Intelligent Co-operative PIM Architecture for Image Analysis and Pattern Recognition

Zaki Ahmed, Reza Sotudeh, D. M. Akbar Hussain *

Abstract—**Computer memory systems are increasingly a bottleneck limiting application performance. Processor-In-Memory (PIM) architectures, which capitalize on merging the processing unit with its memory unit on the same chip [1], promise to remove this limitation by providing a tremendous increase in available memory bandwidth and significant reduction in memory latency for a specific class of computing that deals with significant amount of data with relatively simple operations. Many image processing and pattern recognition applications fall into this category. In this paper, some key characteristics with design philosophy of an intelligent co-operative PIM architecture (CIM) are discussed, and examples of image analysis algorithm that can run on it are given.**

Keywords: Co-operative Intelligent Memory (CIM), Processor-in-Memory (PIM), Shared memory, CPU_major, CPU_minor, Observer, Task optimizer.

1 Introduction

Generally, image processing algorithms have implemented using parallel computation or special purpose computing engines. The major motivation behind these implementations, either by using a co-processor or vector/array processor, is to cope with the large computation required by these algorithms, in real time, with the data-intensive computation loops. Data intensive applications require demandingly high number of memory accesses which have operational characteristics that include a significant amount of memory-to-memory type of instructions. This is in contrast to the usual statistically distributed register-to-register, memory-to-register and memory-to-memory instructions that are found in most programs. It is important to be able to expedite process with data intensive computation loops, inherent in many applications, especially image processing and pattern recognition. These applications usually input and output significant amounts of data which are processed with relatively simple operations. The algorithms deployed in these applications involve data intensive, iterative and most often, highly parallel tasks [2]. For class of tasks which are heavily reliant on memory to memory iterative instructions [3][4][5][6][7][8], the concept of Co-operative Intelligent Memory (CIM) was developed by the intelligent system group of University of Hertfordshire, based on previously developed Co-operative Pseudo Intelligent Memory (CPIM) to reduce the performance gap [9] between the processor and memory by partitioning computation through dividing workload between major (non-recursive) and minor (recursive) CPUs.

*Manuscript submitted February 3, 2010, Dr. Zaki Ahmed is Principal Engineer at The Pakistan Institute of Laser and Optics, Islamabad Pakistan (Email: zaki786@ieee.org), Dr. Reza Sotudeh is Professor and Head of School for ECEE at The University of Hertfordshire, UK (Email: r.sotudeh@herts.ac.uk), Dr. D. M. Akbar Hussain is Associate Professor at The Aalborg University, Denmark (Email: akbar786@ieee.org)

2 Philosophy Behind the Concepts

Both architectures (CPIM, CIM) fall into the category of generalized intelligent system, designed for applications which host inter-independent iterative loops. The system possesses intelligent by observing activities that take place between main processor (CPU_major) and its main memory. This is done in order to determine the nature of operations or tasks performed on data entities residing in main memory and bound by fixed memory partition or frame. The intelligence develops in time as more observations lead to better formation of task identification and once the threshold for recognition is reached, observation ceases and further actions ensue. This clearly requires that the input information (e.g. frame, segments....etc) remains resident within the same memory boundaries (static Locale), irrespective of the dynamic nature of the content (Dynamic Content). It is under this condition that the system exhibits performance gains since it only invests a single learning phase to acquire the knowledge about the Static-Locale-Dynamic-Content type of data structure. Outside this condition a new learning cycle must be entered every time a new data structure profile is executed and hence the benefits will diminish. Any reference to Intelligence in the context of our PIM architecture is limited to the definitions that an intelligent system (IS) is a system which learns how to act towards a certain situation in order to reach its objectives by using experiences and knowledge gained previously.

From the above statement, we can conclude that an intelligent system has two fundamental characteristics learning and serving. Typically, an IS achieves its objective through knowledge and experience, which it builds after moments of its existence. It includes the situation that

ISBN: 978-988-18210-5-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

occurred, the action done, and the results), which is acquired through a learning process. This experience sometime comes from static means, we can develop a, ROM based, system that can act towards a defined situation to reach its objective. This type of system learns how to act towards a situation by an external teacher, ROM program, and meets the objective by the knowledge provided by the external teacher. An example of this is where the ROM contents are merely used as a lookup table to guide the process toward a predefined output based on a known input scenario. This is primarily knowledge driven and the knowledge does not expand beyond the predefined boundaries coded in the lookup table. This is an open loop system with repeated success (correct output) does not build confidence factor or improve future decisions or outputs because there is no feedback to enhance/reward future decisions.

A more sophisticated system would use experience together with knowledge to improve decision making and evolves intelligence through accumulation of success or failure metrics from the tasks performed, like people learning by example. This is a closed-loop system since a confidence factor is built by reflecting on the positive or negative outcome from the previous decisions/tasks. Information from the output is weighted and returned back to form part of the input that drives the next decision making process. Our CPIM system uses the open-loop scenario, means that once the re-assimilated code is linked and executed by the CPU_major, it continues on its non-iterative job. When the by-pass is encountered, the vector components will be loaded into the CPIM registers. This transfer of information about the iterative loop tells the CPIM system how to act towards a certain situation. Information about the desired loop is provided to the system by an external teacher, which is a special program, Taskoptimizer.

The information provided cover the address range that include starting address, ending address (address of the operand block) and address of destination where the computed results are to be stored, with job size, job nature and the number of iterations in the loop. As CPIM is an application specific block and deals with highly iterative memory-to-memory tasks, number of iterations can be used as a job size. Once all the registers have been initialized, CPIM interrupts CPU_major by using an external unit located outside the CPIM namely, Interrupt Management Unit (IMU), for the transfer of related data from main to shared memory. Thereafter, application specific block commences the designated task. During the course of executing the same program, where the corresponding CPIM registers are re-initialized due to the work load or task partitioning technique, by-passed iterative loop is replaced by the extracted vectors and the remaining parts of the loop are filled with NOP, to keep the program sequence unchanged. However, the system enjoys the benefits of CPIM with the computational results obtained during learning cycle to enhance its performance over CPU_major by offering highly optimized job processing algorithm as well as its ability to be clocked at higher speed.

The notable characteristic of CPIM is that whenever the CPIM registers are filled with new entries or data set changed, task specific processor (CPU_minor) performs its job. A widely accepted assumption which is echoed in the above is that an intelligent system has ″learning″ and ″serving″ phases. Vector loading into the CPIM registers demonstrate a learning phase, the application specific block trained for a particular situation. Due to re-initialization of CPIM registers during the course of executing the same program, serving stage, with same data set shows that the system is unable to use experience and knowledge gained previously. However, in the presence of new data set, when a taught situation is detected (iterative loop), it partially (re-initialization of CPIM registers) behaves like an intelligent system, with a new set of results. This new set of results or output partially exhibits the use of experience and knowledge gained previously for a particular situation.

The CIM uses learn and serve policy, which follows the closed-loop doctrine alluded earlier. Migration from CPIM to CIM needs an additional vector, vector instruction block. The vector instruction block, corresponds to the start and end address of the bypassed loop. The major characteristics that make CIM distinctive from the existing PIM systems, is its run time learning capability to gather knowledge for future work. During the first execution cycle, an additional hardware unit called ″observer″ collects information about the desired loop. Eventually information related to the vectors; characterize iteration. The information collected cover the address range that include: starting address, ending address, destination address, and address of instruction block that corresponds to the situation (iterative loop) with the vectors job size (number of iteration in the loop) and job nature. Once the task is completed the vectors component loaded into the application specific block (CPIM) and the related data transfer from main to shared memory.

The vector instruction block is used to apply bypass. The bypass effectively removes the set of instruction related to the situation (iterative loop). Vector loading into the CIM registers demonstrates a learning phase, the application specific block trained for a particular situation. During the course of executing the same program, serving stage, due to use of bypass with same data set or in the presence of new data set system demonstrates the ability to use experience and knowledge gained previously. Thus, the proposed system obeys the basic rules developed for an intelligent system.

ISBN: 978-988-18210-5-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

3 Architectures Description

The CPIM and CIM architectures are shown in figure 1 and 2 respectively. The main CPU, CPU_major, has a conventional architecture and poses no real design constraints on the CPIM architecture and backed up by a deep cache hierarchy and suffers high latency to access memory. The enhancement called CPIM, introducing a new block of memory (shared memory), shared through arbitration between CPU_major and task specific processor, CPU_minor, that consists of a small computational unit performing iterative processing and an Iteration Control Unit (ICU). ICU provides an instruction format for the CPU_minor, consists of a set of registers, namely address register (addr-register), job size register (jobsize-register), job nature register (jobnature-register) and destination register (dest-register).

A detailed discussion of the CPIM architecture with distribution of workload and code optimization technique can be found in [9][10]. The CIM architecture (figure 2) differs from CPIM in terms of approach; instead of Von-Neumann (instruction and data are stored in a single memory) it requires a Harvard approach towards memory (Separate memory for instruction and data). This approach may simplify read / write mechanism, particularly as programs are normally read during execution, while data might be read or altered. Also establish a path for the extraction of vector components by monitoring the activity operating on the address and data buses. The detection of iterative tasks, conducted by an additional hardware unit called *observer* having additional knowledge of the location of specific logic blocks (CPIM) with reference to their computational capability.

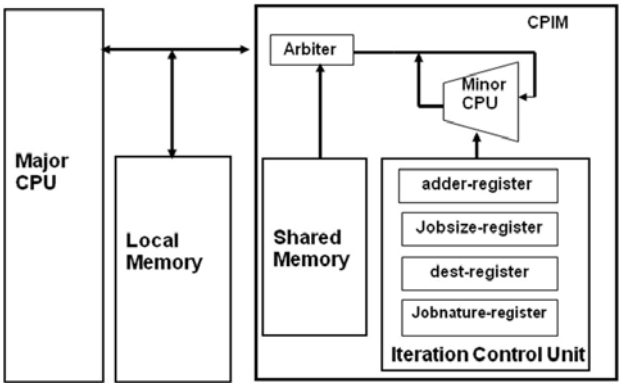

Figure 1: CPIM Architecture

The following jobs are performed by the observer;

- Extraction of vectors that characterize the iteration.
- Transfer of vector components with the related set of data into specific logic block.
- Removal of selected / corresponding iterative loop from the main stream.

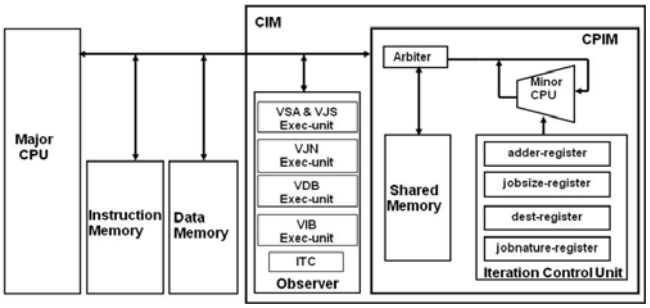

Figure 2: CIM Architecture

A detailed discussion of the CIM architecture with design methodology, acceleration and speedup parameter can be found in [9][10].

Our CPIM and CIM architectures have the following characteristics:

- The memory capacity is large enough to hold large data frames synonymous with high resolution image frames.
- Eliminates the overhead associated with the time it takes to fetch and execute the instruction in a specific program loop.
- No need for special instructions as required in the case of co-processor.
- CPU_major (main CPU) can continue with other operations while the CPIM is completing its allocated task.

The major characteristics that make CIM distinctive from the existing PIM systems, is its learning capability to gather intelligence from the current program execution profile.

4 Implementation Examples

Emerging computer applications in multimedia, specifically in image processing are heavily reliant on memory-to-memory iterative process (memory intensive). Data bandwidth, which is critical for DIP applications due to the memory-intensive nature of most DIP algorithms, exploited in intelligent memory architectures. Implementations of some low-level image processing operations are described below. Low level image processing is suited for two reasons. First, the same computational operation is applied to a number of pixels data. Second, most low-level image processing operations are computed over small pixel neighborhoods. Implemented scenarios with algorithms are described below. These scenarios have been simulated using Mentor graphics with PS6.2 and implemented on a SPARTAN II, XC2S300E-6PQ208C FPGA using the NEXAR 2004 EDS environment.

ISBN: 978-988-18210-5-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

These implementation shows that the DIP is one of the areas, where applications require high bandwidth, low latency access to image data, and most often decomposed into simple iterative operations/loops. Thus, computing in memory or intelligent memory architectures best fit for co-operative processing, executing the functions that they are optimized for, while leaving functions that are mostly serial and compute intensive to the main processor (CPU_major).

4.1 Brightness Adjustment

Brightness adjustment is done by adding or subtracting a value to or from each pixel in order to shift the pixel intensity by the specified number of grey level. Algorithm shows the brightness adjustment for a given image.

Algorithm Brightness adjustment

*/ * Brightness adjustment * /*
for pixel=0 to N−1 pixels
add or subtract a value to or from the pixel;
if pixel is greater than grey-level limit
saturate the pixel;
end for

4.2 Low Pass Filtering

Apart from poor contrast, images may contain random pixels that have values higher or lower than what they should be. One way to reduce this type of noise is to replace the value of each pixel with the weighted average of its neighborhood pixels [11].

Algorithm Weighted Average Filter (Neighborhood Pixels)

*/ * Weighted Average Filter * /*
for pixel=0 to N−1 pixels
find average of 3×3 *neighborhood pixels;*
replace pixel value with this average;
end for

4.3 Edge Detection

The real-time detection of edges in digital images is widely used in robot navigation, industrial inspection, and virtual reality. Convolution is often used to provide an edge filter. In essence, convolution provides gradient operators in both the horizontal and vertical directions and the final edge image is formed by marking pixels of high gradient intensity [12]. The Prewitt and the Sobel are two of the most commonly used 3×3 neighborhood edge detection operators. These compute both the magnitude and direction of the edge.

Algorithm Prewitt Edge Detection

*/ * Prewitt Edge Detection * /*
for pixel=0 to N−1 pixels
find the gradient images in the x y direction;
find a common pixel wise norm;
find he output of the norm block using maximum norm;
find the sum of the absolute value;
find the difference with the threshold limit
if pixel value is greater than threshold saturate it to 255;
end for

The Prewitt operator has been used as an example. After an image frame is loaded into shared memory, the CPU_major simply instructs the CPIM to perform the multiplication-accumulation (MAC) for each pixel, figure 3 shows this scheme.

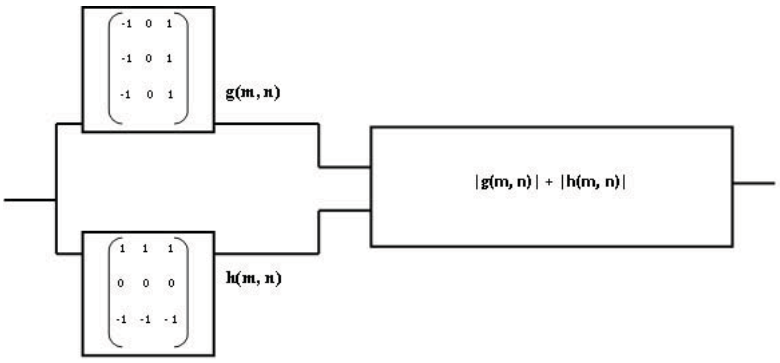

Figure 3: Edge Detection by Prewitt Gradient Masks

4.4 Morphological Processing

Morphology is related to shape, and digital morphology is a way to describe the shape of a digital object. The two basic morphological operations are erosion, in which pixels matching a given pattern are deleted from the image and dilation, in which a small area about a pixel is set to a given pattern.

Algorithm Erosion of a Digital Image

*/ * Binary Erosion * /*
Find start and end pixels (i_s, j_s) *and* (i_e, j_e)
for pixel=starting value to end value
Place Structuring element (SE) over the image
if corresponding pixel in image agree
set pixel in result image;
else
do nothing;
end for

Algorithm Dilation of a Digital Image

ISBN: 978-988-18210-5-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

/ ∗ Binary Dilation ∗ /
Find start and end pixels (i_s, j_s) *and* (i_e, j_e)
for pixel=starting value to end value
Place Structuring element (SE) over the image
if corresponding SE pixel is 1
set pixel in result image;
else
do nothing;
end for

The computation procedure is similar to the edge detection mentioned above. The difference is comparison−merge operation instead of the multiplication−accumulation operation.

5 Conclusion

Both CPIM and CIM architectures have been described; using two level hierarchical architectures. CPIM uses a pre-compilation task optimization methodology for the workload distribution between CPU_major and CPU_minor. In the CIM, the whole process is divided into learning and serving stages. During learning stage, CPU_major works on both iterative and non-iterative parts of the task. However, observer monitors the activities taking place on the address and data buses. If an iterative activity is detected, after a qualifying threshold (Job size), observer records the vectors into the specific registers. Once the learning stage is completed, Information Transfer Control, a sub part of the observer transfers all the recorded information with the related set of data involved in the iteration into specific logic blocks placed in the memory system. During serving stage, When the CPU_major reiterates the same program and encounters the iterative loops; the CPU_major will stop executing task. The CPU_minor does the task that the Major leaves. Thus, CIM serves the system through the knowledge gained during the learning stage.

Application examples described above, show that the DIP is one of the areas that seems well-matched to the computing in memory design. Image processing applications generally require high bandwidth, low latency access to image data, and generally decomposed into simple iterative operations. For this reason, computing in memory or intelligent memory architectures best fit for co-operative processing, executing the functions that they are optimized for, while leaving functions that are mostly serial and compute intensive to the main processor (CPU_major). Hence, the described architectures have the potential for scaling up to tackle more demanding jobs that exhibit frequent and intense program locality behavior. This, directly enhance the capability of security systems that most often use image processing algorithms.

References

[1] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas and K. Yelick. A Case for Intelligent RAM: IRAM. IEEE Micro, April 1997.

[2] Y. Kang, J. Torrellas and T. S. Huang, An IRAM Architecture for Image Analysis and Pattern Recognition. 14th International Conference on Pattern Recognition, 1998.

[3] M. Oskin et al., Active Pages: A computation model for intelligent memory, IEEE, 1999.

[4] Y. Kang et al., FlexRAM: Towards an intelligent memory system, ICCD, Oct 1999.

[5] J. Darper et al., The architecture of DIVA processing in memory chips, ICS, June 2002.

[6] A. Saulsbury et al., Missing the memory wall: The case for processor/memory integration, ICSA,May 1996.

[7] D. Burger et al., Memory bandwidth limitations of future microprocessors,I SCA, Aug 1996.

[8] K. Mai et al., Smart memories: A modular reconfigurable architecture, ISCA, June 2000.

[9] Zaki Ahmad Co-operative Intelligent Memory, PHD thesis, University of Hertfordshire, United Kingdom, 2007.

[10] R.Sotudeh, Z.Ahmad, F.Bensaali Intelligent Co-operative Processor in Memory Architectures The Mediterranean Journal of Electronics and Communication, Vol. 3, 2007, pp 17-30.

[11] R. Boyle and R. Thomas Computer vision: A first course, Blackwell Scientific Publications, 1988.

[12] R. Dougherty and A. Laplante, Introduction to REALTime IMAGING, SPIE optical Engineering Press, 1995.

ISBN: 978-988-18210-5-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)