IMAP-based SIMD and SIMD/MIMD hybrid architectures for image processing

Ben Kelly

April 5, 2011

Abstract

Realtime image processing is a rapidly growing application field. It is especially important to the automotive industry, where a fast image processor can provide collision detection, lane following, and overtake features, among others. Traditionally, special-purpose SIMD chips have been used for this purpose; however, modern image analysis techniques require true multiprocessing capability. To fill that need, hybrid SIMD/MIMD chips are starting to appear. This paper discusses the history of the IMAP architecture and the IMAP-CE, IMAPCAR, and IMAPCAR2 implementations of it, with a particular focus on the dynamic reconfiguration capabilities and programming model of the IMAPCAR2.

1 Background

Multimedia processors are, in this day and age, commonplace. Most development in this area focuses on audio/video transcoding hardware - devices to compress and decompress audio and video streams in real time. This can typically be implemented using comparatively cheap and efficient special-purpose hardware, with each chip implementing a single algorithm.

There is, however, an increased demand for real-time image processing and classification, especially in the domain of semi-autonomous vehicles. Hardware is needed that can not only transform captured images, but also analyze them - searching for obstacles, lane markers, and other features [6]. Performing these operations in real time is beyond the capability of typical embedded processors, such as the ARM, especially when multiple video streams need to be processed. However, general purpose processors (GPPs) – such as those found in modern desktops – are far too large, expensive, and power-hungry; and application specific integrated circuits (ASICs) are too inflexible, requiring a new circuit for each image processing algorithm used – and consequently a hardware redesign if algorithms need to be updated or added. It follows that to be useful for this purpose, a chip is required that is fully programmable, but without the drawbacks inherent in using an off-the-shelf GPP [5].

2 IMAP

2.1 The Integrated Memory Array Processor

A description of the IMAP architecture was first published by Fujita et al. [3] in 1995. It specifies a heavily SIMD-oriented design, consisting of a homogenous array of processor elements (*PE*s), each one possessing local memory (*internal memory* or IMEM) containing a slice of the image data to be operated on. The intent of the design is to exploit the high degree of data-level parallelism inherent in most image processing operations - the image can be divided up, row-wise or column-wise, among the PEs, which then process the image data and produce results to be collected by a central processor. Since the PE array is fully programmable, a degree of flexibility not possible with ASICs is obtained; at the same time, the massively SIMD architecture allows most image processing tasks to be performed very quickly, while the simplicity of the individual PEs keeps manufacturing costs and power requirements down.

For communication between PEs, they are linked in a simple ring network, which permits each PE to communicate with the PEs directly to the left and right of it by exchanging data in certain registers. This facilitates the implementation of algorithms that require data from groups of adjacent pixels - each PE can exchange information about the pixel it is currently processing with its neighbors, thus allowing such algorithms to be implemented without duplication of pixel data across IMEM blocks or expensive IMEM-to-IMEM transfers.

In addition to describing the general principles of the IMAP architecture, the paper demonstrates a sample implementation. This IMAP uses 64 eight-bit microprocessors as the PE array, with 4KB of IMEM attached to each PE. The system used for testing incorporated eight IMAPs (for a total of 512 PEs) controlled by a seperate 16-bit processor. Benchmarks show it performing most simple image processing tasks in less than a millisecond, which is ample for real-time processing - a typical real-time application needs to handle video at 30fps, giving it 33ms to process each frame.

2.2 One-Dimensional C

In a later paper, Fujita et al. [2] describe a language intended for use with IMAP, *one-dimensional C* (1DC). This is based on ANSI C89, but with extensions to facilitate data-parallel operation and alleviate the difficulty inherent in manually managing PEs. (Existing data-parallel C dialects were considered, but deemed insuitable due to implicit assumptions of more powerful message-passing facilities in the underlying hardware than IMAP possesses.)

1DC adds a new declaration keyword, *sep*, which is used to declare a variable that is spread across the PEs; storage for the variable is divided among PE IMEM, and operations on it are automatically dispatched to all of the PEs. It also adds new operators for inter-PE communication and result collection by the CPU, and a new flow control statement, *mif*, used to perform operations only on PEs meeting some condition. In each case, the 1DC compiler emits machine code that handles the minutiae of reading and writing IMEM

and PE registers and coordinating PE operations.

3 IMAP-CE

3.1 Hardware Design

IMAP-CE is an IMAP implementation developed by Kyo et al [8]. Unlike the original IMAP design, which envisioned IMAP as a coprocessor attached to a seperate CPU, it integrates the control processor (*CP*) onto the chip. A single IMAP-CE processor, then, consists of 128 PEs with 2KB of IMEM each (divided into sixteen identical tiles of eight PEs each), a single CP, an external memory interface (EXTIF) connected to an external RAM bank, and a bank of shift registers used to hold incoming video data (see Fig. 1).

The video shift registers are connected directly to IMEM; after each line of video data arrives, it is copied into IMEM for processing by the PEs. With 2KB of IMEM per PE, this allows it to process an entire 512x512px, 32-bit image without ever accessing EMEM, at least in principle. Additionally, the shift registers are connected to video *output*, allowing the IMAP-CE to copy data from IMEM back into the VSRs and output it as a video stream - allowing the IMAP-CE to not just collect and report results in memory, but output an annotated or completely transformed version of the original video.

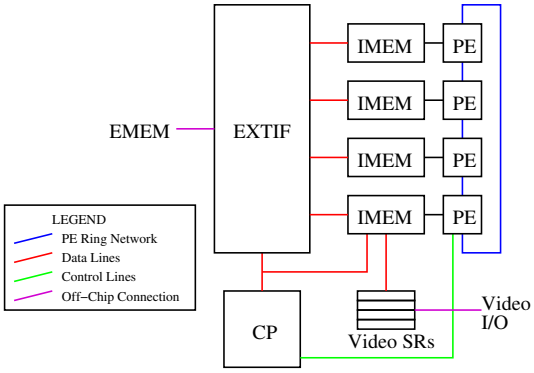

Figure 1: Overview of IMAP-CE internal structure.

When EMEM access is necessary, it is performed via DMA hardware in the EXTIF. It takes only a single clock cycle to initiate a DMA transfer; however, it takes sixteen cycles to transfer a complete row of image data between IMEM and EMEM. For this reason, part of IMEM is set aside as buffers for image transfer, allowing multiple rows of DMA to be queued at once and then processed in the background.

As with the earlier IMAP and IMAP-VISION systems described by Fujita et al., 1DC is used as the programming language for IMAP-CE. The implementation is of course different from the one used by Fujita et al., but the language itself is unchanged.

3.2 Parallelization Techniques

For the purpose of parallelizing existing image processing tasks, they were categorized based on how their memory access patterns could be represented using a *pixel update line*, or PUL. Notionally, the PUL is a line that sweeps across the image, with each pixel it crosses being operated on by a PE as the line crosses it – each column of pixels being

stored in the IMEM of a different PE.

The first access pattern, *row-wise* (see (a) in Fig. 2), is also the simplest: an entire row is processed by having each PE operate on the top pixel in its column, then each PE "moves" down one pixel. Once the entire image is processed, each PE has processed an entire column of the image.

The second, *row-systolic* (b), is used when each PE needs to process a row rather than a column. Unlike row-wise, the initial layout of the PUL is diagonal, not horizontal, with PE 0 operating on the top pixel in its column and PE n operating on the bottom pixel. As before, each PE operates on its current pixel and then moves down; however, before moving, the PE passes its current state one PE to the left using the ring network (with PE 0 sending its results to PE n). Furthermore, when selecting the next pixel, PEs that "fall" off the bottom of the image wrap around to the top. In this way, each PE gradually accumulates the results from an entire row, rather than column, of the image.

The third access pattern, *slant-systolic* (c), is used for operations with simple adjacent-pixel data dependencies. It starts with only a single pixel in one corner being processed; once that is handled, the two adjacent pixels (one below, and one beside) can be processed; and so forth until the entire image is handled, with the number of active PEs starting at 1, peaking halfway through the image, and then falling back off.

The final access pattern, *autonomous* (d), is used when the extent of the region(s) to be processed must be determined as the algorithm runs. In this mode, each PE has a section of memory reserved as a stack of pixels of interest; each operation may push more pixels onto this stack, or the stacks of neighboring PEs. The CP manages this process until all PE stacks are empty. (Do not confuse this with instruction-parallel processing; the PEs are still performing the same operations on the pixels they examine. It is only the choice of which pixels to operate on that varies between PEs.)

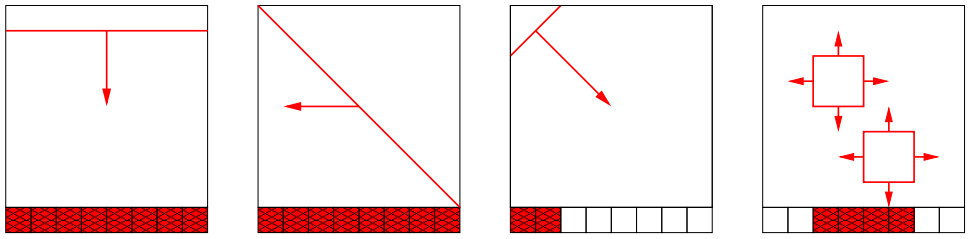

Figure 2: Memory access patterns based on a pixel update line.

3.3 Performance

For performance evaluation, several image processing kernels were run in three different environments: once on the IMAP-CE at 100MHz; once on an Intel Pentium 4 at 2.4GHz, compiled with the Intel optimizing compiler; and once on the same P4, but compiled from the IMAP-CE 1DC source code using an optimizing 1DC compiler (which makes use of the MMX SIMD instructions supported by the P4).

The IMAP-CE proved to be significantly faster at these image processing tasks, demonstrating an average speedup of 8 relative to the C implementation and 3 relative to the 1DC implementation with MMX. Furthermore, it had a sustained power requirement while doing so of only 2W, compared to approximately 100W for the P4.

It was also compared to three other data-parallel media processor chips - the Imagine, MorphoSys2, and VIRAM. In these comparisons it did not far so well; although the most power- and space-efficient, it proved to be between 2 and 10 times slower. This is primarily attributable to its significantly slower clock rate, but also to the fact that the other three processors mentioned are all full 16-bit processors, whereas IMAP-CE still relies on 8-bit PEs. The authors suggest that expanding IMAP-CE to be fully 16-bit as well may be a worthwhile performance improvement.

4 IMAPCAR

4.1 Design

The IMAPCAR design, also described by Kyo et al. [7], is an incremental improvement on IMAP-CE. It does not make the same sort of drastic changes to IMAP-CE that IMAP-CE made to IMAP; rather, it addresses the most easily correctable performance deficiencies in IMAP-CE, and adds the reliability features necessary for it to be safely used as an automotive vision processor.

To alleviate the slowness of EMEM access, as discussed above, the DMA capabilities of the EXTIF were upgraded. The DMA request queue was extended, making it possible to queue up twice as many DMA transfers in the background. Furthermore, scaling and translation support was added - when performing DMA operations on rectangular subsections of the image, the EXTIF is capable of performing simple up- or down-scaling operations and relocating the image region in memory as the DMA transfer occurs, freeing the PEs from the task.

In addition to the DMA upgrades, the number of video shift registers was tripled, allowing it to process three video streams of width 512px, or two video streams of width 640px or 768px, simultaneously. The interconnections among the SRs were also upgraded to support two different patterns for allocating video data among the PEs - one in which each PE gets a set of adjacent columns, and one in which each PE gets a set of columns evenly spaced within the frame. As with the IMAP-CE, the SRs operate on the video clock until a complete row has been buffered, then operate on the system clock during the horizontal-blank period to copy the row into or out of IMEM.

Various reliablity enhancements were also made. The temperature tolerances were improved to meet the -40°C to +85°C range required for automotive use; the vibration tolerances were similarly increased. The SDRAM (dynamic RAM) originally used for EMEM was replaced with SSRAM (static RAM), and the CP instruction cache C$ and EMEM were augmented with four bits of ECC (error correction code) per 32-bit instruction word. Finally, IMEM and the CP data cache D$ were given one parity bit per byte of data. Parity failures, and errors uncorrectable by ECC, will cause the IMAPCAR

to stop execution and raise an exception (ideally, to be handled by the vehicle's master control system).

4.2 Performance

Like IMAP-CE, IMAPCAR proves to be sigificantly faster than general-purpose processors for performing data-parallel image processing tasks. Furthermore - once program code is updated to take advantage of the new region of interest (ROI) scaling features added to the EXTIF - it also proves to be up to three times faster than the IMAP-CE for the same operations. This is due primarily to the increased DMA queue depth and the time saved by performing simple scaling operations during DMA rather than using the PEs for them before or afterwards.

Despite these improvements, the IMAPCAR also consumes less power - 2W maximum, compared to 2-4W for the IMAP-CE - and operates at a lower core voltage (1.2V compared to 1.8V), although the IO and EMEM voltage remains the same at 3.3V.

In real-world use, such as the overtaking-vehicle detection system described by Kyo et al. in 2007 [11], the performance improvements relative to GPPs are not quite as dramatic as the image processing benchmarks would suggest; this is primarily attributable to the fact that in such a system, the IMAPCAR must necessarily spend some time performing serial operations with the PEs idle, losing the huge advantage that is has when performing data-parallel operations - becoming, in effect, a 100MHz processor rather than a 12.8GHz one. Nonetheless, it proves to be nearly three times as fast as a 3GHz GPP, and can perform a complete overtake-detection cycle in approximately 31ms, giving it a comfortable 2ms to spare per frame.

5 IMAPCAR2

5.1 Overview

As noted above, the IMAPCAR's greatest weakness is that when *not* performing data-parallel operations, it effectively operates as a rather slow single-core RISC processor. This is compounded by the fact that most image analysis tasks conclude with region of interest analysis - the inspection of several previously-identified regions of the image - and this operation is generally *not* data-parallel; each region may be using a completely different algorithm, and even when not, the ROI analysis algorithm is generally too complex to be treated as a SIMD operation over multiple regions. As a result, once reaching this stage, the IMAPCAR ends up examining each ROI in serial, relying primarily on the CP alone.

The obvious solution to this is to add some sort of MIMD (multiple instruction, multiple data - ie, true multithreading) support to the IMAPCAR. While this could be done by adding more CPs, the increase in complexity, size, and cost is prohibitive. Instead, a dynamic reconfiguration approach was taken when designing its successor, the IMAPCAR2: the chip can operate in both SIMD and MIMD modes, re-using mostly the same

hardware for both. [4]

This was accomplished by dividing each tile of eight PEs into two groups of four, and then adding additional control circuity to each group (see Fig. 3). This permits each group of four PEs to operate either as four SIMD processing elements, as in the original design, or - using the new hardware - as a single *processing unit* or PU, with capabilities equivalent to that of the CP itself. This new hardware increases the size of each tile by approximately 20% (and the gate count by approximately 10%).

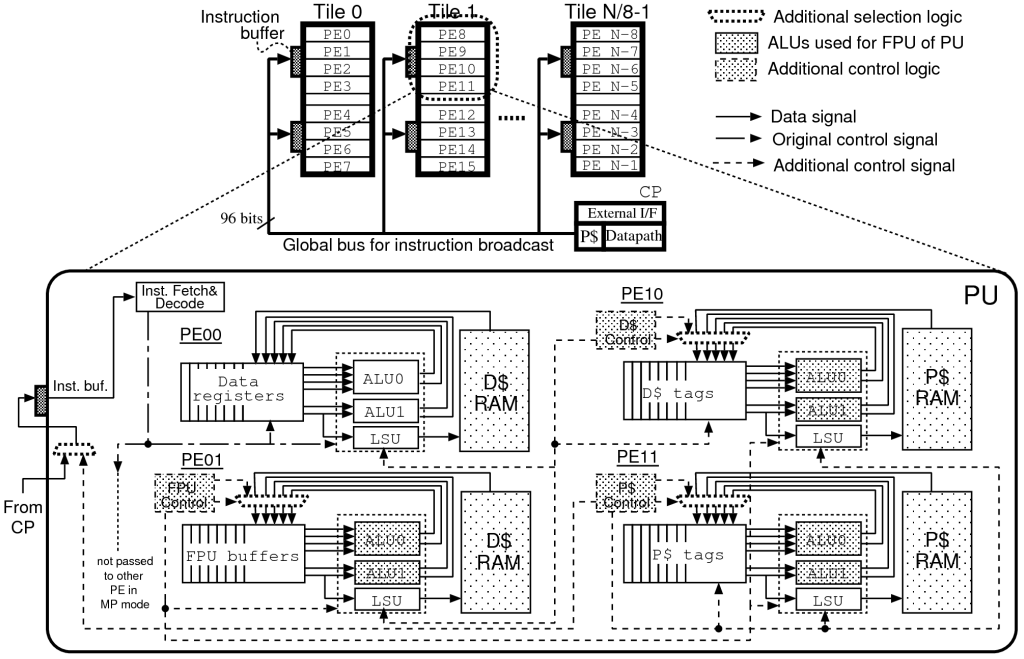

Figure 3: Detail of one half of a reconfigurable PE tile (from [4]).

5.2 Changes in CP and PE Design

To support this SIMD/MIMD mode switching, some drastic changes - beyond the added control circuitry needed to support the mode switching in the first place - were made to the PE design. Not only was the amount of IMEM doubled (from 2KB to 4KB), but the entire datapath was replaced with a duplicate of the one present in the CP, and the instruction decoder revised to support a subset of the CP instruction set (only the CP has instructions for PE control, direct EMEM access, and dynamic reconfiguration).

As a result of this, each PE now has 23 16-bit general purpose registers and six special-purpose registers; one load/store unit, for accessing IMEM; and two arithmetic/logic units. Each ALU supports up to two operations per clock, provided they don't overlap in hardware usage; in ideal situations, the PE can now execute five instructions per clock - two arithmetic/logic operations and one load/store.

The instruction fetcher and decoder used by both the CP and the PEs has been upgraded to match; the shared instruction format is variable-width (16 or 32 bits per instruction),

but has provisions for packing up to five 16-bit instructions that share common elements into a single 96-bit instruction that can be fetched and dispatched to the PE array in a single clock cycle.

Finally, the ring network used for communication among PEs has been renamed the *N-ring*, and two additional ring networks have been added to the PE array, the *M-ring* and *C-ring*. The M-ring connects all of the PUs to a DMA controller attached to the CP, and is used for copying data between EMEM and the PU instruction and data caches; the C-ring connects all of the PUs to each other (and to the CP), and is used for message-passing between PUs. Unlike the N-ring, the C-ring contains additional selector hardware that lets it "skip" tiles, meaning that messages can be passed around the ring at a rate of one clock cycle per tile, rather than the one clock per PE that the N-ring is limited to; unlike the M-ring, the C-ring can also be used for message passes between PEs.

5.3 Dynamic Reconfiguration

The chip supports three modes: *SIMD mode*, in which all of the PEs are active; *mixed mode*, in which at most half of the PUs are active (the ones in the "lower half" of each tile, specifically), while the remaining 64 PEs (the "upper half" ones) are also in use; and *MIMD mode*, where more than half of the PEs are being used as PUs. The chip returns to SIMD mode when all PUs report that they have completed execution.

Reconfiguration for MIMD operation is initiated by a *forkinit* instruction issued by the CP, which activates the connective circuitry in the selected PE groups. The CP then uses the *forkp* and *forkd* instructions to copy data into the PU instruction and data caches, respectively, and then finally the *fork* instruction to start the PU running.

When operating in PU mode, nearly all of the resources of the four PEs are used to make up the PU. The datapath of PE0, being nearly identical to the CP datapath, is re-used entire as the PU datapath. The IMEM blocks are combined to make up the 8KB instruction and data caches, and the registers from PE2 and PE3 are used for cache tags. Finally, the registers from PE1 and the ALUs from PEs 1 through 3 are combined to form the FPU.

In this mode, the IMEM is gone, and the data and instruction caches behave much more similarly to the caches found on GPPs - when the code executing on the PU attempts to access a page of memory, it is transparently copied from EMEM into the appropriate cache if not already present. However, this is a relatively slow operation, and may copy surrounding memory that is not required by the PU. To alleviate this, instructions are provided to explicitly transfer regions of memory between EMEM and the instruction and data caches. These instructions are doubly important, as the PUs have no cache coherency hardware - changes to the cache will not automatically be written back to EMEM, and changes to EMEM will not automatically be copied into the cache. Thus, explicit transfers must be used to update external memory, and fetch updates from it.

5.4 Programming Model

The SIMD programming model is identical to that of the IMAPCAR, using 1DC as the programming language. This section will therefore only discuss the programming model for the new MIMD mode.

In MIMD mode, the 1DC data-parallel extensions are unavailable to the PUs; they are programmed in plain ANSI C. Communication with other PUs and with the CP is performed with *send* and *recv* primitives, which implement one-to-one, one-to-many, and one-to-any message passing using the C-ring; communication with main memory happens automatically on a cache miss, or can be triggered explicitly using *roiread* and *roiwrite*, and uses the M-ring.

At the application level, a pthreads-compatible API is exposed, with synchronization primitives such as semaphores implemented using message passing. However, this has one major drawback: pthreads is designed around the assumption of shared memory. While this is a safe assumption in single-core systems, and multi-core systems with cache coherency, the IMAPCAR2 does not have coherent caches. Thus, any communication more complex than simple synchronization (implemented internally with the message-passing operations) must be done using low-level message passing or ROI transfer operations; updating shared memory structures will not work, as once a page is copied into cache, it is no longer shared.

5.5 Performance

The IMAPCAR2 proves to be approximately twice as fast as the IMAPCAR, even when operating solely in SIMD mode. This is attributable primarily to the increased capabilities of the PEs (twice as much IMEM, 16-bit datapath, and more efficient instruction dispatch), all of which increase the amount of work that can be done per instruction and reduce the overall instruction count of each operation. An increase in the clock speed to 150MHz is also a major factor. [10]

In MIMD mode, the speedup is greatly dependent on what operations are being performed. For heavily parallelizeable operations that previously had to be executed solely on the CP, speed improvements of up to 10x were seen when using all 32 PUs. In practice, performance tends to increase linearly up to 8 PUs; beyond that, M-ring and DMA contention becomes a serious bottleneck as all of the PUs attempt to access EMEM, and diminishing returns set in rapidly. It is likely that algorithms less dependent on EMEM access could show performance improvements approaching the theoretical maximum.

5.6 Future Work

The IMAPCAR2's biggest weakness is the mismatch between the environment for MIMD programming presented to the programmer, and the actual capabilities of the chip. The use of pthreads as an API encourages the programmer to use a shared-memory design, but in practice any use of shared memory must be done with great care and explicit synchronization between cache and EMEM.

IMAPCAR2 could greatly benefit from a message-passing multithreading built on top of, or replacing, the pthreads API. In practice, IMAPCAR2's cache-acoherent design and message-passing primitives more closely resemble the structure of a modern high-performance computing cluster than they do a shared-memory symmetric multiprocessing system, and for this reason Kyo et al. suggest MPI [9] or something like it as a suitable API for use of IMAPCAR2. Pilot [1], a much simpler API traditionally implemented on top of MPI, would also be a suitable choice, and a Pilot implementation on IMAPCAR2 is likely to be the subject of my thesis.

6 Conclusion

The IMAP architecture has proven to be a highly efficient and effective architecture for image processing tasks. Despite its simplicity, all implementations of it have performance that completely outclasses GPPs in this problem domain, and because of it, low power requirements.

However, while ideal for data-parallel operations, IMAP's single-core design hurts it severely when performing instruction-parallel operations such as region-of-interest inspection. IMAPCAR2 addresses this by allowing SIMD processing elements to be reused as additional MIMD cores as needed.

IMAPCAR2's greatest weakness is its MIMD programming model, which uses a shared-memory API, despite the fact that the underlying hardware more closely resembles a local-memory system with message-passing. Development of a message-passing API that more closely corresponds to the IMAPCAR2's capabilities could be a fruitful area for further research.

References

[1] Carter, J. D., Gardner, W. B., and Grewal, G. The pilot library for novice MPI programmers. *SIGPLAN Not. 45* (January 2010), 351–352.

[2] Fujita, Y., Kyo, S., Yamashita, N., and Okazaki, S. A 10 GIPS SIMD processor for pc-based real-time vision applications — architecture, algorithm implementation and language support. In *Proceedings of the 1997 Computer Architectures for Machine Perception (CAMP '97)* (Washington, DC, USA, 1997), CAMP '97, IEEE Computer Society, pp. 22–.

[3] Fujita, Y., Yamashita, N., Okazaki, S., Cantoni, V., Lombardi, L., Mosconi, M., Savini, M., and Setti, A. A 64 parallel integrated memory array processor and a 30 GIPS real-time vision system. In *Proceedings of the Computer Architectures for Machine Perception* (Washington, DC, USA, 1995), CAMP '95, IEEE Computer Society, pp. 242–.

[4] Kyo, S., Koga, T., Hanno, L., Nomoto, S., and Okazaki, S. A low-cost mixed-mode parallel processor architecture for embedded systems. In *ICS '07: Proceedings of the 21st annual international conference on Supercomputing* (New York, NY, USA, 2007), ACM, pp. 253–262.

[5] Kyo, S., Koga, T., Okazaki, S., and Kuroda, I. A 51.2 GOPS scalable video recognition processor for intelligent cruise control based on a linear array of 128 4-way VLIW processing elements. *IEEE Journal of Solid-State Circuits 38*, 11 (2003), 1992–2000.

[6] Kyo, S., and Okazaki, S. In-vehicle vision processors for driver assistance systems. In *ASP-DAC '08: Proceedings of the 2008 Asia and South Pacific Design Automation Conference* (Los Alamitos, CA, USA, 2008), IEEE Computer Society Press, pp. 383–388.

[7] Kyo, S., and Okazaki, S. IMAPCAR: A 100 GOPS in-vehicle vision processor based on 128 ring connected four-way VLIW processing elements. *J. Signal Process. Syst. 62* (January 2011), 5–16.

[8] Kyo, S., Okazaki, S., and Arai, T. An integrated memory array processor architecture for embedded image recognition systems. In *ISCA '05: Proceedings of the 32nd annual international symposium on Computer Architecture* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 134–145.

[9] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard, Version 2.2.*

[10] Nomoto, S., Kyo, S., and Okazaki, S. Performance evaluation of a dynamically switchable simd/mimd processor by using an image recognition application. *Information and Media Technologies 5*, 2 (2010), 366–375.

[11] Sakurai, K., Kyo, S., and Okazaki, S. Overtaking vehicle detection method and its implementation using IMAPCAR highly parallel image processor. *IEICE - Trans. Inf. Syst. E91-D*, 7 (2008), 1899–1905.