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

Shorin Kyo\textsuperscript{1}, Shouhei Nomoto\textsuperscript{1}, Takuya Koga\textsuperscript{2}
Hanno Lieske\textsuperscript{1}, Shinichiro Okazaki\textsuperscript{1}

NEC Corporation\textsuperscript{1}
NEC Electronics Corporation\textsuperscript{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

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

- Eye gaze monitoring
- Drowsiness detection
- PODS

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

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)
  - ⚠ Lack of performance
  
  - DSPx1-x4
  + SIMD extensions

- **Enhance the SIMD part**
  - ⚠ Cheaper, however, limited applicability

- **Increase number of cores (MIMD)**
  - ✔ More general purpose, however, more expensive

- **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

<table>
<thead>
<tr>
<th>MIMD</th>
<th>SIMD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>SIMD &gt; MIMD</td>
<td>SIMD == MIMD</td>
</tr>
<tr>
<td>SIMD &lt; MIMD</td>
<td></td>
</tr>
</tbody>
</table>
Pros and cons of “MIMD + SIMD”

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

Suppose $N=32$, $M=4$

Performance is simply averaged

Maximum Speedup

More MIMD

Amount of SIMDizable portion in an application

More SIMD

Pros and cons
Our Approach

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

Suppose $N=32$, $M=4$

Dynamic mode switching cost

Maximum speedup

Amount of SIMDizable portion in an application

More MIMD  More SIMD

More MIMD  More SIMD
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)**

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

---

**IMAP: Integrated Memory Array Processor**

128 parallel SIMD PE array

**RAM block (2KB)**

- General Register: 8b x 24w, 8R/3W
- MR: 1b x 2w
- Acc reg.: 24b
- MEM
- ALU1
- ALU2
- MUL
- ACC
- 4 Way VLIW 8bit PE pipe

**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.

**Pixel processing**
- 2D pixel data

**Feature vector extraction**
- 2D feature data

**Classifier**
- 1D vectors

**Decision**
- Pedestrian yes / no

**ROI:** Region of Interest
- Uniform data parallel processing (Suitable in SIMD)
- Non-uniform ROI processing/ (Suitable in MIMD)

**Ex. Candidate Search**
- Ex. Candidate verification

**IMAPCAR**
- (SIMD only)

**IMAPCAR2**
- design goals

1) Expected speedup by SIMD enhancements
2) Expected speedup by MIMD support
XC Core: SIMD Performance Enhancements

**SIMD enhancements (vs. IMAPCAR)**

- 16-bit (2x) architecture, 32-bit/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)

**Diagram:**
- **RAM:** 2x b/w, 2x capacity
- **PE:** 8-bit → 16-bit, 4-way → 5-way VLIW
- **FPU added, 1-way → 6-way VLIW**
- **Instruction broadcast**
- **Enhanced inter-PE network (C-ring)**
- **Shift registers**
- **Two staged structure of the C-ring**

**XC core:** eXtensible Computing core
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

**In SIMD mode**

Highly parallel SIMD array

**In MIMD mode**

1/4 parallel MIMD array

Dynamic mode switching

Four PE

Ext. mem.

I$,D$

CP

I$,D$

CP

PE

PE

PE

PE

PE

PE

PE

PE

PE

PE

PE

PE

RU: MIMD Processing Unit
FPU: Floating Point Unit
**Detail HW Resource Reuse Strategy**

**Instructions broadcast (from CP)**

- **IFU**: Instruction Fetch Unit
- **MUX**: MUX switch
- **ALU1** and **ALU0**: Arithmetic Logic Units
- **L/S Registers**: Local Store Registers
- **SP RAM**: Scratch-pad RAM
- **I$C**: Instruction cache Control
- **I$ Tags**: Instruction tags
- **D$C**: Data cache Control
- **D$ Tags**: Data tags
- **FPC**: Floating Point Control
- **FPU**: Floating Point Unit
- **SP RAM**: Scratch-pad RAM
- **I$$**: Instruction cache
- **D$$**: Data cache
- **FPU Pipe**: Floating Point Pipe
- **PE0**, **PE1**, **PE2**, **PE3**: Processing Elements

**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)

<table>
<thead>
<tr>
<th></th>
<th>MIMD core only logic</th>
<th>Resource sharing logic</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMD</td>
<td>8.4%</td>
<td>1.3%</td>
<td>9.7%</td>
</tr>
<tr>
<td>MIMD</td>
<td>9.7%</td>
<td>1.3%</td>
<td>9.7%</td>
</tr>
</tbody>
</table>

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

```c
int i, k;
sep_int r, w, v[10];
outside sep_int e[10];

lxEmemrd(v,e,10);
for (i=0; i<10; i++)
  v[w] = v[r] + k;
lxEmemwr(v,e,10);
```
Automated LT + SIMD Kernels

LT (Line by line Transfer)

Automated non-blocking LT read/write:
SIMDized user kernel:

SIMDizing methods: traverse the IMEM space for data analyses

Pixels processed in a single operation

Row-wise
Row-systolic
Slant-systolic
Autonomous
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

**Master (CP)**

**Workers (PU)**

- Task messages

- C-ring
- M-ring

Ext. mem.
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

![Graph showing performance comparison](image)

- Detection
- Verification

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

- **a)** Robust lane recognition
- **b)** Pedestrian recognition

**Maximum speedup**

- More MIMD
- Amount of SIMDizable portion in an application
- More SIMD
128PE $\leftrightarrow$ 64PE+16PU $\leftrightarrow$ 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

NEC
Appendix
SIMD vs. MIMD Architectures

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

PE: Processing Element
CE: Control Element

Total cost comparison: 1/8 or less
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

**SIMD mode**

- Instruction broadcast
- 8 PE/Tile
- M/C-ring
- Shift registers

**MIXED mode**

- Instruction broadcast
- 4 PE

**MIMD mode**

- Instruction broadcast
- One PU
IMAPCAR2 Series Processor Block Diagram

- **System Clock**: 64MHz / 128MHz, 66MHz / 132MHz
- **Inter PE Connector**: RAM 4K, RAM 4K, RAM 4K, ..., RAM 4K, RAM 4K, RAM 4K
- **PC**: P$(32K)$, CP
- **Arbiter**: Shared-RAM(2K)
- **XC Core**: Inter PE Connector
- **SDRAM Controller**: SDRAM-32, SDRAM-64
- **SDRAM**: SDR SDRAM 133Mbit/s/spin
- **Input**: Video 1, Video 2, Video 3, Video 4
- **Output**: Video 1
- **Video Capture**: Async, Sync
- **EMAC**: 8pin, 8pin, 8pin, 8pin
- **TMR**: 8pin, 8pin
- **INTC**: Debug I/F (CSI), Host I/F (CSI)
- **Arbiter**: Debug I/F (CSI), Host I/F (CSI)
- **Chips**: HOT CHIPS 21
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

**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

**IMAPCAR** | **IMAPCAR2 (128PE/32PU)**
--- | ---
PO | 128PE@100MHz
LNO | 128PE@108MHz
SO | 128PE@100MHz
GlO | 128PE@108MHz
GeO | 128PE@100MHz
RNO | 128PE@108MHz
OO | 128PE@100MHz
Ave. | 128PE@108MHz