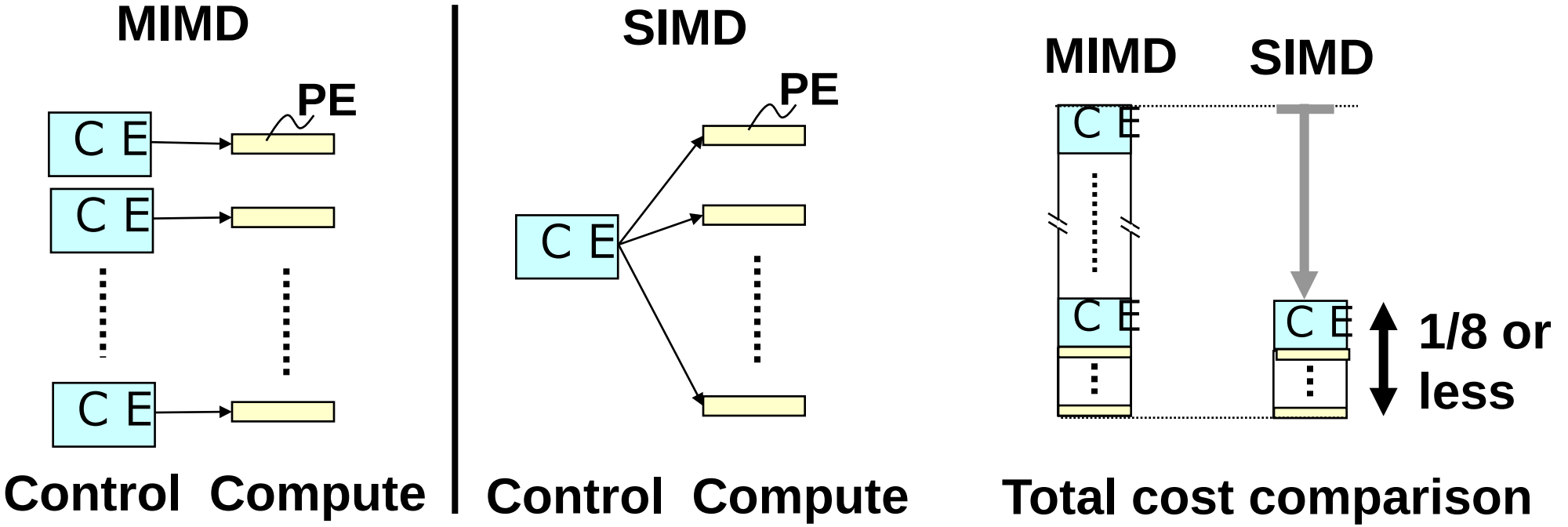
- A *tilable* SIMD/MIMD core (potentially superior to SIMD+MIMD)
- Only 10% HW overhead compared with a pure highly parallel SIMD
- Low-cost achieved by trading-off parallelism with flexibility
- Fully adapt to the SIMD/MIMD nature of the target applications
- Envisioned SIMDizing & MIMDizing methods reduce design efforts
- Platform approach facilitates performance scalability & code reuse

Empowered by Innovation

Appendix

SIMD vs. MIMD Architectures

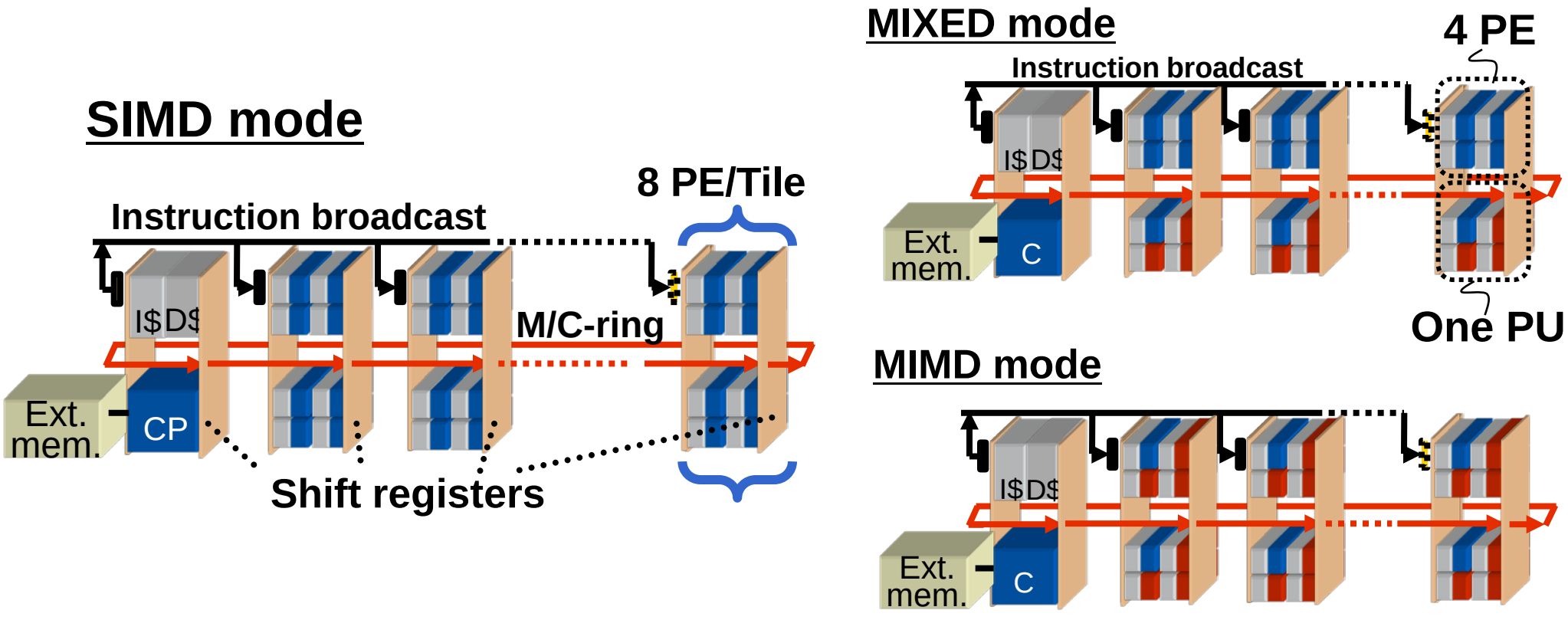
PE: Processing Element
CE: Control Element



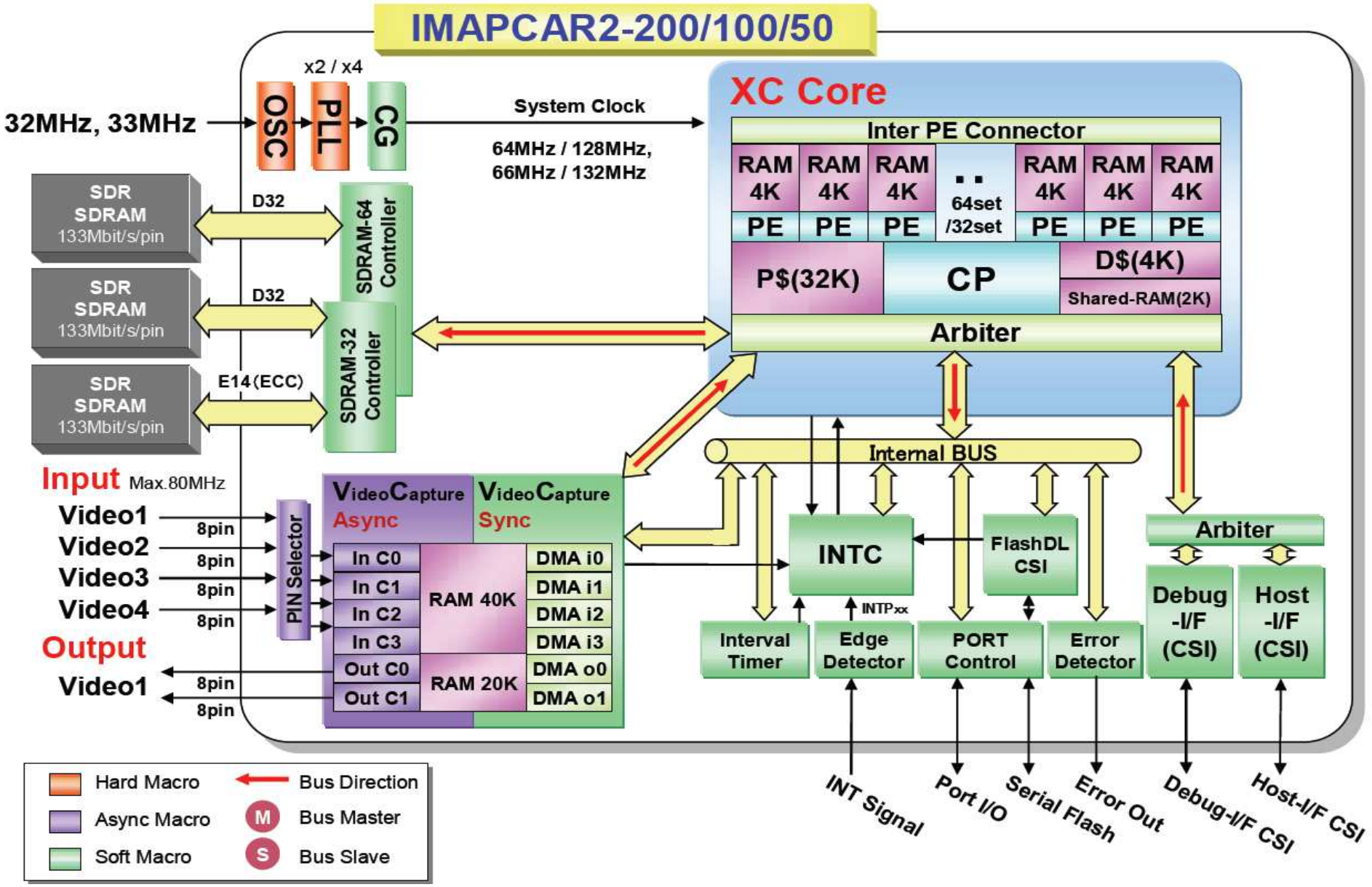
- MIMD is general purpose, however, more expensive
- SIMD is cheaper, however, limited applicability

Execution Modes and Ring Networks

- "8PE" or "4PE+1PU" or "2PU" per Tile
- Tiles are connected by light-weight circuit-switched ring network
- M-ring for memory access
- C-ring for inter PE/PU communication

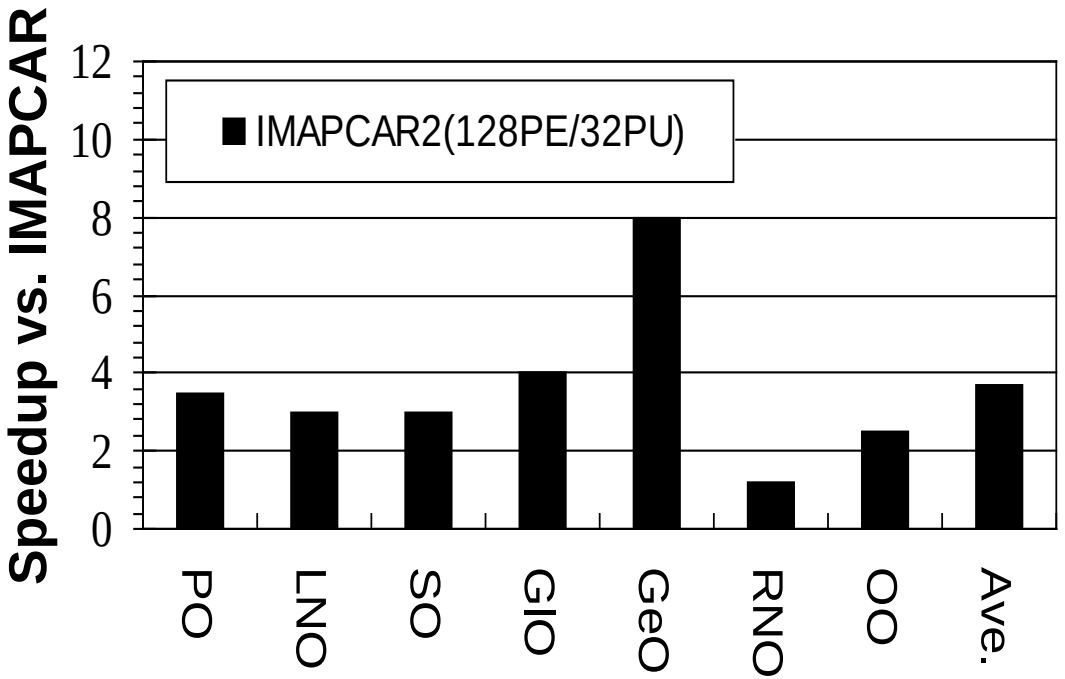


IMAPCAR2 Series Processor Block Diagram



SIMD Performance (vs. IMAPCAR)

- Use kernels representing typical memory access patterns of image tasks
- 8x speedup for GeO (Geometrical Operation) tasks by using the C-ring
- 4x speedup in average for various memory access patterns



IMAPCAR: 128PE@100MHz

IMAPCAR2-300: 128PE@108MHz

Task category	Used kernel task
PO	Color format transform
LNO	3x3 average 2-D filter
SO	Histogram calculation
GlO	FFT (24b fixed point)
GeO	90 degree rotation
RNO	Distance transform
OO	Connected component Labeling

PO: Point Operation, LNO: Local Neighborhood Operation
SO: Statistical Operation, GlO: Global Operation
GeO: Geometrical Operation
RNO: Recursive Neighborhood Operation, OO: Object Operation