Dynamic Co-operative Intelligent Memory

Xiaoyong Wen, Faycal Bensaali and Reza Sotudeh
School of Electronic, Communication and Electrical Engineering
University of Hertfordshire, Hatfield, United Kingdom
{x.wen, f.bensaali, r.sotudeh}@herts.ac.uk

Abstract

As semiconductor technology advances, the performance gap between processor and memory has become one of the major issues in computer design. In order to bridge this gap, many methods, for example, cache, Massively Parallel Processor (MPP) and interleaved memory have been developed. However, computer system performance fails to make reasonable improvement in the data intensive applications, due to the long latency and limited bandwidth. In a continuing effort to bridge this widening gap between processor and main memory speed, a new concept called Processor-In-Memory (PIM) is introduced, which capitalises on merging the processor unit with its memory unit on the same chip. Several architectures based on this concept have been proposed such as, IRAM, Active Pages, FlexRAM, Computational RAM, etc. The most immediate benefit provided by these architectures is the increased on chip bandwidth and low memory latency. These architectures have some limitations such as their adaptability, scalability and cost-effectiveness. In this paper, a Dynamic Cooperative Intelligent Memory (DCIM) architecture is presented, where some of those limitations are bridged.

1. Introduction

In the past 30 years, an exponential rate of improvement has been witnessed in semiconductor technology. The processor performance increases at a rate of 60% per year while the memory performance increases just 10% per year [1]. This situation causes a 50% growing gap between processor and memory in the performance. This imbalance has become one major bottleneck in further improving the computer performance.

Some traditional approaches have been developed to reduce the widening gap between processor and main memory speed, such as cache, Massively Parallel Processor (MPP), multithreading, interleaved memory and dynamic access ordering. However, the long latency and limited bandwidth are still major bottlenecks in those solutions. Advances in VLSI technology are enabling the processor-memory integration to bridge the processor-memory performance gap. It is also a key driver in the innovation of a new concept called Processor-In-Memory (PIM). The PIM architecture incorporates computational units and control logic directly on the memory to provide immediate access to the data. It offers the promise of high performance though a notable reduction in access latency and dramatic increase in available memory bandwidth for a specific class of computing that deals with significant amounts of data processed with simple or complex operations.

Reconfigurable hardware devices in the form of Field-Programmable Gate Arrays (FPGAs) have been proposed as viable system building blocks in the construction of high performance systems at an economical price. Given the importance and the use of PIM-based systems in scientific computing applications, it seems an ideal candidate to harness and exploit the advantages offered by FPGAs including flexibility and programmability. To increase their flexibility, recent FPGA devices provide new fixed circuit functions. Since they are already committed to the silicon die at manufacture time, there is no added cost in circuit area if a designer chooses to use them. Furthermore, a single FPGA device now contains enough logic blocks to hold multiple units that can perform computation concurrently.

It is the aim of this paper to propose a Dynamic Co-operative Intelligent Memory (DCIM) FPGA-based system with the potential to implement real-time task partitioning capability. Unlike existing PIMs, the concept of dynamic reconfiguration will be exploited in proposed DCIM architecture.

0-7695-3110-5/08 $25.00 © 2008 IEEE
DOI 10.1109/DELTA.2008.33

The structure of the rest of this paper is as follows. Related work is presented Section 2. The proposed DCIM architecture and its description are presented in Section 3. Section 4 is concerned with the DCIM working procedure. Concluding remarks and future work are given in Section 5.

2. Related Work

This section takes a closer look at the most recent architectures and systems based on the PIM concept.

Due to VLSI technology scaling demand, the future computing devices will be narrowly focused to achieve high performance and high efficiency and also target the high volumes and low costs of widely applicable general purpose designs. To achieve those requirements, the Smart Memory [2] has been proposed. A Smart Memory tile consists of a reconfigurable memory system, a crossbar interconnection network, a processor core and a quad interface. The network also connects to high speed links on the pins of the chip to allow for the construction of multi-chip systems. In the Smart Memory, the user can program the wires and the memory, as well as the processors. This let the user configure the computing substrate to better match the structure of the applications, which greatly increases the efficiency.

The Date IntensiVe Architecture (DIVA) system [3] combines PIM memories with one or more external host processors and a PIM-to-PIM interconnect. DIVA increases memory bandwidth though two mechanisms. First, the selected computation is performed in memory to reduce the quantity of data transferred across the processor-memory interface. Second, providing communication mechanisms called parcels is to move both data and computation throughout memory, further bypassing the processor-memory bus. DIVA specially supports acceleration of important irregular applications, such as sparse-matrix and pointer-based computations.

System Level Intelligent Intensive Computing (SLIIC) Quick Look (QL) architecture described in [4], is a single board PIM-based multiprocessor with programmable interconnection. It consists of eight Commercial Off-The-Shelf (COTS) PIM chips and two FPGA chips. The two FPGAs can be used for two purposes: first, they provide a flexible interconnect network between the processors; second, they provide programmable logic that can be used for processor synchronization. The SLIIC QL board was designed to promote the investigation of possible benefits of PIMs by using current COTS PIM technology [4].

In [5], a low complexity PIM architecture for image and video compression is presented. It consists of Processing Elements (PEs), where two are combined to form a subPE unit, a Discrete Cosine Transform (DCT) and a Discrete Wavelet Transform (DWT) hardware units. The PEs and subPEs implement 8-bit and 16-bit operations, such as addition, subtraction, and multiplication. The DCT and DWT hardware are control units for determining the performed operations and storing intermediate results. The architecture gains up to 40% higher throughput per watt when executing DCT and DWT and occupies as little as 0.9% area compared to a commercial digital signal processing.

To support the data-intensive applications, the parallel PIM architecture (PPIM) is proposed [6]. It is based on a distributed data-parallel architecture with limited support for control parallelism. PPIM is a distributed multiple-SIMD architecture which consists of a small number of controllers that broadcast instructions to all the PEs. Every PE can choose to receive its control from any controller. Each PPIM processor has a 16Mbits memory and each PE has a 256Kbits memory. The PE has the similar SIMD architecture as the Computational RAM (C-RAM).

A cooperative intelligent memory was developed within our SoC group, based on a previously developed Cooperative Pseudo Intelligent Memory (CPIM) architecture [7]. Both architectures use a hierarchical two level CPU structure referred to as CPU_major and CPU_minor. The CPU_major has a conventional architecture while CPU_minor is a task specific processor dealing with highly iterative memory-to-memory processing. CPIM uses a pre-compilation task optimization process to determine the division of work between CPU_major and CPU_minor. In the CIM system, the whole work principle is divided into learning and serving stages. In the learning stage, the Major processor executes the program from memory. Then a special unit called Observer detects the iterative loops and the corresponding information about loops is recorded. In the serving stage, when the processor reiterates this program again and encounters the iterative loops, the processor will stop executing job. The Minor CPU does the job that the Major leaves. By this way, the CIM can reduce the Major CPU execution time and improve the computer performance [7].

Although existing PIM systems can improve the computer performance, they have some limitations. Three major limitations can be identified as follows:

1. Each PIM system is fixed in its application, which limits its adaptability, scalability and cost-effectiveness.
2. A pre-processing stage is required to partition applications between a processor and an intelligent memory system, which introduces software overhead.
3. The capability of PIM is limited as the effectiveness of the CPU for computation intensive tasks is largely offset by its administration burdens, i.e. task partitioning, logic-in-memory allocation, coordinating between the CPU and others etc.

To overcome those limitations, a new PIM system called DCIM is introduced and presented in the next section.

3. Proposed Architecture

Based on the existing work limitations, the main objectives of this work can be summarized as follow:

- To explore a self-learning and self-acting, truly intelligent PIM system for the improvement of processor-memory performance.
- To explore the feasibility of PIM platform on a dynamically reconfigurable System-on-Chip (SoC) in order to achieve intelligence, improve adaptability, eliminate scalability bottleneck and reduce on-chip energy consumption and manufacturing cost.
- To develop a methodology for real-time task partitioning and allocation to the logic-in-memory in order to eliminate the pre-processing overhead.
- To develop a logic allocation and scheduling algorithm for better utilisation of the Logic-In-Memory (LIM) and for better performance of the dynamically reconfigurable system.
- To publicise methodologies/algorithms above in order to promote the use of reconfigurable computing platforms more extensively.
- To develop benchmarking specifications to enable performance comparisons for relevant applications using the above techniques.

Figure 1 shows the building blocks of the proposed DCIM.

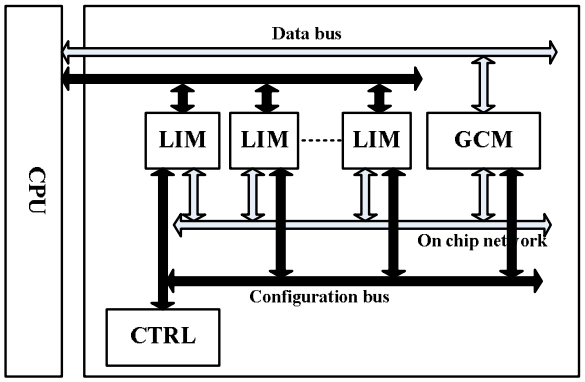


Figure 1. DCIM building blocks

The proposed DCIM consists of a Global Coordinating Memory (GCM), a Control Logic (CTRL), a Logic-In-Memory (LIM) and a external CPU.

The LIM and GCM communicate through an on-chip communication network which is an important element on this system to avoid the congestion. Compared to the conventional network topologies, such as bus, ring, star, tree, mesh and crossbar, on-chip communication network reduces the latency, power consumption and cost [8, 9]. A dedicated bus for data transfer is directly connected from CPU to GCM that can improve data transfer efficiently. Configuration lines are used between LIMs and the CPU and CTRL unit.

Control Logic (CTRL)

The Control Logic acts as a coordinator, responsible for all administration jobs in the system except task partitioning. The jobs include uploading the LIM configuration information, real-time allocation the DCIM resources and coordination between the CPU and other units in the system.

Logic-In-Memory (LIM)

The LIMs are the tasks-specific functional logic to fulfil data-intensive tasks. LIM configurations are initially held in an external system and instantiated after they are loaded into system; the uploading is via the configuration data bus.

Global Coordinating Memory (GCM)

GCM is the memory where data and instructions are stored and it is different from the conventional memory. GCM can perform real-time task partitioning by detecting data-intensive tasks in the memory. The detection of data-intensive jobs is implemented in the logic called Observer.

The internal structure of each block of the DCIM is shown in Figure 2

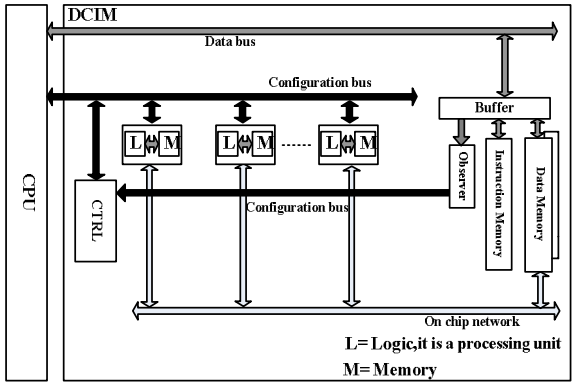
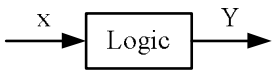


Figure 2. DCIM internal structure

The GCM consists of an Observer and a memory. The reason of putting the Observer close to memory is to get low latency. It contains a list of desired instruction, monitors the activities of memory read by the CPU and read the instruction in the buffer. If the instruction is being read by the CPU matching with one of the instructions specified in the list, the instruction address in memory will be recorded; otherwise, it will just let it pass by. The memory consists of two separate memories which are data memory and instruction memory. Those two memories can be used independently either at compile or run time. This flexibility increases the efficiency of on-chip memory utilization and supports faster program execution. The data memory consists of more than one memory bank and each memory bank can communicate with the CPU and LIMs concurrently and independently.

The LIM unit will achieve the function described below:



Y=function (x), where x represents data from internal memory of LIM.

The aim of the buffer is to hold program temporarily and let the Observer detect the desired function in program.

4. Working Procedure

A learn and act policy is introduced in the proposed system.

4.1. Learning Stage

In the learning stage, the system leaves the external CPU to do both data-intensive and computation-intensive tasks listed in the GCM. However, the Observer will be reading the activity (task operations) on the instructions from the buffer. If a desired function is detected, the associated information will be recorded and reported to the CTRL unit. Then, the CTRL allocates a corresponding LIM accordingly and then the data will be transferred to its internal memory via the on-chip network from data memory. For each task, the DCIM system will learn just once.

LIMs are reconfigurable which means their internal architectures can be changed and adapted to the required application. This makes the proposed DCIM a general purpose system.

To reduce the LIM uploading overhead, a LIM in this system may be shared by multiple tasks or inherited from previous applications. There are four situations for logic-allocation.

1. If a LIM that can take on the new task is left from previous applications and has not been assigned any task in the current application, the LIM will be re-activated and granted to this task.
2. If the LIM has already been assigned a task but is sharable with the new task, the LIM will then be shared between the previously assigned task and the new task.
3. If the LIM is on system but not available for the new task, or it is not on system at all, a new LIM will be uploaded from the external system given that there is an unassigned LIM available on system.
4. Otherwise, if there is no unassigned LIM on system, the system controller will have two options: (a) overwrite a LIM which is left from previous applications but has not been assigned any tasks in the current application, or (b) fail to fulfil the request, in which case

GCM and the system controller have to retry later.

4.2. Acting Stage

In the acting stage, the record of data-intensive will be retrieved, and these tasks are assigned to the corresponding LIMs. When the CPU read program from memory again and once meets the Observer desired information, the CPU will jump to next instruction reading and let LIMs do data-intensive job by themselves. When the tasks are done, the processed data will be transferred back to GCM via data bus.

Figure 3 illustrates an example for DCIM working stages.

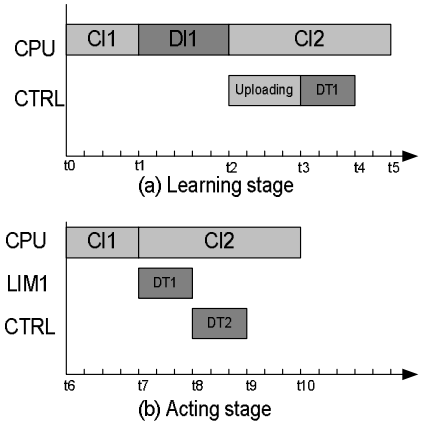


Figure 3. DCIM working stages

The CPU starts a program from t_0 by executing computation-intensive task CI1. A data-intensive task, DI1, is detected by GCM at t_1. Once DI1 is finished by the CPU at t_2, the uploading of the corresponding logic, namely LIM1, is started. The CPU in the meantime continues on the program, executing CI2. The uploading is completed at *t3* followed by data transfer, DT1, from the data memory to the memory in LIM1, which is completed at t_4. The program is finished by the CPU at t_5. In this example, only one data-intensive task is detected. The same program is reiterated by the CPU at t_6. When the CPU reaches the point in the program at which DI1 is located at t_7, it is signalled by the GCM to skip DI1 and jump to the next computation-intensive task, CI2; meanwhile, LIM1 is activated to execute DI1. DI1 is finished by LIM1 at t_8 and then data are transferred back to GCM, DT2, at t_9. CI2 is finished by the CPU at t_{10}. The speedup of the system each time the program is reiterated in this example DI1 execution time.

5. Conclusions and Future Work

5.1. Conclusions

Due to the imbalance in the processor and memory performance, a lot of efforts in research and development have been dedicated to reduce the widening gap between them. Several traditional solutions have been developed such as cache, prefetching, MPP, etc. With these solutions, the long memory latency and low bandwidth still remain a major bottleneck. To continue bridge this gap, the concept of PIM was introduced. Several architectures based on this concept have been proposed such as the IRAM, Active Pages, FlexRAM, DIVA etc. Those proposed architectures have some limitations such as their adaptability, scalability and cost-effectiveness. The main goal of the work presented in this report is to propose a new reconfigurable PIM architecture, with real time task partitioning capability, where the limitations of existing PIM architectures are overcome.

The second and following pages should begin 1.0 inch (2.54 cm) from the top edge. On all pages, the bottom margin should be 1-1/8 inches (2.86 cm) from the bottom edge of the page for 8.5 x 11-inch paper; for A4 paper, approximately 1-5/8 inches (4.13 cm) from the bottom edge of the page.

5.2. Future Work

The next steps are concerned with the investigation into the design and efficient implementation of the different blocks in the proposed architecture.

The LIM reconfiguration and CTRL parts are the main challenges in this proposed system, where the following problems will be addressed:

- Efficiency of transferring data from main memory to LIM's memory.
- The storage of the LIM's configuration information.
- The configuration of the LIMs.
- Getting and uploading the configuration information to the LIM by the CTRL unit.
- Coordination between LIMs, Memory and CPU

6. References

[1] A. Wulf and S.A. McKee, "Hitting the Memory Wall: Implications of obvious", *ACM Computer Architecture News*, New York, 1995, Vol. 23, No. 1.

[2] K. Mai, T. Paaske, N. Jayasena, R. Ho, W.J. Dally and M. Hoeowitz, "Smart Memories: A Modular Configurable Architecture", *Proceedings of the 27th International Symposium on Computer Architecture*, British Columbia, Canada, 2000, pp. 161-171.

[3] M.W. Hall, P. Kogge, J. Koller, P. Diniz, J. Chame, J. Draper, J. LaCross, J. Brockman, W. Athas, A. Srivasava, V. Freech, J. Shin, and J. Park, **"**Mapping Irregular Applications to DIVA, a PIM-based Data-Intensive Architecture", *Proceedings of the 1999 ACM/IEEE conference on Supercomputing,* USA, 1999, pp. 57 – 57.

[4] J. Suh, C. Li, S.P. Crago and R. Parker, "A PIM Based Multiprocessor System", *Proceedings of the 15th International Parallel & Distributed Processing Symposium table of contents*, San Francisco, 2001, Vol. 1 pp. 4-9.

[5] B.J. Jasionowski, M.K. Lay and M. Margala, "A Processor-In-Memory Architecture for Multimedia Compression", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* 2007, Vol. 15, pp. 478-483.

[6] K.K. Rangan, N. Pisolkar, N.B. Abu-Ghazaleh and P.A. Wilsey, "PPIM-SIM: an efficient simulator for a parallel processor in memory", *Proceedings of the 34th Annual Simulation Symposium,* USA, 2001, pp. 117-124.

[7] Zaki Ahmad, "Cooperative Intelligent Memory", *PHD thesis*, University of Hertfordshire, United Kingdom, 2007.

[8] A. Jantsch and G. Liang**,** "Adaptive Power Management for the On-Chip Communication Network", *Proceedings of the 9th EUROMICRO Conference on Digital System Design*, 2006, pp. 649 – 656.

[9] W. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks", *Proceeding of the 38th ACM Design Automation Conference*, 2001, pp. 681 – 689.