An Integrated Memory Array Processor Architecture for Embedded Image Recognition Systems

Shorin Kyo, Shin'ichiro Okazaki
NEC Corporation
Media and Information Research Laboratories
{s-kyo, s-okazaki}@cq.jp.nec.com

Tamio Arai
University of Tokyo
Department of Precision Engineering
arai-tamio@prince.pe.u-tokyo.ac.jp

Abstract

Embedded processors for video image recognition require to address both the cost (die size and power) versus real-time performance issue, and also to achieve high flexibility due to the immense diversity of recognition targets, situations, and applications. This paper describes IMAP, a highly parallel SIMD linear processor and memory array architecture that addresses these trading-off requirements.

By using parallel and systolic algorithmic techniques, despite of its simple architecture IMAP achieves to exploit not only the straightforward per image row data level parallelism (DLP), but also the inherent DLP of other memory access patterns frequently found in various image recognition tasks, under the use of an explicit parallel C language (1DC). We describe and evaluate IMAP-CE, a latest IMAP processor, which integrates 128 of 100MHz 8 bit 4-way VLIW PEs, 128 of 2KByte RAMs, and one 16 bit RISC control processor, into a single chip. The PE instruction set is enhanced for supporting 1DC codes. IMAP-CE is evaluated mainly by comparing its performance running 1DC codes with that of a 2.4GHz Intel P4 running optimized C codes. Based on the use of parallelizing techniques, benchmark results show a speedup of up to 20 for image filter kernels, and of 4 for a full image recognition application.

1. Introduction

Many efforts have been made thus far in designing embedded processors which can provide sufficient performance with less cost (die size and power). Although currently most of them aim at accelerating multimedia applications such as video processing, speech transcoding, and high-bandwidth telecommunication due to the huge market, video image processing and pattern recognition (hereafter image recognition) applications such as vision-based driver support system for ASV (Advanced Safety Vehicle), natural human interfaces, and robot vision systems, are also emerging markets for embedded processors. For example, ASV is predicted to spread to over a considerable amount of the world wide vehicle sales volume within four to five years.

Video image recognition tasks alike conventional multimedia tasks in some aspects, such as the large amount of data level parallelism and the requirement of real-time responses. However, prominent distinction between them are that, image recognition algorithms are much more diverse, due to their continuous evolution for coping with the variety of applications, recognition targets (can be anything), and changes of situations (day and night, weather conditions, appearances and sizes). These immense diversity demand image recognition processors not only to address the conventional high performance versus cost issue, but also to provide high enough flexibility for achieving compiler based application developments.

In this paper, we first describe our architecture design strategy of choosing a highly parallel SIMD architecture, to achieve highest performance at lowest cost, while maintaining a compiler based programmable flexibility. We next show that parallel and systolic algorithmic techniques can be applied for mapping classified memory access patterns of image recognition tasks onto one of such architecture, an integrated memory array processor (IMAP), using an explicit parallel C language design calls 1DC (One Dimensional C). We then describe and evaluate IMAP-CE[27], an implementation of the IMAP architecture, focusing on its enhanced RISC PE instruction set for supporting compiler generated codes of 1DC. IMAP-CE is evaluated by using various image filter kernels as well as a full application. Its performance is compared with a 24 times higher frequency Intel P4, by which a speedup of up to 20 for the image filter kernels, and of 4 for the full application are obtained, with a 50 to 100 times better power efficiency.

The rest of the paper is organized as follows. Section 2 describes our architecture design strategy. Section 3 analyzes the structure of image recognition tasks and derives techniques for mapping them onto the IMAP architecture.

0-7695-2270-X/05/$20.00 (C) 2005 IEEE

Section 4 describes the programming language for IMAP. In section 5 IMAP-CE are briefly described. In section 6, evaluation and the analysis results are shown. Section 7 concludes this paper and shows future directions.

2. Architecture Design Strategy

Generally, total circuitry (= total cost) of a processor LSI can be roughly classified into two categories: *operational circuitry* (datapaths and wired-logics) that determine its theoretical peak performance, and *control circuitry* (instruction issue management and interface logics) that contribute mainly for improving flexibility. Based on this classification, the inevitable trade-off between performance, cost, and flexibility can now be illustrated Fig. 1(a), while Fig. 1(b) shows the Control versus Operational circuit Ratio (COR) observation of several representative processor LSI implementations, estimated from their die photos [31][14][35][32][12], demonstrating a fairly accurate match between the COR value of each processor category and their well-known flexibility degree: a larger COR the higher a degree of flexibility.

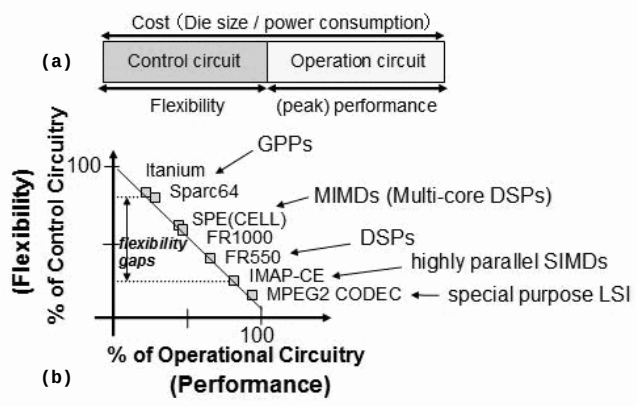


Figure 1. Relationship between performance, cost, and flexibility, and the Control versus Operational circuit Ratio (COR).

According to Fig. 1, the category "highly parallel SIMD", due to its nature of controlling a large amount of datapath using only a single instruction issue circuit (the SIMD paradigm), can be the best choice to maximize performance without raising the total cost, while still maintaining a compiler programmability. However, comparing with GPPs (General Purpose Processors), the reduced amount of control circuitry of highly parallel SIMD processors reveal a *flexibility gap*, poses the following general question to processor architects: "how are you going to keep these much of PEs busy?". In order not to be irresponsible to just pass on the issue to programmers, algorithm designers, or compiler engineers, architectural consideration should be taken to some extent, which however, hopefully not to result in raising its COR value. Our strategy toward this issue is to, first establish parallelizing techniques for mapping various image recognition tasks onto a simple highly parallel SIMD working model which has potentially an enough low COR. Then, we fill the inevitable *flexibility gap* by feeding back knowledge of those parallelizing techniques to the design of hardware and as well as its programming language, so as to ensure an efficient use of the highly parallel PE array via high level programming language codes. The rest of this paper provides the detail of our approach, and the evaluation results.

3. Parallelizing Techniques

In this section, we first categorize memory access patterns existing in image recognition applications, from a point of view of their inherent feasibility to be implemented on parallel architectures. Next, after choosing a linear array connection of processor elements as our underlying highly parallel SIMD architecture, we describe techniques for parallelizing each of the memory access pattern category onto that architecture.

3.1. Memory Access Patterns

Generally, tasks involved in an image recognition application can be classified into either low-, intermediate-, or high-level. Low-level tasks perform pixel-to-pixel transformation, while intermediate-level tasks transform pixel data into symbols, and finally high-level tasks perform rule based reasoning upon these symbols to derive a final recognition result. Within this three-level structure, by focusing on the most computational intensive portion, i.e. the low- and intermediate-level tasks, inherent pixel access patterns have been thus far classified into seven major pixel operation groups[17]: PO(Point Operation), LNO(Local Neighborhood Operation), SO(Statistical Operation), GeO(Geometrical Operation), RNO(Recursive Neighborhood Operation), and OO(Object Operation). Mathematical definition of these pixel operation groups can be found in [21]. Fig.2 illustrates their features in respect of source pixel access ranges, and their distribution among low- and intermediate-level tasks.

For facilitating parallelizing technique design, we have classified these pixel operation groups into the following four memory access pattern categories, from a viewpoint of the existence of data locality and pixel updating order constrains in them: Local/Unconstrained (LU),

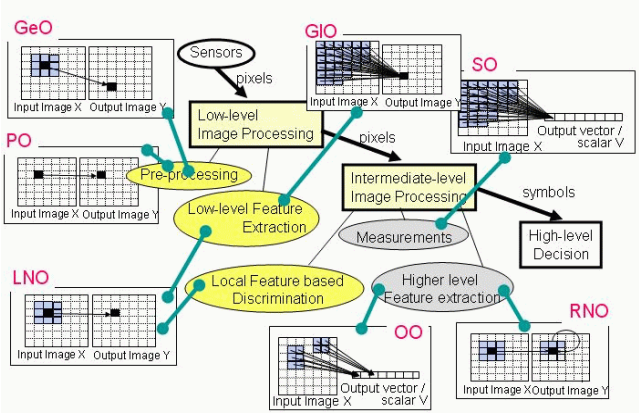


Figure 2. Distribution of the seven typical operation groups.

Global/Unconstrained (GU), Local/Statically-constrained (LS), and Local/Dynamically-constrained (LD). In our definition, data locality exists if pixel data required by an operation is within a considerable grid distance from the destination pixel location. We assume that source and destination image pixel of a similar grid location are mutually possessing data locality. Explanations of our classification criteria, including a brief view of each pixel operation group are presented in below.

In **PO**, source pixels are accessed according to each destination pixel location. Sample tasks are *Thresholding* and *Color Conversions*. In **LNO**, the source pixel access range is extended but still within a local area centered by each destination pixel location. Sample tasks are the various *2-D image filters*. We classifies PO and LNO to the LU category due to their local and unconstrained pixel updating sequences. In **GlO**, all source pixels are accessed for calculating a resulting destination pixel value. A typical sample task is the *Fourier Transform*. In **SO**, source pixels are accessed and (for example) accumulated as to obtain a scalar or a vector data. Sample tasks are *Histogram Calculation* or the *Hough Transform*. In **GeO**, source pixels are to be moved from a certain location to another destination pixel location. Sample tasks are *Affine Transform* and *Image Transposition*. We classifies GlO, SO, and GeO to the GU category due to their non-local (global) and unconstrained pixel updating sequences. In **RNO**, not only local source pixels but also some previously defined local destination pixel data are required for calculating each current destination pixel value, thereby their updating sequences are constrained by the destination pixel locations they should referred to. RNO is then classified to the LS category due to this local but statically constrained pixel updating sequences. Sample tasks are *Distance Transform[3]* or *Dither Transform*. Finally, **OO** is classified to the LD category due to its local but dynamically constrained pixel updating sequence nature, in most cases depending on image content, i.e. the pixel values around each destination pixel location. Sample tasks are *Region Growing*, or *Connected Component Labelling*.

With these four memory access patterns in mind, We next proceed to the choice of array configuration of a highly parallel SIMD architecture, and also the design of parallel mapping techniques.

3.2. Parallelizing Technique Design

Highly parallel SIMD architectures for image processing have been developed since early 80's. Since then, there exists four archetypes for the inter PE connection: Linear Processor Array (LPA)[8][10][11][20], Square Processor Array (SPA)[7][2], PYRamid (PYR)[30][5], and HyPeRcube (HPR)[16][33]. Among them SPA seems to provide a best match to the 2-D structure of an image. However, previous studies[18][13][6][15] have shown that LPA has no less computational and data I/O efficiency than SPA when the number of PE is the same, despite its cheapest hardware cost. Based on the above knowledges, and together based on the observation that a flexible memory architecture will facilitate the design of parallelizing techniques, we have chosen a most straightforward memory and processor linear array architecture, where each PE is attached with a considerable amount of locally addressable memory and connected in a ring, as our underlying architecture. Hereafter this array configuration will be referred to as IMAP (Integrated Memory Array Processor).

Before proceeding to the explanation of parallelizing techniques, Fig.3 shows the setup of the IMAP working model. For simplicity, an image width equal to the PE number, and a column wise mapping of an image to each PE is assumed. 2-D memory plane is a collection of all PE local memories, where the source, destination, and work images are assumed to be stored. The dot line drawn upon the 2-D memory plane is called a Pixel Updating Line (PUL).

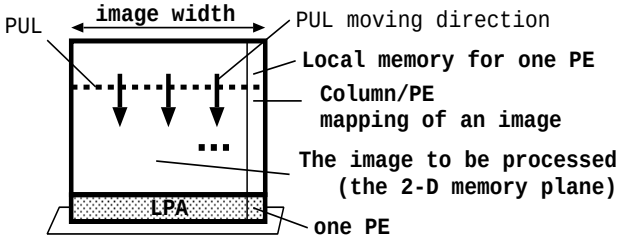


Figure 3. The IMAP working model.

PUL is consisted of a collection of memory addresses (or pixel locations), upon which, PEs work on simultaneously at each time unit. Thus, trajectory of a PUL over time can be regarded as the source or destination pixel area coverage of a task running on IMAP. Parallelizing techniques have been designed based on the idea of sweeping the PUL in various ways across the 2-D memory plane[24][25]. First of all, it is easy to understand that parallelizing LU can be achieved by making a clean sweep by a straight and horizontal *row-wise PUL* upon the 2-D memory plane, from top to bottom (or vice versa) as shown in Fig.3. *Row-wise PUL* can be easily implemented on IMAP by broadcasting to all PEs a single memory address (the line address) and increment it (proceed the PUL forward) as many times as the number of image lines. Parallelizing GU is achieved by the use of a PUL wrapping around at the leftmost and rightmost end while sweeps across the 2-D memory plane in a tilt shape and moves in a horizontal direction (the *row-systolic PUL*). The *row-systolic PUL* proceeds through the 2-D image plane and collect global information or perform global pixel location transformation. An example will be shown in Section 4.

Parallelizing LS is achieved also by using a tilt PUL but in combination with an interval activation of each PE (the *slant-systolic PUL* in Fig.4), for fulfilling the required delay of updating each neighboring pixel location. Finally, parallelizing LD is achieved by maintaining a software stack in each PE's local memory as a temporal storage of ready pixel location information, for proceeding an *autonomous PUL* as shown in Fig.4. These software stacks are used in combination with the follwing three procedures: 1) seed (or starting point) pixel location pushing, 2) neighboring ready pixel location detection and push, and 3) pixel location pop and *update*. By first initializing the PE stacks using 1), and then iterating 2) and 3) until all PE stacks are empty, *autonomous PULs* can be controlled to proceed and sweep through regions of interest within the 2-D image plane.

Fig.4 shows the correspondence between the seven pixel operation groups (PO/LNO/SO/GlO/GeO/RNO/OO), the four memory access pattern categories (GU/LU/LS/LD), and the parallelizing techniques (four types of PULs). Note that usually each parallelizing technique is not self-contained but is used in a mixture style. For example, a 2-D FFT will be implemented by first applying a *row-wise PUL* (1D-FFT), continued by applying a *row-systolic PUL* (transpose column to row), and finally by applying another *row-wise PUL* (1D-FFT). Hereafter, these parallelizing techniques will be collectively referred to as *line methods*[29]. Table 1 summarizes the expected speedup based on the use of *line methods* when the number of PE is *N*, where *M* represents the maximum pixel distance between the destination and the referred pixel location for LNO and RNO tasks.

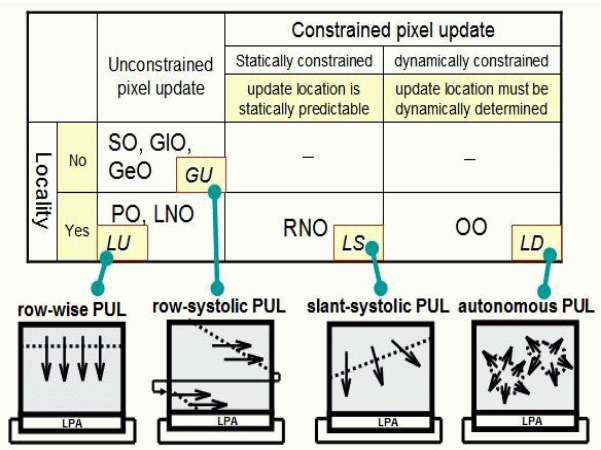


Figure 4. Correspondence between pixel operation groups, memory access pattern categories, and parallelizing techniques.

Op. group	Original complexity	Complexity on IMAP		Expected speed up
PO	$O(N^2)$	$O(N)$		N
LNO	$O((2M+1)^2 \times N^2)$	$=< O((2M+1)^2 \times N)$		N
SO	$O(N^2)$	$O(2N)$		N/2
GlO	$O(2N^3)$	x,y separable	$O(2N^2)$	N
	$O(N^4)$	otherwise	$O(N^4)$	1
GeO	$O(N^2)$	average case	$O(2N)$	N/2
RNO	$O((2M-1) \times M \times N^2)$	raster type	$O(M \times N+N)$	$O((2M-1) \times M \times N /(M+1))$
OO	$O(N^2)$	$O(N)$ up to $O(N^2)$		up to N

Table 1. Expected speedup of each operation group on IMAP using line-methods.

Applicability of *line methods* on IMAP can be evaluated by considering the efficacy of other general parallelizing methods when applying them to IMAP. For example, according to the experience of applying the conventional *divide-and-conquer* technique to image processing tasks on the WARP multi-processor system[37], although at the beginning processing time decreases according to the increase of the number of PE, gradually the speedup saturates, and finally processing time starts to increase due to inter PE communication overhead at the *conquering* stage[37]. This overhead is further maximized if a poor PE inter-connection configuration such as a LPA, other than a binary tree or a SPA is used. *Line methods* overcome such issue of a LPA, by using a parallel and systolic way for data communication between PEs, as shown by the *row-systolic PUL* and *slant-systolic PUL*.

4. Programming Language Design

In order for efficiently implementing *line methods* on IMAP using high level programming language, a data parallel C extension calls One Dimensional C (1DC)[24] is designed. In 1DC, entities associated with the PE array are declared by using a *sep* (or *separate*) keyword. A *sep* data possesses as many scalar elements as a multiple of the number of PE. Using sep variables in 1DC expressions specifies explicitly parallel operations. Requirements from *line methods* are fed back to the design of 1DC, resulting into six primitive extensions from C as shown in Fig.5.

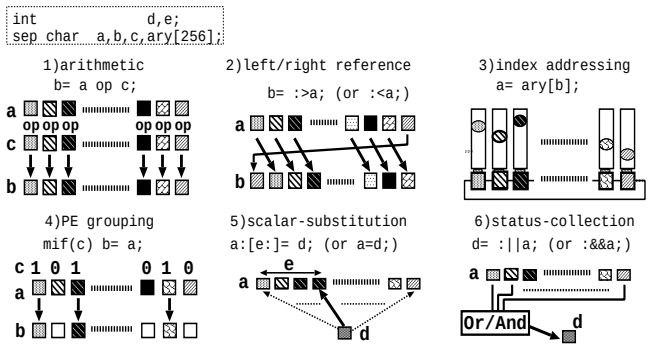


Figure 5. The six primitive 1DC syntax forms.

Table 2 shows the correpsondance between each parallelizing technique and the indispensable 1DC syntax. Architectural support of *line methods* can now be put in other words, that is, to design an IMAP processor which can achieve efficient execution of 1DC codes consisting of these syntax extensions.

PUL type	row-wise	row-systolic	slant-systolic	auto-nomous
1) arithmetic	X	X	X	X
2) left/right ref.	X	X	X	X
3) index addr.	-	X	X	X
4) PE grouping	-	-	X	X
5) scalar-subs.	-	-	X	X
6) status-collect.	-	-	-	X

Table 2. Correspondence between parallelizing techniques and the 1DC syntax.

Three sample 1DC codes: *binarize* (image thresholding:PO), *average* (average filter:LNO), and *histogram* (pixel histogram calculation:SO), are shown in Fig.6, where *NROW* designates the number of image rows, and the *sep* array *src* and *dst* stores respectively the source and destination image. Also for simplifying explanations for *histogram*, we assume each sep data with 256 elements (i.e. 256 pseudo PEs), and use a same pixel width source and destination images.

```
sep unsigned char src[NROW],dst[NROW];
void binarize(int thres) {   /*row-wise*/
  int i;
  for(i=1; i<NROW-1; i++)
      mif(src[i]>thres) dst[i]=0xff; melse dst[i]=0;
}
void average() {   /*row-wise*/
  sep unsigned int acc;
  int i;
  for(i=1; i<NROW-1; i++) {
    acc = src[i-1] + src[i] + src[i+1];
    dst[i] = (:<acc + acc + :>acc)/ 9;
  }
}
sep unsigned int histogram() {
  sep unsigned int hst[256], res=0;
  sep unsigned char idx=PENUM;/*1*/
  int i;
 /*2*/for(i=0; i<256; i++)  hst[i]=0;   /*row-wise*/
 /*3*/for(i=0;i<NROW;i++)  hst[src[i]]++;  /*row-wise*/
 /*4*/for(i=0; i<256; i++) {  /*row-systolic*/
       /*4-1*/res = :<(hst[idx] + res);
       /*4-2*/idx = :<idx;
  }
  return(res);
}
```

Figure 6. 1DC sample codes.

In *binarize* and *average*, resulting pixel data are calculated line by line, controlled by a scalar variable i of the array controller. The "PE grouping" (*mif...[melse...]*) syntax is used for dividing PEs into two groups, where one substitutes *0* and the other *0xff*, into each *dst[i]*, i.e. one line of the destination image. In *average*, acc is the vertical sum of 1×3 pixels, and the 3×3 pixel sum is calculated by using the "left/right reference" (":>" and ":<") operators to refer to adjacent PEs' acc. In *histogram*, each PE first independently calculates the histogram of an image column within its local memory and stores the result into a local *sep* array hst (/*2*/ and /*3*/) using *row-wise PUL*. Next, the *row-systolic PUL* (/*4*/) is used to combine these distributed hsts across all PEs. *PENUM*, a *sep* constant whose value is zero at the leftmost PE and increases one by one until 255 (the maximum PE number), is used as the initial value of idx (/*1*/) which is used in /*4-1*/ for indexing

hst while rotating its value between PEs(/*4-2*/), thereby generating a tilt and wrapped around PUL. Fig. 7 shows the breakdown of the loop /*4*/ focusing on the movement of one scalar data of *hst[1]*. According to Fig. 7, in the end of loop /*4*/, every N^{th} element of hsts on PEs are shown to be summed up into res of the N^{th} PE, thus the sep data res stores the final histogram result (in a distributed way).

```
i=0: res of PE0   = {hst[1] + 0} of PE1
i=1: res of PE255 = {hst[1]+res} of PE0
                  = {hst[1] of PE1} +
                    {hst[1] of PE0}
i=2: res of PE254 = {hst[1]+res} of PE255
                  = {hst[1] of PE255}+
                    {hst[1] of PE1} +
                    {hst[1] of PE0}
    ..........
i=255:res of PE1  = {hst[1]+res} of PE2
                  = {hst[1] of PE2} +
                    {hst[1] of PE1} +
                    {hst[1] of PE0} +
                    {hst[1] of PE255}+
                    ............
                    {hst[1] of PE3}
```

Figure 7. The histogram summing loop.

5. The IMAP-CE Linear Processor Array

This section describes the design of IMAP-CE, including its building blocks, memory and video data I/O system, its enhanced features for efficient support of *line methods*, and the software system. Connections of multiple IMAP-CE chips to form a larger SIMD or multi-SIMD system are also supported by the chip in hardware, however in this paper we focus only on the single chip configuration.

5.1. Building Blocks

Main building blocks of IMAP-CE are the control processor (CP), the 128 PE array (PE8×16, PE8 is a group 8 PEs), and the external memory interface (EXTIF). Fig.8 shows the block diagram and die photo of the chip, which is fabricated using 0.18um CMOS process integrating 32.7M transistors into a 11x11 mm^2 die, and packaged using 500-pin TBGA. Application level power consumption is estimated to be in average around 2 Watts.

CP is a 6 stage pipelined general purpose 16-bit RISC processor equipped with a 32KB program cache, a 2KB data cache, and host interfaces. CP issues up to 4 instructions per cycle, out of which one is for itself, and up to 4 are broadcasted to the PE array. Each PE of the PE array is a 3 stage pipelined 8b RISC datapath attached with a 2KB single port RAM (IMEM), 24 8b general purpose registers, an 8b ALU, an 8bx8b multiplier, a load-store unit (LSU), a reduction unit (RDU) for communicating with CP, and an inter PE communication unit (COMM) using the ring inter-connection between PEs. Each PE can execute up to 4 instructions per cycle (4-Way VLIW). EXTIF contains a DMA engine for data transfer between IMEMs and EMEMs (SDRAM), a hardware image line data scaling unit (from 25% to 400%) attached to the DMA engine, and a SDRAM controller with an arbiter for arbitrating accesses from 1) CP (during program/data cache misses), 2) host (via bus interfaces), and 3) the DMA engine.

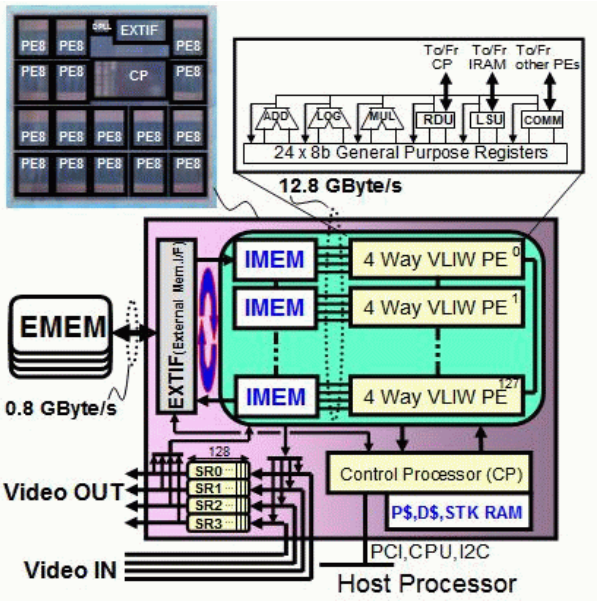


Figure 8. IMAP-CE Block Diagram/Die-photo.

5.2. Memory System and Video Data I/O

The collection of all IMEMs (256KB) forms the 2-D memory plane as described in Fig.3, and provides working spaces for applying *line method* based algorithms. The 1DC compiler for IMAP-CE explicitly maps all sep data in default onto IMEMs, while maps sep data with an additional $outside$ declaration onto the EMEM. EMEM (in max. 64MB) provides both swapping area for the 2-D memory plane, and the program and data memory area (each up to 16MB) for CP.

Data exchanges between sep and *outside sep* spaces must be done by explicitly invoking the DMA engine from the user program. The DMA engine contains two priority request queues of 32 depth, to allow software scheduled data transfers of any specified rectangle area of the 2-D memory plane in burst modes. During each DMA data transfer, the 128 single port IMEMs of the PE array are ef-

ficiently shared by the DMA engine using line-buffers, i.e. a sixteen shift register configuration, where each shift register is 64b and mapped to one PE8. The DMA engine occupies only one single IMEM access cycle of the PE array, during which the 12.8GB bandwidth is used for transferring one complete row of image or 128B of data between IMEMs and line-buffers, while an extra 16 continuous cycles are further required for shifting them into or out from, between line-buffers and the EMEM space (via EXTIF). These actions are done in parallel with the PE array execution thereby overcoming memory latency and fully exploit the 0.8GB/s memory bandwidth although 1/16 of that between the PE array and the IMEMs.

Four video shift register channels (SR0-SR3, 128x8b each shown at the bottom of Fig.8) are used for efficient I/O of stereo or color(R/G/B or YCrCb) video data of cameras. These SRs operate in video clock and each element of all four SRs are mapped to each PE. Connection between inter-PE SR elements are re-configurable, exchanging a mapping of from 1 up to 4 video pixel data to each PE, with the number of available SR channels[27]. Whenever each video image line has been completely shifted into a SR channel, it is transferred to a pre-assigned work space of the EMEM by the DMA engine invoked from a software interrupt routine, which is in term activated by a hardware interrupt signal generated by internal control-logics which count video clock ticks based on the NTSC *horizontal sync.* and *valid* video signal inputs provided by an off-chip video decoder.

5.3. The Enhanced PE Design

Fig. 9 shows the pipeline stages of CP and the PE array. Most PE instructions perform register to register operation, and accomplish in one cycle (RISC instruction set). Table 3 shows representative PE instructions. $ir*$ and $cr*$ are respectively general purpose registers of the PE array and CP. $ir * P$ represents a pair of $ir*$. While a $cr*$ represents a scalar data, a ir* represents a vector data of 128 length. ped is a special purpose CP register for storing collected PE status information. The notation $fs(ir3, expr)$ represents the value of a resulting flag of $expr$ operation, where the flag type is specified by $ir3$. mr and mf are two single bit special registers: if the *mask bit field* of an instruction is 1, then for PEs whose mr is 0, write-back action of the instruction is inhibited. mf is a temporal storage for keeping the reversed value of mr. Both of them are implicitly defined by a group of mif PE instructions.

Enhancements of IMAP-CE toward efficient execution of the six primitive 1DC syntax are summarized in below.

1. **Arithmetic Operation:** A 4-way VLIW design enables each PE to execute in maximum three different *sep* data operations and one memory access in one cycle, exploiting instruction level parallelism within the single instruction stream broadcasted from CP.
2. **Left/right Reference Operation:** Each PE can perform a single cycle register to register data access of neighboring PEs using a 16b wired inter PE connection ($mvrp, mvlp$ in Table 3).
3. **Index Addressing Operation:** Each PE has a private RAM (IMEM), enabling all PEs to access to a mutu-

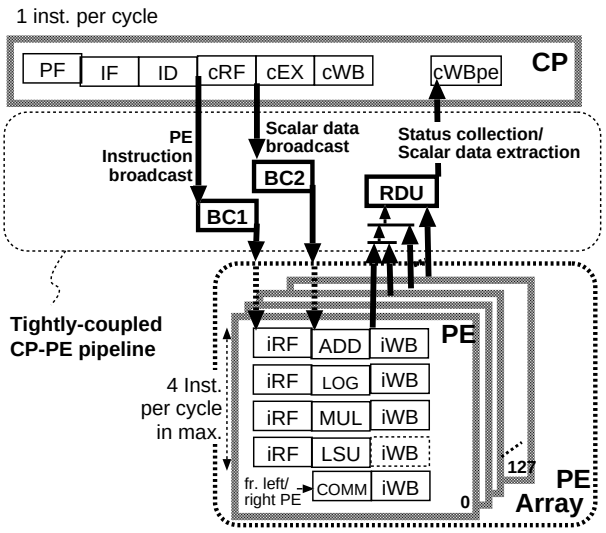


Figure 9. CP and the PE array pipeline stages.

name	*action*	*name*	*action*
Instructions for arithmetic operations (39 in total)			
add	*ir3=ir1+ir2*	*sub*	*ir3=ir1-ir2*
mul	*ir3P=ir1 x ir2*	*ssub*	*ir3=saturate(ir1-ir2)*
abs	*ir3=|ir1-ir2|*	*sadd*	*ir3=saturate(ir1+ir2)*
max	*ir3=max(ir1,ir2,ir3)*	*min*	*ir3=min(ir1,ir2,ir3)*
sll	*ir3=ir1≫1*	*mv*	*ir3=ir1*
Instructions for left/right reference operations (6 in total)			
mvr	*ir3=ir1 of left PE*	*mvl*	*ir3=ir1 of right PE*
mvrp	*ir3P=ir1P of left PE*	*mvlp*	*ir3P=ir1P of right PE*
Instructions for normal/indexed addressing operations (8 in total)			
ld	*ir3=IMEM[cr1+cr2]*	*st*	*IMEM[cr1+cr2]=ir1*
ldt	*ir3=IMEM[cr1+ir2P]*	*stt*	*IMEM[cr1+ir2P]=ir1*
Instructions for PE grouping operation (4 in total)			
mif	*mr=fs(ir3,ir1-ir2)&mr, mf=fs(ir3,ir1-ir2)∧1)&mr*		
mifc	*mr=fs(ir3,ir1-ir2-carry)&mr, mf=fs(ir3,ir1-ir2)∧1)&mr*		
Instructions for scalar substitution operation (4 in total)			
mv2	*ir3=cr1*	*pdp*	*ir3 of PE(cr1) = ped*
Instructions for status collection operation (3 in total)			
sml	*zero clear each PE's mr except the leftmost non zero one*		
sts	*ped = bit-wise OR all PEs' ir1*		

Table 3. Representative PE instructions.

ally different memory address simultaneous within one cycle (ldt, stt in Table 3).

4. **PE Grouping Operation:** Hard-wired instructions, which are combinations of 8/16 bit subtraction, flag reference, and several simple logical operations, are added for achieving a PE grouping operation, i.e. setting the proper value to mr, in one cycle, which will otherwise requires five cycles($mif, mifc$ in Table 3).
5. **Scalar Substitution Operation:** A tightly coupled CP-PE pipeline is designed for a pipelined communication between CP and the PE array of one cycle throughput ($mv2, pdp$ in Table 3).
6. **Status Collection Operation:** bit-wise OR of all PE data for calculating the PE array status are performed in the RDU stage in a hierarchical way(Fig. 9). The tightly coupled CP-PE pipeline also contributes to maintain a single cycle throughput of this PE status collection(sml, sts in Table 3).

5.4. Software Tools

The IMAP-CE software system provides the compile-time and run-time support necessary for debugging and running 1DC programs. As shown in Fig.10, the software system includes an 1DC optimizing compiler, an assembler and a linker, for converting 1DC programs into IMAP executables, and also a symbolic debugger.

The 1DC compiler performs various automatic code optimization such as loop-unrolling and software pipelining for increasing the chance of filling the 4 Way VLIW PE instruction slots. The 1DC symbolic debugger provides not only the basic break-point insertion functionalities, but also various debug-time and run-time support of 1DC programs, such as profiling commands for isolating performance bottlenecks, and GUI tools for dynamically adjusting control variables, respectively for performance tuning and run-time debugging using parameter tuning.

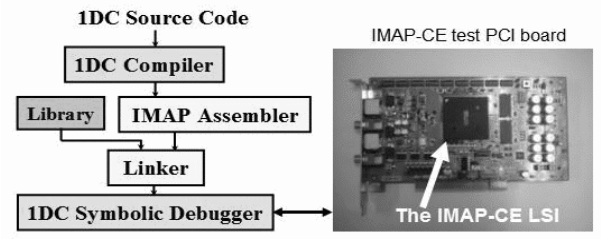


Figure 10. Software tools and the IMAP-CE PCI board.

6. Evaluation

6.1. Operation Group Kernels

1DC codes running on a 100MHz IMAP-CE are compared with a 2.4 GHz Intel P4 running C codes generated by Intel C++ compiler Ver.7 with full speed optimizations. Note that the same operation frequency of both IMAP-CE and GPP, and the same compilation condition for C codes will be assumed throughout this section.

In our first benchmark, the seven typical image processing kernels, each corresponding to one of the pixel operation group described in Section 3.1, are written in 1DC based on the use of *line methods*, while the corresponding C codes are based on conventional sequential algorithms. Size and precision of the source image is 128x240 and 8b/pixel, while the pixel precision of destination images vary from 8b (for *rot90*) to 32b (for *fft*).

Fig.11 shows the benchmark results together with the expected parallelism of each kernel under the use of *line methods*. The speedup gained by IMAP-CE are shown to be proportional to the expected parallelism of each kernel under the use of *line methods*, demonstrating that the combinations of IMAP-CE and the *line methods* based 1DC codes achieve in exploiting inherent parallelism of these kernels.

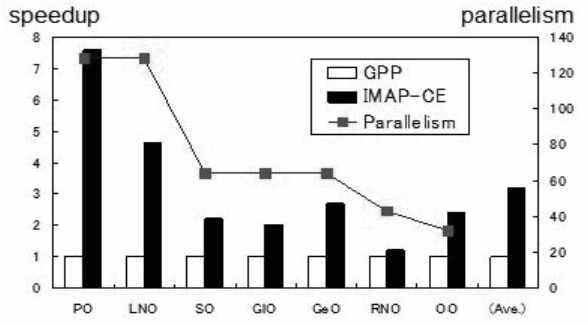
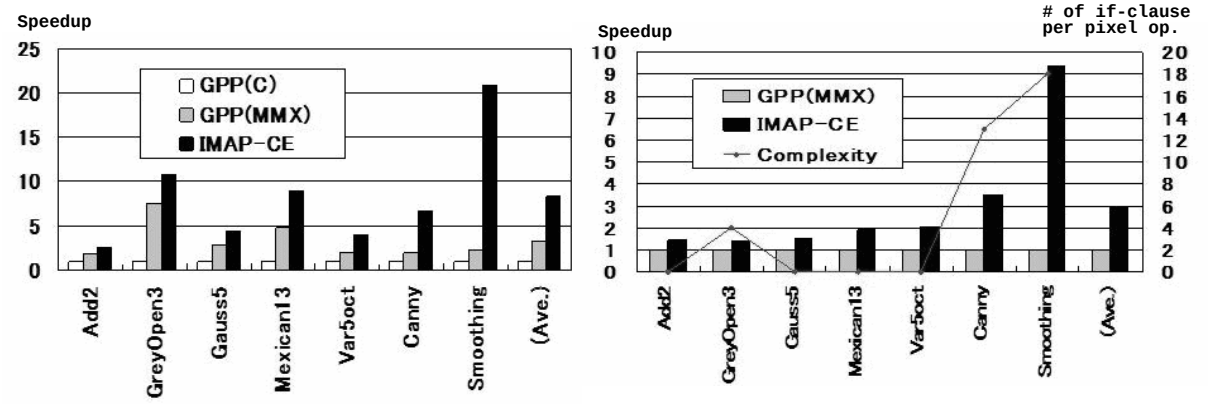


Figure 11. Performance compared with a 2.4GHz GPP using operation group kernels.

On the one hand, Table 4 summarizes instruction per cycle (IPC) and PE instruction active cycle ratio data (the Active column) of IMAP-CE. In average, PE instructions are issued at 87.4% of the total cycle, i.e. 12.6% of them are occupied by bookkeeping single CP instructions. Such PE array idling ratio reaches in maximum 22.8% in the case of the $labelling$ kernel. The 22.8% of CP instructions are mainly consisted of function call overheads, loop control overheads, and waiting cycles of the DMA engine due to the insufficient IMEM capacity of keeping all required working image spaces in IMEM at the same time. The average IPC is 1.44, showing in average 1.44 slots out of the 4 VLIW



(a) Speedup of 1DC codes and MMX codes compared with C codes.

(b) Speedup of 1DC codes compared with MMX codes

Figure 12. Speedup ratio compared with MMX codes and C codes running on a 2.4 GHz GPP.

Op. Grp.	Kernel	IPC	Active (%)	Remarks
PO	hsi2rgb	1.40	94.6	Color format trans.
LNO	ave3	1.33	90.6	3x3 average filter
SO	hist	1.66	91.0	Histogram calc.
GlO	fft	1.55	80.2	Fast Fourier Trans.
GeO	rot90	1.23	89.5	90 Degree Rotation
RNO	dtrans	1.52	88.9	Distance trans.[3]
OO	labelling	1.40	77.2	Connected Component Labeling
Average		1.44	87.4	-

Table 4. Performance data of IMAP-CE for each operation group kernel.

instruction slots are used. This result is reasonable if taking into account the use of compiler generated codes, and the existence of in average over 13% of bookkeeping CP instructions. The design choice to include only one ALU and use a single ported IMEM are also found to limit efficient instruction insertion by the compiler. Based on these insights, to increase the number of ALU can be one choice for a future IMAP LSI design, however, should be carefully examined by evaluating the trade-off between IPC improvement and the increase of hardware cost, as any additional circuitry of a PE will be amplified by 128 times into the total chip cost.

6.2. Highly Parallel vs. Sub-word Level SIMD

Fig.12 shows the result of comparing the performance of the following two SIMD approaches: highly parallel SIMD with enhanced PE designs (IMAP-CE), and the sub-word SIMD parallelism using media extended instructions (recent GPPs and DSPs). In this benchmark we use the MMX instruction set of Intel GPPs. Table 5 describes the kernels being used. Due to the limited support of memory access patterns of most media extended instructions, only kernels belonging to PO (Add2) and LNO (others) are used. Processing time of these kernels in 1DC codes running on IMAP-CE, and both in MMX codes and in C codes running on the Intel P4, are measured and compared. MMX codes are generated automatically by compiling the same 1DC code for IMAP-CE using a MMX 1DC compiler, which has been shown in [28] to be able to generate codes with comparable performance to hand-optimized MMX codes for 2-D filter 1DC programs.

Kernel	Description	Complexity(# of if-clause/pixel op.)
Add2	simple dyadic arithmetic	0
GreyOpen3	grey-level Morph. (3x3)	4
Gauss5	Gaussian filter (5x5)	0
Mexican13[34]	edge analysis (13x13)	0
Var5oct[34]	texture analysis (5x5)	0
Canny[4]	precise edge detection (3x3)	13
Smooth	edge preserving smoothing (7x7)	18

Table 5. PO and LNO Kernel descriptions.

According to Fig.12(a), 1DC codes achieve an average speedup of respectively 3 and 8, compared with MMX codes and C codes, while further insights of these results can be found from Fig.12(b) which focuses on performance differences between 1DC and MMX codes. The speedup gained by 1DC codes starts from approximately 1.3, which is a figure conforming to the byte precision peak performance difference between the two target processors. How-

Processor Name	Cycle counts	proc. time (us)	word size	MHz	die-size (mm^2)	power	Tech. (um)
Imagine(Float.)	2176	7.4	16	296	12*12	4W	0.15
Morphosys2	2636	5.8	16	450	16*16	4W	0.13
IMAP-CE	5000	50.0	8	100	11*11	2W(ave.)	0.18
VIRAM	5280	26.4	16	200	15*18	2W(ave.)	0.18

Table 6. 1024 point 1D-FFT performance compared with other media processors.

ever, the speedup gradually increases in proportion to the increase of kernel complexity, i.e. the number of conditional branch statements, while such tendency is not observed from performance differences between 1DC and C codes shown in Fig.12(a). The above speedup behavior difference reveals the importance of supporting efficient SIMD style conditional branch executions in hardware, such as the PE grouping instructions of IMAP-CE.

6.3. Comparison with Recent Media Processors

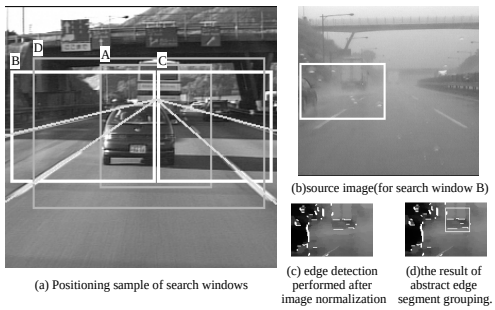
The programming style of IMAP is based on an extensive use of a 2-D memory plane, which is physically a collection of hundreds of (banks of) single cycle reachable non-cached on-chip RAMs. Similar approaches of using such *scratch pad memories* can be found in the Streaming Register File (SRF) of Imagine[1], the Frame Buffer of Morpohsys[36][19], and the Local Store of each SPE (Synergistic PE) in CELL[9]. However, most of them are organized in one to several banks. The highly banked local memory configuration of IMAP thus provide the highest flexibility for each PE to freely address image pixels. The cost of this flexibility is estimated to be approximately a 10% larger chip size, when compared with a condition of unifying eight 2KB (8b/W) RAMs into a 16KB (64b/W) RAM.

There exists also similarities between IMAP architecture and vector microarchitectures. VIRAM[22] partitions vertically a vector and assign them to four vector lanes, while IMAP-CE statically partitions the sep data (a vector whose size is a multiple of 128) and assign them to the 128 PE array. CODE[23] issues multiple vector instructions in parallel by using a vector cluster configuration and implements vector chaining by synchronized inter cluster communications, while the VLIW PE design of IMAP-CE enables execution of multiple *vector instructions* upon sep data in each cycle, and *vector chaining* are fully supported by complete operand forwarding across the PE pipeline stages.

Table 6 shows a quick performance comparison of four multimedia processors including IMAP-CE, using a 1024 point 1D-FFT kernel. Figures for other processors are based on the data from [19], while cycle counts for IMAP-CE are estimated by porting the fixed point sequential C program of a radix-2 1-D FFT into 1DC, execute the 1DC compiler generated code on every PE, and then divide the processing time by the PE number, in order to take into account the highly parallel nature of the architecture. The reason of the double cycle counts of IMAP-CE compared with Imagine and Morphosys2 can be explained by the miss match of the 32 bit precision FFT algorithm with a 8 bit architecture. Bit width expansion can be a future design choice for the IMAP architecture, but again must be carefully examined by evaluating the trade-off between its effect toward real world image recognition applications and the total increase in hardware cost.

6.4. Application Level Evaluation

The application used in the benchmark is a vehicle detection program described in [26]. As shown in Fig.13(a), the source image is first divides into four overlapping regions A to D, upon each of which a sequence of image recognition tasks are applied: normalization of image pixel values within the each region, (the *Prepare* task), edge detection (the *Detect_edge* task), and edge segment selection based on sizes and strengths of each detected edges (the *Select_edge* task, also refer to Fig.13(c)). Selected edge segments of all four regions are then grouped together for identifying locations of *potential vehicles*, based on priori knowledge such as, 1) a *potential vehicle* possesses prominent vertical edge pairs, 2) each pair edges should locate at an adequate horizontal distance according to their position within the image (the *Grouping* task, refer to Fig.13(d)). Finally, *potential vehicles* are verified by grey level pattern information within their square-shaped regions (the *Select_rect* task), by which final vehicle detection decisions are made.



(a) Positioning sample of search windows

(b)source image(for search window B)

(c) edge detection performed after image normalization

(d)the result of abstract edge segment grouping.

Figure 13. The vehicle detection algorithm.

Fig.14(a) shows the performance comparison of 1DC codes running on IMAP-CE and C codes running on a GPP (Intel P4). C codes are functionally equal to that of the 1DC code, except that sequential algorithms for each task are used. The 1DC program is designed based on the use of *line methods*. As shown in Fig.14(a), IMAP-CE outperforms GPP approximately 4 times in total processing time. When considering of the 2 Watt in average power consumption of IMAP-CE, the application level power efficiency of IMAP-CE can be estimated as of 50 to 100 times better than that of the GPP.

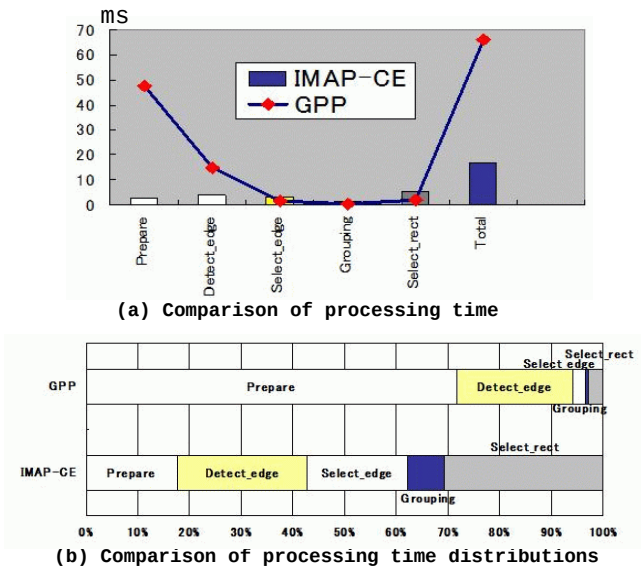


Figure 14. The application benchmark result.

However, the processing time breakdown of both processors (Fig.14(b)) show that IMAP-CE outperforms GPP for *Prepare* and *Detect_edge*, while GPP outperforms IMAP-CE 2 to 4 times for *Select_edge*, *Grouping*, and *Select_rect*. In *Select_edge* and *Grouping*, although edge segment (pair) examinations are performed in parallel by the PE array, the number of edge segment being detected by *Detect_edge* are usually modest compared with the PE number, i.e. existing parallelism are smaller than the number of PE. In *Select_rect*, as *potential vehicle* regions may appear in a scattered and overlapped way within the whole source image, sequential examination of each region is inevitable due to the rather high cost of re-distributing pixel data of all *potential vehicle* regions into an un-overlapped layout for facilitating examination of them in parallel by the PE array. The sequential examination approach limits the parallelism to only the pixel width of each *potential vehicle* region, which is usually rather smaller than the number of existing PE.

According to the processing time distribution of IMAP-CE shown in Fig.14(b), the processing time consumed by tasks of coarse grain parallelism (*Select_edge*, *Grouping*, and *Select_rect*) now occupies up to 55% of the total processing time, partly due to the successful acceleration of data intensive fine grain tasks (*Prepare* and *Detect_edge*)) which originally occupy over 95% of the GPP processing time. The results suggest that, in order to fruitfully apply highly parallel SIMD architectures to real world applications, enhancement of the architecture towards not only an efficient support of data intensive fine grain parallelism, but also towards a more flexible way of supporting other levels of parallelism, such as *region parallelism*, is becoming one of the important future issues to be addressed.

7. Conclusions and Future Directions

Multimedia processing on embedded systems is an emerging computing area with significantly different needs from the desktop domain. This paper focused on image recognition applications, which unlike other multimedia applications such as video CODEC, strongly demand the embedded processor to provide not only high performance under low cost, but also high flexibility due to the algorithmic diversity which come from the inherent open air scene analysis complexity.

In this paper, we have demonstrated the efficacy of the IMAP architecture for the image recognition application domain, by evaluating various kernel benchmarks and a vision-based ASV application, using one of its LSI realization, IMAP-CE. In our kernel benchmarks of high level language (1DC) codes, a speedup in proportion to the inherent parallelism of each operation group kernel shows the efficacy of using IMAP-CE in combination with *line methods*, while the average speedup is respectively 3 and 8 for data intensive image filter kernels, comparing with subword SIMD instruction codes and C codes, running on a GPP of 24 times higher operation frequency. A benchmark of 1024 point 1D-FFT also shows that IMAP-CE has an equivalent performance even when using compiler generated codes, compared with recent parallel media processors based on the same SIMD paradigm. Finally the flexibility of the IMAP architecture is demonstrated by a full application benchmark written in 1DC, showing that a 100MHz IMAP-CE consuming only 2 Watts in average obtain a promising speedup of 4 compared with a 2.4GHz GPP that consumes near 100 Watts.

On the other hand, one potential draw back of the IMAP architecture is its pure SIMD model, which may in some cases inhibit a more efficient usage of the PE array. Such inefficiency has been demonstrated by some of the vehicle detection application benchmark results. However, as highly parallel SIMD is a best architectural choice regarding its

low COR as described in Section 2, extensions toward the SPMD or MIMD model will not be a future choice. We think that, reinforcement of performance and array control flexibility of the control processor, an approach that makes little change to the COR value of the architecture, will be one important issue to be addressed, for both achieving a better sequential task performance to avoid the Amdahl's law, and also for achieving a more flexible PE array control and load balancing for real world applications.

Acknowlegements

We would like to acknowledge all the members of the IMAP research team, in particular T.Koga and K.Sato for their intensive works for respectively implementing the IMAP-CE LSI and the 1DC compiler.

References

[1] J.H.Ahn, et al.:"Evaluating the imagine stream architecture", Proceedings of 31st Annual Int. Symp. on Computer Architecture, pp.14-25, 2004.

[2] R.S.Bajwa, et al.:"Image processing with the MGAP: a cost effective solution", Proceedings of Seventh Int. Parallel Processing Symposium, pp:439–443, 1993.

[3] G. Borgefors: "Distance Transformations in Digital Images", Computer Vision, Graphics, and Image Processing, Vol.34, pp.344–371, 1986.

[4] J. Canny: "A Computational Approach to Edge Detection", IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol.9, No.6, pp.679–698, 1986.

[5] V.Cantoni, L.Lombardi: "Hierarchical architectures for computer vision", Proc of Euromicro Workshop on Parallel and Distributed Processing, pp.392–398, 1995.

[6] R.Cypher, J.L.C.Sanz: "SIMD architectures and algorithms for image processing and computer vision", IEEE Trans. on Acoustics, Speech and Signal Processing, Vol.37, No.12, pp.2158–2174, 1989.

[7] M.J.Duff: CLIP 4: A large scale integrated circuit array parallel processor, Proc. of IEEE Int. Joint Conf. on Pattern Recognition, pp.728–733, Nov. 1976.

[8] A.L.Fisher, et al.:Architecture of a VLSI SIMD processing element, Proc. IEEE Int. conf. on Computer Design: VLIS computer process, pp. 324-327, 1987.

[9] B. Flachs, et al.: "A Streaming Processing Unit for a CELL Processor", ISSCC Digest of Technical Papers, 7.4, pp.134–135, 2005.

[10] T.J.Fountain, K.N.Matthews, and M.J.B.Duff, "The CLIP7A Image Processor", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol.10, No.3, pp.310–319 (1988).

[11] Y.Fujita , N. Yamashita, S. Okazaki,"IMAP: Integrated Memory Array Processor",Journal of Circuits, Systems and Computers, Vol.2, No.3, pp.227–245 (1992).

[12] J.Geerlings, et al.: "A Single-Chip MPEG2 CODEC for DVD+RW", ISSCC Digest of Technical Papers, 2.2, 2003.

[13] D.W.Hammerstrom and D.P.Lulich: "Image Processing Using One-Dimensional Processor Arrays", Proceedings of the IEEE, Vol.84, No.7, pp. 1005–1018, 1996.

[14] J.Hart, et al.: "Implementation of a 4th-Generation 1.8GHz Dual-Core SPARC V9 Microprocessor", ISSCC Digest of Technical Papers, 10.3, pp.186-187, 2005.

[15] D.Helman, J.JaJa: Efficient Image Processing Algorithms on the Scan Line Array Processor, IEEE Trans. on Pattern Analysis and Machine Intelligence , Vol.17, pp.47–56, 1995.

[16] W.D.Hills: The Connection Machine, MIT Press, Cambridge, MA, 1985.

[17] P.P.Jonker: Architectures for Multidimensional Low- and Intermidiate Level Image Processing, Proc. of IAPR Workshop on Machine Vision Applications, pp.307–316, 1990.

[18] P.P.Jonker: "Why linear arrys are better image processors", Proc. of IAPR Conf. on Pattern Recognition, Vol.3, pp.334–338, 1994.

[19] A.H.Kamalizad, et al.:"Fast parallel FFT on a reconfigurable computation platform", Proc. of 15th Symposium on Computer Architecture and High Performance Computing, pp.254–259, 2003.

[20] R.P.Kleihorst, et al.:IEEE Int. Symp. Circuits System, Vol.2001, Vol.5, pp.215–218, 1991.

[21] E.R.Komen: "Low-level Image Processing Architectures", Ph.d Thesis, Delft University of Technology, 1990.

[22] C.E.Kozyrakis, D.A.Patterson: "Scalable Vector Processors for Embedded Systems", IEEE Micro, Vol.23, Issue 6, pp.36-45, Nov.-Dec. 2003.

[23] C.E.Kozyrakis, D.A.Patterson: "Overcoming the Limitation of Conventional Vector Processor", Proc. of 30st Annual Int. Symp. on Computer Architecture, pp.399-409, 2003.

[24] S. Kyo and S.Sato: "Efficient Implementation of Image Processing Algorithms on Linear Processor Arrays using the Data Parallel Language 1DC", Proc. of IAPR Workshop on Machine Vision Applications (MVA'96), pp.160–165, 1996.

[25] S.Kyo, et al.: "A Parallelizing Method for Implementing Image Processing Tasks on SIMD Linear Processor Arrays", Proc. of IEEE Workshop on Computer Architecture for Machine Perception, pp.180–184, 1997.

[26] S.Kyo, et al.:A Robust Vehicle Detecting and Tracking System for Wet Weather Conditions using the IMAP-VISION Image Processing Board, Proc. of IEEE Int. Conf. on Intelligent Transportation Systems, pp.423–428, 1999.

[27] S.Kyo, et al.: "A 51.2GOPS Scalable Video Recognition Processor for Intelligent Cruise Control based on a Linear Array of 128 4-Way VLIW Processing Elements", ISSCC Digest of Technical Papers, 2.6, pp.48–49, 2003.

[28] S.Kyo, et al.: "An Extended C Language and a SIMD Compiler for Efficient Implementation of Image Filters on Media Extended Micro-processors", Proc. of Advanced Concepts for Intelligent Vision Systems (ACIVS), pp.234–241, 2003.

[29] S.Kyo : "Design and Development of a Linear Processor Array (LPA) Architecture for Accelerating Image Recognition Tasks", Ph.d Thesis, University of Tokyo, 2004.

[30] A.Merigot, B.Zavidovique, F.Devos: SPHINX: a pyramidal approach to parallel image processing, in Proc. IEEE Workshop Computer Architecture Pattern Analysis Image Database Management, pp.107–111, 1985.

[31] S.Naffziger, et al.: "The Implementation of a 2-core Multi-Threaded Itanium-Family Processor", ISSCC Digest of Technical Papers, 10.1, pp.182–183, 2005.

[32] Okano, et al.: "An 8-way VLIW Embedded Multimedia Processor Built in 7-layer Metal 0.11um CMOS Technology", ISSCC Digest of Technical Papers, pp.374–375, 2002.

[33] T.Poggio, et.al.: The MIT Vision Machine, Proc. of Image Understanding Workshop, pp.177-198, 1988.

[34] J.C.Russ: The Image Processing Handbook, CRC Press, (1994).

[35] T.Shiota, et al.: "A 51.2 GOPS 1.0GB/s-DMA Single-Chip Multi-Processor Integrating Quardruple 8-Way VLIW Processors", ISSCC Digest of Technical Papers, 10.7, pp.194–195, 2005.

[36] H.Singh, et al.:"MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications", IEEE Trans. on Computers, Vol. 49 , No. 5, pp. 465-481, 2000.

[37] J.A.Webb: Steps Toward Architecture- Independent Image Processing, Computer, pp.21–31, 1992.