IET Computers & Digital Techniques

Review Article

IET Journals
The Institution of Engineering and Technology

Survey on memory management techniques in heterogeneous computing systems

ISSN 1751-8601
Received on 3rd April 2019
Revised 28th September 2019
Accepted on 28th October 2019
E-First on 21st January 2020
doi: 10.1049/iet-cdt.2019.0092
www.ietdl.org

Anakhi Hazarika[1] ✉, *Soumyajit Poddar*[1], *Hafizur Rahaman*[2]

[1]*Indian Institute of Information Technology Guwahati, Guwahati, India*
[2]*Indian Institute of Engineering Science and Technology, Shibpur, India*
✉ *E-mail: anakhi22@gmail.com*

Abstract: A major issue faced by data scientists today is how to scale up their processing infrastructure to meet the challenge of big data and high-performance computing (HPC) workloads. With today's HPC domain, it is required to connect multiple graphics processing units (GPUs) to accomplish large-scale parallel computing along with CPUs. Data movement between the processor and on-chip or off-chip memory creates a major bottleneck in overall system performance. The CPU/GPU processes all the data on a computer's memory and hence the speed of the data movement to/from memory and the size of the memory affect computer speed. During memory access by any processing element, the memory management unit (MMU) controls the data flow of the computer's main memory and impacts the system performance and power. Change in dynamic random access memory (DRAM) architecture, integration of memory-centric hardware accelerator in the heterogeneous system and Processing-in-Memory (PIM) are the techniques adopted from all the available shared resource management techniques to maximise the system throughput. This survey study presents an analysis of various DRAM designs and their performances. The authors also focus on the architecture, functionality, and performance of different hardware accelerators and PIM systems to reduce memory access time. Some insights and potential directions toward enhancements to existing techniques are also discussed. The requirement of fast, reconfigurable, self-adaptive memory management schemes in the high-speed processing scenario motivates us to track the trend. An effective MMU handles memory protection, cache control and bus arbitration associated with the processors.

1 Introduction

In modern computers, processor speed is affected by the varying speeds of individual computer parts. Among them, memory greatly affects computer speed as the processor must move information into memory and retrieve data from it while running applications. A perfect computer's memory is the repository that can supply any data immediately on CPU requests. However, capacity, cost and speed factor opposes the existence of ideal memory. In a high-performance computing (HPC) system, the ratio of processor speed and memory speed increasing rapidly that degrades overall computer performance. A recent survey estimated that while CPU speed is measured in gigahertz (GHz), memory speed just can progress up to megahertz (MHz) only [1]. There are several reasons to account for this growing disparity of processor speed and main memory speed [2]. One of the prime reasons is that the two diverged semiconductor industries of processor and memory focus on two different objectives. Processor technology headed towards its speed whereas memory has increased in capacity. This growing ratio of speed and capacity suggests a need for a memory management unit (MMU) that can help to improve main memory throughput. MMU ensures the correct operation of the main memory, i.e. dynamic random access memory (DRAM). It buffers and schedules DRAM requests for high-performance quality of service. It also manages power consumption and thermals in DRAM. MMU handles three types of memory interference problems in a heterogeneous computing system:

- Different threads within the same GPU application.
- Concurrently executed CPU–GPU application.
- Multiple GPU application.

To address the aforesaid issues in shared memory systems, several techniques have been proposed for main memory optimisation. They are hardware and software-based cache bypassing techniques [3–7], cache insertion and replacement policies [8], memory scheduling for CPU and GPU [4, 9–11], DRAM design [12–19], hardware accelerator [20–27], Processing-in-Memory (PIM) [28–36], etc.

Among these memory management techniques, we have selected three hardware-based techniques and focus on their architecture for this survey work. Firstly, DRAM is the prime component in any computer memory system. Amelioration techniques involved in the several DRAM designs provide us the information about their impact on overall system performance. Secondly, in the big-data processing scenario, the hardware accelerator becomes an essential component for the time and power-efficient processing than general-purpose processors. Finally, we have considered the PIM technology for memory optimisation. PIM improves the data transfer rate between the processor and memory and hence increases the speed and power efficiency of the system. In big-data processing, data movement is a high-cost operation as it requires a high latency and consumes a considerable amount of energy. These memory management techniques especially help to reduce unnecessary data movement and increase memory performance.

A heterogeneous computing system offers high-performance efficiency by changing hardware architectures of processors [37, 38]. Integration of CPU with GPU in proximity opened up a new computing era and gave challenges to the designer of the memory controller for memory access, queuing and execution of instructions in the multi-core processor (MCP) systems [39]. GPU applications are more bandwidth-demanding than CPU applications, as the GPU can execute multiple parallel threads simultaneously [40]. In deep learning applications, the GPU provided state-of-the-art inference performance and recognised widely as the best suitable computing resource for both speed and energy efficiency [41]. However, increasing memory bandwidth demand by GPU along with latency-sensitive memory requests from CPU cores in the same system leads to poor performance.

This paper presents a qualitative survey on various DRAM designs and hardware accelerator architectures used for memory

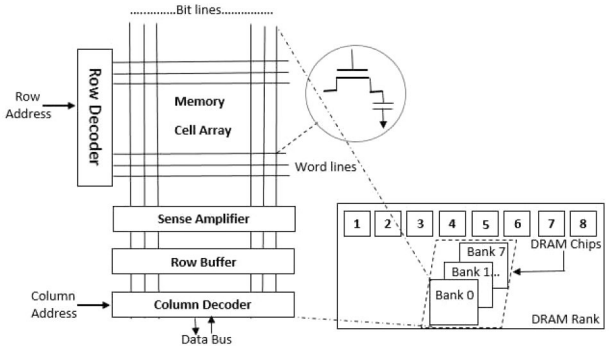

Fig. 1 *Hierarchical DRAM organisation*

management in heterogeneous systems. Moderation in DRAM architecture reduces memory latency that further improves the performance of GPU based computing. Hardware accelerator coupled with a general-purpose processor (CPU and GPU) boosts up the overall system performance. These accelerators deal with the memory-bound tasks in memory-intensive applications such as graphics processing, machine learning, etc. Self-learning algorithms in a hardware accelerator aid the memory optimisation processes. In addition, this study also presents a performance analysis of PIM architectures used to reduce memory access latency and energy consumption in HPC devices.

The major contribution of this study is to provide an in-depth analysis of three different types of techniques to minimise memory interference problems and methods to upgrade significantly the performance and efficiency of heterogeneous systems. This survey covered two important aspects related to memory management techniques. First, an architectural study of DRAM, memory-centric hardware accelerator, and PIM on various memory intensive and embedded applications. Second, problems and potential directions towards the efficiency enhancement of the existing techniques are also discussed. We believe that this survey will provide insights to the researchers and designers about the state-of-the-art approaches of the aforesaid areas and will motivate them to address the memory management problems with novel solutions.

The organisation of this paper is as follows: Section 2 presents the background knowledge related to this survey. Section 3 discusses the various design architecture of DRAM, hardware accelerators and PIM. This section also presents a survey of their relative performances. Section 4 provides a discussion and future research direction followed by a summary of this report in Section 5.

2 Background

This section presents the background on DRAM, GPU, hardware accelerator and PIM.

2.1 DRAM and memory controller

In the modern computer system, DRAM has an array of bit cells (transistor–capacitor pair), which are addressed through the row or word lines and data can be accessed from memory via column or bit lines. The number of bit cell arrays varies from architecture to architecture for the design and performance trade-off of different DRAM devices [42]. The dynamic nature of this memory is governed by the periodic refresh of the capacitor to maintain the information in it. Modern DRAMs are organised into ranks (up to 4) that contain multiple banks. A bank is an array of rows and columns with a row-buffer and each bank can be accessed parallelly by buses [43]. Memory performance can be characterised by two most critical parameters latency and bandwidth along with several other parameters (energy efficiency, size etc.) and hence memory architecture cannot be absolutely superior to another based on single parameter or application. Earlier, DRAMs were fabricated by integrated circuit technology and now they were found in multi-chip plug-in modules (DIMMs, SIMMs etc.) installed directly on the motherboard [44]. Fig. 1 shows the hierarchical architecture of the DRAM organisation.

The MMU is the physical hardware that allocates memory resources among the processors and maximises memory performance. As a part of MMU, the DRAM controller plays a key role in improving CPU and GPU performance. The backend of the memory controller issues commands to the DRAM module and the front end connects to the processor [45]. The behaviour of the contemporary DRAM controller depends on the memory access pattern, different DRAM architecture and its timing parameters [46]. Software frameworks like CUDA and OpenCL manage memory on the CPU/GPU system and provide memory management methods to enhance the system performance [47].

2.2 Graphics processing unit (GPU)

GPU is a highly parallel, multi-threaded programmable logic chip that served as both a scalable general-purpose parallel processor and a graphics processor. The basic GPU architecture composed of streaming processors, DRAM channels having high bandwidth and on-chip L2 cache. GPU provides high computational throughput and energy efficiency by executing thousands of instructions parallelly on the available cores to hide latency [47, 48]. MCP enhances performance throughput at lower power than a single-core processor [49]. The current computational platform integrates CPUs and GPUs for general-purpose programmings, where both shared memory and common address space. Because of the simpler programming environment and no data transfer between CPU and GPU, the heterogeneous system possesses several advantages in cost, efficiency and performance improvement. General-purpose computing by GPU in heterogeneous systems highly benefits the big-data processing by accelerating the computation. Heterogeneous computing is accelerated by 10 to 100 times than CPU-only computing since 2007 [50]. NVIDIA GRID [51] and AMD Navi [52] are two commercially available GPUs used for multiple applications.

2.3 Hardware accelerator

Hardware accelerators are specially designed processors to speed up the computing performance efficiently, for example GPU, ASIC, FPGA, AI accelerator, cryptographic accelerator, etc. Among these, GPUs and FPGAs are two main computational platforms that can perform better performance than CPU alone on different workloads. Han *et al.* [53] provide a performance analysis of different GPU-based accelerators on deep learning applications and also recommend resources used for efficient heterogeneous computing. Accelerators have different hardware architecture depending on various applications and workloads. Reconfigurable, memory-centric accelerators operate at the level of memory and optimise the memory resources to reduce the processing time of computing elements. Generally, accelerators are connected to the host processor via high speed interconnects, but efficiency reduces when there is a transfer of massive data regardless of the speed and/or efficiency of the accelerator [20]. Having adequate external memory bandwidth is a challenging problem while designing an efficient accelerator architecture [21]. FPGA or ASIC based accelerator architecture works faster than accelerators with standard shared memories. Intel Xeon Phi and NVIDIA GPU achieve high computational performance in multi-core processing systems [54].

2.4 Processing-in-memory

PIM is an embedded logic-in-memory mechanism where a processor is integrated with DRAM on the same die. The PIM technology avoids excess latency caused by off-chip data transfer between processor and memory. It also aims to reduce the energy consumed by the system. In a PIM chip, processor's logic devices are coupled with die-stacked memory. In big-data processing workloads, PIM provides high memory bandwidth requirement and memory-capacity-proportional bandwidth. In modern computer systems, a single hybrid memory cube (HMC) of specifications 1.0 and 2.0 offers up to 320 and 480 GB/s of external memory bandwidth, respectively [55, 56], whereas DRAM offers 512 GB/s per cube internal memory bandwidth. The disparity between the

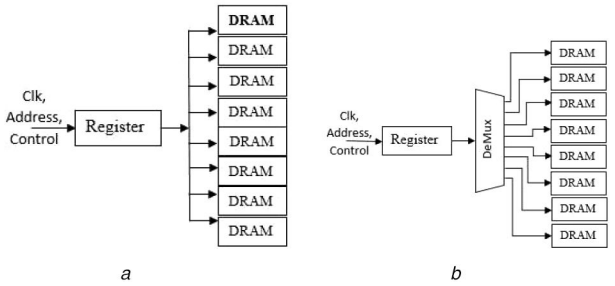

Fig. 2 *Different architectures of DIMM*
(a) Conventional DIMM architecture, ***(b)*** MCDIMM architecture

internal and external memory bandwidths increases with the use of more HMC's. If we consider a system having 16 HMCs, conventional processors are capable of a maximum of 480 GB/s of memory bandwidth. On the other hand, PIM can provide 8 TB/s (16 × 512 GB/s) internal bandwidth to the in-memory computation unit [57]. Thus, PIM technology overcomes the limitation of performance throughput caused by memory latency.

3 Approaches to efficient memory management

In this section, we discuss several architectural techniques of DRAM, hardware accelerator, and PIM used in heterogeneous systems. In addition, we analyse the impact of their existing architectures on system performance. Further, we provide a detailed summary of their performance on different workloads.

3.1 DRAM designs

In a heterogeneous system, memory interference impacts the overall performance of the system. Modification in DRAM design reduces memory latency and improves memory bandwidth than conventional DRAM. This technique alleviates the shared memory interference on GPU based systems and enhances the overall system performance. This subsection provides detail of the internal operation, organisation, and control of DRAM on system performance.

Ahn *et al.* [12] proposed a power-efficient memory architecture having multi-core dual in-line memory modules (MCDIMMs). MCDIMM is designed with a small modification of existing DIMM. DRAM chips present in this module take less activation and precharge power than conventional DIMM. In this architecture, a Demux is placed after the conventional registers and en-route the received command signal to the appropriate DRAM chip. The Demux translates the command signal received from the memory controller into multiple signals via the narrow data bus as shown in Fig. 2. Instead of loading thousands of bits into the sense amplifier at per ACTIVATE command, the Demux register loads fewer bits into the sense amplifier on row activation. The latency issue of this design is mitigated by designing a *virtual memory* device with several DRAM chips. They have compared the result of the design with the NAS parallel benchmark [58], SPEC CINT 2006 [59] and SPECjbb2000 [60] application benchmarks. The proposed design achieved improvement in memory power, instructions per cycle and system-energy delay product by 22, 7.6, and 18%, respectively, for a set of multithreaded applications and different workloads. These application benchmarks achieve performance improvement at the cost of longer serialisation latencies [61].

Conventional DRAM devices have a disadvantage in servicing memory requests as the entire DRAM rank becomes inactive while being refreshed. In double data rate (DDR) DRAM, refresh in cells is performed at a rank level and hence performance degrades with the further increase of refresh latency. This problem will be more ubiquitous with future increases in DRAM density (8–32 GB by 2020). Although low power DDR DRAM uses bank-level refreshing, yet it has few shortcomings. To overcome these issues, Chang *et al.* [62] present two approaches namely, dynamic access refresh parallelisation (DARP) and subarray access refresh parallelisation (SARP) for parallelising refresh and access within DRAM.

In DARP, the memory controller first specifies the idle bank (no pending memory request) and is then refreshed. Secondly, it performs refresh operations when banks are serving write requests. SARP techniques take the advantages of having subarrays in each DRAM bank and sense amplifiers integrated with each subarray. Here, idle subarrays serve the memory request while others in the bank are being refreshed. SARP demands DRAM hardware modification. Refresh parallelisation negatively affects the energy, performance and density scaling of DRAM devices.

Researchers had used both memory intensive and memory non-intensive benchmarks such as SPEC CPU2006 [63], STREAM [64], TPC [65] to evaluate the system performance of DARP, SARP, and DSARP across 100 workloads by using various DRAM densities. DARP provides improved system performance overrank level refreshing by 2.8, 4.9, 3.8% and bank-level refreshing by 7.4, 9.8, 8.3% on average for 8, 16, and 32 GB DRAMs, respectively. SARP's average system performance improvement for bank-level and rank level refreshing is 3.3, 6.7, 13.7 and 7.9, 11.7, 18.6%, respectively, for 8, 16, and 32 GB DRAM. A combination of SARP and DARP mechanism (DSRAP) provides 3.1% on average and up to 7.1, 14.5, and 27.0% improvement for 8, 16, and 32 GB DRAM devices.

Inefficiency in bulk data movement between DRAM and processors reduces the system performance in terms of high latency and power consumption. Chang *et al.* [16] present a rapid and energy-efficient DRAM architecture called Low-Cost Inter Linked Subarray (LISA) to transfer bulk data between DRAM and processor. LISA uses isolation transistors to connect the bit lines of neighbouring cell subarrays resulting in low latency, energy, and cost. Isolation transistors are referred to as a link and the link connects the bit lines for the same column of two adjacent subarrays. LISA can be used to implement three mechanisms for bulk data copy and latency reduction. RISC (Rapid Inter Subarray Copy) achieves low latency and DRAM energy by using high bandwidth connectivity between subarrays within a bank. LISA architecture in VILLA (Variable Latency) DRAM reduces access latency of frequently accessed (hot) data by caching it in fast subarrays within the bank. Another application LIP (Linked Precharge), is also used to reduce the precharge latency for a subarray by linking its precharge units with the neighbouring idle precharge unit. LISA introduces a new DRAM command RBM for a row buffer movement that transfers data between row buffers of a bank through the links. In this scheme, data coherence may appear in DRAM as the processor may have old cache lines that belong to the section of memory being copied. This architecture does not provide significantly faster performance and the cost (in terms of latency) of copy operations may reduce overall system benefits.

The authors have performed a quantitative evaluation for the three applications: LISA-RISC, LISA-VILLA, and LISA-LIP. They use benchmarks from TPC (–C/–H), DynoGraph [66], SPEC CPU2006 and STREAM. These three applications accelerate the system performance and improve the energy efficiency of memory when used individually or together, across a variety of workloads and system configurations. LISA-RISC finishes the copy of a single row of data with 9.2× lower latency than RC-InterSA. For a range of 4-core workloads, LISA-VILLA and LISA-LIP achieve, on average 5.1 and 10.3% system performance improvement, respectively.

To effectively address the memory latency issue of DRAM, scaling down the DRAM cell size is important to meet the growing demand for increasing memory capacity. CHARM (Center-High-Aspect-Ratio-Mat) [17] is a novel DRAM bank organisation that introduces a few fast banks to reduce DRAM access latency (shown in Fig. 3). CHARM improves the access time for local banks with minimal area across entire memory by increasing the aspect ratio of *mats* (subarrays). As the memory accesses for different operations are randomly spread over the memory, they have reduced the area overhead by placing the *mats* having different aspect ratio*s* together. Thus, mats with high aspect ratio are placed near the input-output interface to reduce the access time. DRAM processes (i.e. sense, restore, and precharge) are aware of bit-line capacitance and hence decrease in the number of word-lines per mat is required to reduce the capacitance. In CHARM

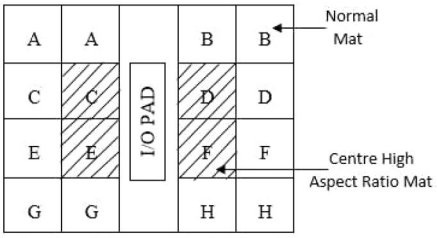

Fig. 3 *CHARM architecture*

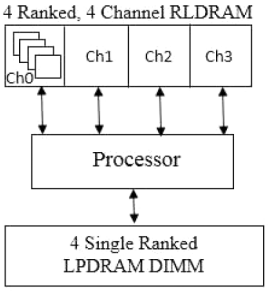

Fig. 4 *Heterogeneous DRAM architecture*

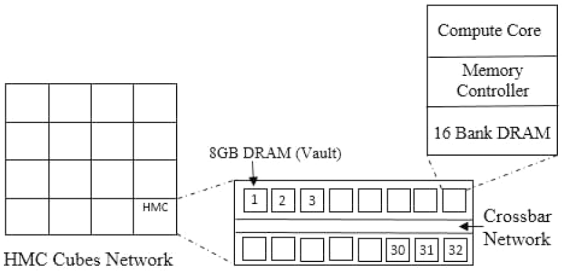

Fig. 5 *Basic Tesseract architecture*

microarchitecture, *mats* with high aspect ratios are placed at the centre blocks. The column decoders and inter-bank data-lines are reorganised to reduce the access time to centre high aspect ratio mats by half than the access time to the other mats. The timing parameters of DRAM are kept unchanged. CHARM microarchitecture uses extra sense amplifier for latency reduction and hence achieves low latency access at the cost of additional chip area.

Performance and energy efficiency parameters 'Instruction per cycle' (IPC) and 'Energy-delay product' (EDP), respectively, are evaluated by using benchmark SPEC CPU2006, SPLASH-2 [67] and PARSEC [68] multi-programmed and multi-threaded workloads. On CMP, CHARM raises IPC and EDP values up to 21 and 32%, with a 3% increase in area overhead.

Chatterjee *et al.* [18] proposed a low complexity DRAM architecture to provide heterogeneity where access latencies and energy per memory access is different in a different area of memory. Here, the cache line with high priority is put in a low-latency area of the main memory, and then the cache line with less priority is placed in a low energy area. The memory industry has produced three types of DIMMs such as DDR3, low power DRAM (LPDRAM) [69], and reduced latency DRAM (RLDRAM) [70] for latency, power and bandwidth optimisation. As shown in Fig. 4, this architecture employs low-latency and high-latency DIMMs separately to reduce the memory controller complexity and design cost. The heterogeneous DRAM module combined with low latency, low power and high bandwidth RLDRAM and LPDRAM devices. Further, LPDRAM chips are integrated with DLL (Delay Locked Loop) to synchronise the data signal with the global clock used by the memory controller and ODT (On Die Termination) resistors to increase signal integrity at high frequency. This architecture is especially designed for latency reduction. The capacity of total memory is equal to the size of RLDRAM and LPDRAM only and management of on-chip caches limit the overall performance.

They have run multithreaded workloads (each core running 1 thread) on OpenMP NAS Parallel Benchmark suite and STREAM benchmark and multi-programmed workloads on SPEC CPU 2006 suite. The evaluation of this proposed scheme was carried out for three memory configurations. In RD and RL configuration, 1 GB RLDRAM3 is integrated with 7 GB DDR3 and 7 GB LPDDR2, respectively, with an additional 1 GB ECC. Similarly, in DL configuration, 1 GB DDR3 is coupled with 7 GB LPDDR2 plus 1 GB ECC.

RD provides 21% better performance in average throughput over DDR3, whereas RL offers a 13% average improvement over the baseline system. In comparing to DDR3, the overall system energy consumption drops by about 6 and 13% with RL and DL schemes, respectively. Hence, the RL scheme is most suitable for a wide range of workloads.

In the conventional DRAM device, precharging the BLs consumes maximum time in the deactivation of row buffer because of its capacitance, which further degrades the performance. Seongil *et al.* [19] propose a DRAM microarchitecture, which decouples the BLs from the row buffer through an isolation transistor. When the isolation transistor is in one state, DRAM functioning is operated conventionally. On the contrary, data in the BL sense amplifiers controlled by the (de)multiplexers and precharging of BLs occur simultaneously. This modified architecture minimises the row deactivation time. Since the early precharge policy consumes more energy, they have also designed a scheduling policy for DRAM micro-operations enabled by row buffer decoupling. In memory-intensive applications, data movement consumes a maximum amount of energy. In this regard, this proposal is inefficient in handling such an energy bottleneck as it solely focuses on the reduction of activation energy. Row-buffer decoupling ameliorates the IPC and MIPS^2/W (square of million instructions per second per watt) by 14 and 29%, respectively, for nine memory-intensive SPEC CPU2006 applications.

Limited memory bandwidth of conventional DRAM devices is the main bottleneck in effective big-data processing. It demands a need for scalable hardware design in the main memory to process and analyses data-intensive workloads. To mitigate this, a programmable accelerator (called *Tesseract*) having 3D stacked memory has been designed with PIM technology [15]. Fig. 5 depicts the basic architecture of *Tesseract*. HMC memory organisation is adopted with eight 8 Gb DRAM-base layers. In an HMC, 32 vaults are connected with 8 high speed (40 GB/s) serial links through crossbar networks. Each vault is composed of a 16-bank DRAM and a memory controller. In this system, the *Tesseract* architecture accelerates the non-cacheable memory mapping of the host processor's dedicated memory. A Tesseract core can access only its own local DRAM region rather than the entire memory space. Therefore, a message-passing communication mechanism is developed between different Tesseract cores. This mechanism can hide long remote access latencies, avoid cache coherence between tesseract core and L1 cache and updates memory of shared data without software synchronisation primitives. Moreover, they have introduced two prefetching hardware *list prefetcher* and *message triggered prefetcher* so that cores can utilise the available memory bandwidth effectively. This technique requires a programmer effort to identify and specify which operation should be run close to memory and how to map the data into different 3D stacked memory.

The performance of the Tesseract system is evaluated against conventional DDR3 and HMC based memory systems. Tesseract is evaluated in all three forms: Tesseract, Tesseract-LP (with list prefetching) and Tesseract-LP + MTP (with list prefetching and message triggered prefetching). Tesseract average performance without prefetching is improved by 9×, whereas with prefetching is by 14× compared to conventional systems. Moreover, it shows an 87% average reduction in energy consumption tested in five different graph processing workloads.

DRAM refresh impacts the overall memory performance by stalling the program execution. To address this issue, Bhati *et al.* [71] proposed a modified DRAM architecture that introduces a dummy refresh command to skip refresh operations and to increment the internal refresh counter. It results in a significant

improvement in performance and energy efficiency. Researchers aim to develop the interface of the refresh counter so that the memory controller can make the refresh counter both readable and writable. The prior refresh reduction techniques failed to meet the required performance levels in the high-density DRAM scenario. The proposed *Flexible Auto-Refresh* (REFLEX) technique introduces an auto-refresh (AR) command to serve almost all refresh operations. In the REFLEX-1x scheme, the memory controller schedules refresh commands only if there are weak rows among the rows refreshed in an AR; otherwise a ‘ dummy-refresh’ is issued to increment the refresh counter. Thus, REFLEX-1x reduces the overall refresh activity and hence total energy consumption. However, this technique does not reduce the power of other DRAM operations and does not consider the data retention time of DRAM cells. Evaluation of this scheme was conducted by using multi-programmed and multi-threaded workloads from the SPEC CPU2006 suite and the NAS parallel benchmark suite. The REFLEX technique can skip 3/4th of the total refreshes and can save 25% energy than a row-level refresh.

With the increasing density of DRAM, refresh operation increases the performance overhead of DRAM memory. As the number of cells increases in the DRAM memory, the requirement of longer or frequent refresh operations induced maximum stalls during execution. To mitigate this problem, Kate *et al.* [72] proposed a non-blocking DRAM refresh scheme where read access refreshes the data present in the memory block. Non-blocking refresh can refresh a portion of the data present in the memory block and improves DRAM refresh overhead of all the DRAM-based memories used in servers, personal computers, GPUs, embedded platforms, etc. This technique uses redundant data in each memory block to compute the refreshing. Instead of refreshing all the data of memory blocks simultaneously, the proposed technique refreshes a portion of the memory block on a single refresh operation. When a memory controller sent a read access request to a particular memory block, this access request refreshes the memory block with the help of redundant data. For performance evaluation, simulated various NASA Parallel, Parsec, and SPEC2006 benchmarks are simulated on the Gem5 simulator. Non-blocking DRAM Refresh achieves 13 and 17% performance benefits for 16 GB Intel/AMD and IBM server systems, respectively. For Intel/AMD and IBM server having 32 GB DRAM chip, DRAM refresh provides improvement up to 21 and 35%, respectively. IBM server achieves more benefits as they have a large amount of redundant data.

In the recent scenario of growing processor speed and memory size, the relatively slow main memory (DRAM) creates a bottleneck in system performance. Development of low powered, high bandwidth 3D stacked DRAM memory favours the situation. 3D DRAM has limitations on the distribution of clock to all memory die and utilisation of a large number of through-silicon vias (TSVs), that result overhead on power, area, and cost. To address the aforesaid issues, Douglass and Khatri [73] proposed a memory architecture using a ring based scheme (MARS) that faster the data transfer within the 3D memory module. This architecture uses a 3D standing wave oscillator (SWO) for high-speed communication and provides a fast, low skewed clock to all DRAM layers. In addition, MARS architecture introduces a ring-based interconnect alongside a 3D SWO which exchanged data between the DRAM channel and ring interconnect. This paper presents three MARS configurations MARS8, MARS16, and MARS32 based on the number of channels used to match the performance with high bandwidth memory (HBM). This technique can reduce the read latencies and power consumption and improves the throughput of the system. MARS provides 2.7× and 4.2× faster speeds than Wide I/O and traditional HBM, respectively. Further, MARS uses 4× fewer TSVs per channel than HBM or Wide I/O and hence achieves 6.9× lower power consumption than HBM.

The heterogeneous system architecture has its own cache hierarchy. However, both the CPU and GPU share a global cache or main memory. Different attributes of computing capabilities and different bandwidth/latency demands of CPUs and GPUs introduce a coherence problem. To effectively address this issue without the overhead of prior works [74–76], Koukos *et al.* [77] provide a unified virtual memory (UVM) space. UVM allows any processor in the system to access virtual address space. The use of UVM simplifies the architecture of MMU by replacing the private TLBs with virtually-indexed, virtually-tagged (VIVT) GPU L1 caches that eliminate the need for translations. The advantages of adopting UVM provide overall speedup, energy savings, and lower EDP. Further, this architecture reduces 50% of the TLB area and saves 70% MMU energy.

In the growing scenario of data-intensive applications, GPGPU demands high memory capacity that creates a major bottleneck to the programmer. To address this challenge, UVM supports the GPU and CPU by using a shared pointer to a unified virtual address space. Reducing the overhead of address translation by UVM is first explored by Zheng *et al.* [78]. They have proposed a user-directed hardware prefetcher that is further developed and implemented in the latest NVIDIA GPUs [79]. Recently, Ganguly *et al.* [80] proposed pre-eviction policies compatible with hardware prefetching in UVM. This scheme achieves significantly better performance speed up than LRU (Least Recently Used) based on page replacement policies. Also, effective and efficient memory sharing by utilising the UVM address space is well studied in [81, 82]. These papers include the methods of allocating both physical and virtual memory addresses in the heterogeneous systems.

Table 1 shows the comparison of the overall system performance of different DRAM design architecture in terms of different system parameters. New DRAM designs are able to reduce latency present in conventional DRAM and hence can improve DRAM bandwidth and performance of GPU based systems.

3.2 Hardware accelerator in heterogeneous system

In data-intensive applications, the requirement of sufficient on-chip memory, memory bandwidth and parallelism in data transfer create bottlenecks in memory design. In addition, the energy associated with data transfer for varying on-chip memory size also affects system performance. GPU and FPGA based hardware accelerator assist general-purpose processors in performing complex and intense computations efficiently and improving system performance. In this section, it has been discussed that how GPU and FPGA together help in improving memory performance as an accelerator.

Developers of heterogeneous multi-core systems face the following typical challenges:

- How to utilise accelerator's processors in general computations.
- How accelerator and task map.
- How to partition the applications to execute in phases on accelerators.

Depending on the unique demand of computing resources, the architecture and working model of accelerators vary significantly. Che *et al.* [22] performed a comparative study on development costs and performance of FPGA as well as GPU on the basis of three applications: *Gaussian Elimination*, *Data Encryption Standard* (DES) and *Needleman–Wunsch*. Based on this study, they have proposed an application characteristic for accelerator-application mapping to help the designers in the selection of appropriate architecture for certain applications. They have used the same algorithms for both FPGAs and GPUs in their implementations. Experiments are performed on NVIDIA GeForce 8800GTX GPU, Xilinx Virtex-II Pro FPGA, and Intel Xenon Multicore CPU.

In Gaussian Elimination, both FPGAs and GPUs provide better performance than CPUs for a 64×64 input size with parallel computation. FPGA and GPU take 262 K, 746 K cycles, respectively, whereas the single-threaded CPU version requires 3.15 M cycles. Four-threaded OpenMP version requires 9.45 M cycles.

To process a single, 64-bit block on DES encryption, FPGA requires 83 cycles, while GPU requires 580 K cycles to execute the same operation. The reason behind the huge performance gap is that FPGA takes a single cycle for bit-wise calculations, and the

on-chip memory is enough to store all the intermediate data but the GPU involves 300–400 cycles for memory access. The performance gap between GPU and CPU will reduce if we implement the application for a larger data set, but the complexity of FPGA will increase. Finally, this paper concludes that the FPGAs are suitable for bit operations in the combinatorial logic system and the GPUs shine on parallel workloads with deterministic memory access.

In the *Needleman–Wunsch* application, data are processed in order instead of parallel processing. While processing a smaller input (64×64), GPU consumes 15 times more execution cycles than FPGA, but the ratio of GPU and FPGA execution cycles decreases for larger input sizes. From the development point of view, GPU has better programmability and time efficiency than FPGA development.

In an investigation performed by Kanev *et al.* [83] in Google data centre, observed that ∼5% of CPU cycles are consumed during memory copy and memory move functions. Duarte and Wong [25] presented a cache-based hardware accelerator architecture to lower the data movement cost in a multi-core system. Previously designed direct-mapped cache architecture consists of a cache directory coupled with cache memory. The cache directory is the array of main memory addresses whose data are stored in the cache memory to speed up memory access. In a multi-core system, the proposed accelerator improves the cache memory performance with the support of the message passing protocol. This work removed the need for any data movement at all by simply keeping an indirection table in the cache. A copy operation is performed by placing an entry in the table. A specific logic unit near the cache maintains the data of the memory addresses if they are being modified. The major limitation of this work is that the size of the memory region to be copied must be the same as the size of the cache. Also, cache access overhead is observed as the entry of the indirection table has to be checked at every cache access. System performance analysis of the cache-based accelerator is carried out in Simics full-system simulator. STREAM benchmark is used for a comparative study of system throughput of the proposed accelerator with five copy kernels. The proposed accelerator reaches speed-ups, decreases the number of instructions and achieves a higher cache hit rate. Further, a theoretical analysis is also reported to study the system.

In heterogeneous system architecture, mapReduce operation [84] simplified the parallel programming execution of data-intensive applications. Chen *et al.* [23] focus on the architecture

Table 1 Performance comparison of different DRAM design techniques

Reference	Configuration	System performance improvement	Benchmarks	Principal operation
Ahn *et al.* [12]	MCDIMM	Memory power: 22%	NAS parallel benchmark, SPEC CINT, SPECjbb2000	Row activation time reduction
—	—	Instruction per cycle: 7.6%	—	—
—	—	System-Energy delay product: 18%	—	—
Chang *et al.* [62]	DARP	Mean: 7.4, 7.9, 7.8% (8 GB/16 GB/32 GB)	SPEC CPU2006, STREAM, TPC	Parallelising refresh operation
—	SARP	Mean: 9.8, 11.7, 12.3% (8 GB/16 GB/32 GB)	—	—
—	DSARP	Mean: 8.3, 18.6, 20.2% (8 GB/16 GB/32 GB)	—	—
Chang *et al.* [16]	LISA-RISC	Speed up: 66.2	TPC (–C/–H), DynoGraph, SPEC CPU2006 and STREAM	Bulk data copy between memory and processor
—	LISA-VILLA	Speed-up: 5.1%	—	—
—	LISA-LIP	Speed-up: 12.2%	—	—
Son *et al.* [17]	Asymmetric DRAM	Instruction per cycle: 21%	SPEC CPU2006, SPLASH −2 and PARSEC	Scaling down the DRAM cell
—	—	System-energy delay product: 32%	—	—
—	—	Area overhead: 3%	—	—
Chatterjee *et al.* [18]	RD	Average throughput: 21%	DDR3 baseline	Heterogeneity in memory module
—	RL	Average throughput: 12.9%,	—	—
—	DL	Energy consumption drop: 6%	—	—
—	—	Energy consumption drop: 13%	—	—
Seongil *et al.* [19]	DRAM	Instruction per cycle: 14%	SPEC CPU2006	Minimising row deactivation time
—	—	$MIPS^2/W$: 29%	—	—
Ahn *et al.* [15]	Tesseract	Without prefetching: 9×	DDR3-OoO, HMC-OoO, and HMC-MC	Accelerating memory mapping
—	—	With prefetching: 14×	—	—
Bhati *et al.* [71]	REFLEX	Energy saving: 25%	SPEC CPU2006, NAS parallel benchmark	Skipping refresh operation
Kate *et al.* [72]	Non-blocking refresh	16 GB: 13 and 17%	Intel/AMD system, IBM server	Refresh operation by read access
—	—	32 GB: 21 and 35%	—	—
Douglass and Khatri [73]	MARS	Latency: 4×	HBM	Fast data transfer
—	—	Power consumption: 6.9×	—	—
—	—	Performance/Watt: 8×	—	—
Koukos *et al.* [77]	UVM	Speedup: 45%	VI Hammer	Virtual coherence
—	—	Power savings: 20%	—	—
—	—	Lower EDP: 45%	—	—
—	—	TLB area reduction: 50%	—	—

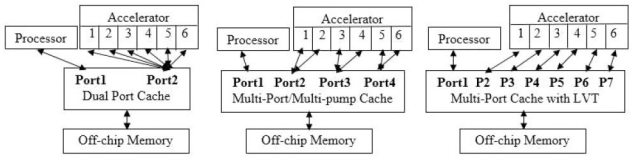

Fig. 6 *Different architecture of the multi-ported cache*

and execution of MapReduce applications on an integrated CPU-GPU chip. MapReduce tasks are scheduled in two schemes: *map-dividing scheme* and *pipelining scheme*.

The map-dividing scheme separates the mapping task between both CPU and GPU and the Pipelining scheme executes the maps in the pipeline. In this design, one core of the CPU act as a scheduler and the rest of the cores and streaming processor of the GPU works as scheduling units. They have used a zero-copy buffer for the input data and scheduling information to avoid runtime data transfer between the CPU and the GPU. Also, they have adopted a runtime tuning method in the map-dividing scheme to balance the task block size for each scheduling unit. In the pipelining scheme, both dynamic load balancing and static load balancing based implementations are adopted. This architecture achieves approximately 7.5×, 4×, 3×, 28.5×, and 9× performance improvement over sequential implementations in five different MapReduce applications such as Clustering (KM), Word Count (WC), Naive Boys Classifier (NBC), Matrix Multiplication (MM) and *k*-nearest neighbour (kNN) search, respectively. Furthermore, the runtime tuning method provides satisfactory load balancing in low scheduling. Despite these advantages, this scheme can work on homogeneous workloads more efficiently. Recent advances in coupled CPI-GPU architectures offers high throughput.

DRAM bandwidth is a typical bottleneck in high performance and energy-efficient big-data processing systems. Akin *et al.* [24] proposed a highly parameterised and scalable accelerator architecture by optimising DRAM memory access pattern with the help of 1D, 2D, and 3D fast Fourier transform (FFT) algorithms. This work has maximised the bandwidth utilisation and introduces 3D memory for tiled data layout. For hardware implementation of DRAM optimised FFT algorithms, an automatic design generation technique has been adopted. For a conventional FFT algorithm in DRAM, strided memory access (multi-directional memory access) is required. To avoid this inefficient strided access pattern, they have used a tiled and cubic memory mapping scheme for FFT. The proposed architecture is configured with two DRAM controllers, local memories and FFT core. The controllers perform read/write operations parallelly throughout the computation. Local memories serve as a local fast buffer, as well as an interface between the FFT core and the DRAM controller. In this work, the optimised FFT algorithm is implemented on off-chip DRAM, 3D-stacked DRAM, ASIC, and FPGA and evaluates the energy and system performance of the designed DRAM architecture. In comparison to the conventional row-column FFT algorithm, this design offers 6× improvement in overall system performance and 6.5× improvement in power efficiency.

Choi *et al.* [26] describe the impact of multi-ported cache architecture on host processors and multiple accelerators placed on the same chip, as shown in Fig. 6. The authors have put the emphasis on the dependency of all the accelerators on the cache ports number for the performance benefit in parallel memory access. In this cache micro-architecture, data stored in DRAM and cache is shared by both processor and accelerator and hence no requirement of cache coherence makes this model simpler than other memory architecture. In this work, the L1 cache is designed and implemented on FPGA on-chip memory. Data shared by both processor and parallel accelerator can be accessed via this cache. Accelerator's non-shared, local data are stored in FPGA RAMs. The proposed multi-ported cache memory design has two approaches: multi-pumping approach and live-vale table approach. Cache attributes (cache size, line size, and associativity) are modified by using Verilog parameters and do not require memory partitioning. Dual ported cache architecture limits the performance of many accelerators works in parallel, and hence two designs of multi-ported caches are configured based on *multi-pumping* and *live-value tables* to improve the memory bandwidth and to allow several concurrent accesses. When multiple hardware accelerators operate in parallel, the number of cache ports and its interface impacts the system performance. However, this approach has a performance bottleneck while accessing the same off-chip memory by both the processor and accelerator through a single multi-port cache. Coordinated operations of processors and accelerators are not discussed in this paper. Analysis of performance reveals that 4-port multi-pump direct-mapped cache with size 16 kB offers the leading performance regarding total execution time, area and energy consumption. On an average cache design with six accelerators shows speed up from 0.73× to 6.14×.

Nowadays, various accelerators are proposed on the FPGA platform for deep CNN applications due to its reconfigurability, high performance, etc. The mismatch between the computation throughput and memory bandwidth of FPGA creates a bottleneck in performance improvement. Zhang *et al.* [85] proposed a novel architecture for the CNN accelerator on the FPGA board to overcome this problem by using the roofline model [86]. The proposed CNN accelerator design is composed of several PEs, on-chip buffer, external memory and interconnect. The organisation of all components should be carefully done so that on-chip data can process efficiently. To achieve an efficient design for each layer of CNN, researchers have optimised the computation and memory access. The entire system is implemented on a single FPGA chip with MicroBlaze soft processor core and uses DDR3 DRAM. Vivado HLS tool is used to synthesise the accelerator design. The performance of the proposed accelerator is compared with software-based counterpart and it yields around 18× and 5× speedup for single thread and 16 thread applications, respectively. The overall performance of the accelerator achieves 61.62 GFLOPS. FPGA implementation of the accelerator consumes much lesser energy than its software counterpart.

Eyeriss by Chen *et al.* [27] is an ASIC CNN accelerator that couples an array of processing units with register file and control unit, enabling flexibility in scheduling CNN computation. This flexibility limits arithmetic unit under-utilisation. However, under-utilisation still persists if the kernel size and output feature map height of CNN layers are not adaptable with the dimensions of the computing array. This accelerator offers minimum energy consumption by minimising data movement with several different data reuse strategies. State-of-the-art accuracy is achieved due to the efficient dataflow and supporting hardware including a spatial array of processing elements, memory hierarchy, and on-chip network. In addition, Eyeriss exploits data statistics to minimise energy through zeros skipping/gating to avoid unnecessary reads and computations; and data compression to reduce off-chip memory bandwidth, which is the most expensive data movement. Eyeriss architecture is an ASIC implementation with 12 × 14 PEs and 108 kB scratchpad memory which achieves maximum computational efficiency of 92, 80, and 93% on the AlexNet layer 1, 2, and 3–5, respectively. Eyeriss uses a run-length encoding scheme when transferring activations to and from DRAM. This saves energy (and time) by reducing the number of DRAM accesses. However, after the data is loaded into the on-chip buffer, the data is stored in the expanded form. Thus, there are no savings on data transfer cycles from one internal buffer to another internal buffer or to the multipliers.

In a heterogeneous system, CPU works with several GPUs and other customised accelerators. These application-specific accelerators require a virtual address space between the host processor and accelerators and hardware support for address translation. Address translation overheads of accelerators deteriorate their performance. In [87], Cong *et al.* proposed a translation lookaside buffer (TLB) design for a heterogeneous system with an accelerator to improve the address translation. A two-level TLB hierarchy is designed with small TLBs (16–32 entries) per accelerator and a shared large TLB (512 entries) for all to minimise the number of page table walks (shown in Fig. 7). In this design, MMU uses the same page table walker for both the host CPU and GPU. However, in multi-application GPUs, GPU page walks performed by host MMU is inapplicable in real hardware. Also, the uses of TLB greatly reduce the energy

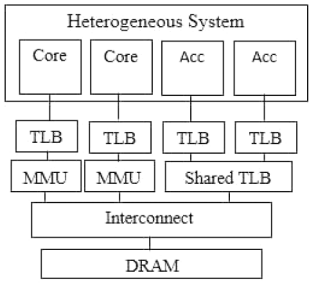

Fig. 7 *Hierarchical TLB architecture*

efficiency of accelerators. To compare the performance of this mechanism, an ideal address translation model is simulated with zero page walk latency and infinite TLB size. The evaluation shows a reduction in page walk latency and 93.6% achievement in performance.

FPGA-based hardware accelerators for the deep neural network (DNN) fail to allocate on-chip memory efficiently to process the complex layer structure of DNN. Under-utilisation of on-chip memory degrades the overall DNN performance. Further, different layers of DNN are deal with diverse computational behaviour. Prior works use uniform memory management technique for the entire data of all the layers of DNN and hence fails to maximise the performance under complex data dependency among DNN layers. To address this issue, Wei *et al.* [88] proposed a layer conscious memory management (LCMM) system for FPGA-based DNN accelerators. This scheme exploits the complex layer structures and improves the performance by efficiently utilising the on-chip memory. LCMM has adopted an algorithm for on-chip and off-chip memory allocation based on memory lifetime of data accessed by different diverse layers. In addition, the authors have proposed a buffer sharing and buffer splitting design to improve the on-chip memory utilisation. To reduce the weight loading latency, a weight buffer prefetching technique is integrated and works coordinately with the three aforesaid techniques for performance improvement. The performance benefit and resource utilisation by LCMM is evaluated for three DNN models such as ResNet-152, GoogleNet, and Inception-v4. Vivado HLS tool is used for design implementation and the experiment is conducted on the Xilinx VU9P FPGA. LCMM achieves an average of 1.36× performance improvement compared to the state-of-the-art accelerator designs.

A variety of DNN models is available for different machine learning applications. Among them, the recently developed CapsuleNets [89] model demands more computations and memory requirements for their multi-dimensional capsules structure. Marchisio and Shafique [90] proposed an accelerator design for CapsuleNet inference with a hierarchical on-chip memory structure to minimise the off-chip memory access and hence the overall power consumption. The study states that 96% of the total energy has been consumed during data transfer to/from on-chip and off-chip memory. Therefore, this work presents a detailed analysis of total memory requirements and energy consumption by the CapsuleNet inference accelerator. Further, a multi-banked, power-gated on-chip memory architecture is proposed for efficient power consumption while moving data to the accelerator. The proposed on-chip and off-chip memory hierarchy save 66% of the total energy compared to the state-of-the-art fully on-chip memory architecture without degrading the overall performance. Utilisation of on-chip memory depends on the number of operations and the power-gating further reduces the power consumption of the non-utilised memory section.

In heterogeneous computing systems, the off-chip accelerator unit provides performance benefits; however, it introduces overheads in parallelism while sharing the memory with general-purpose computing cores. Further, the overhead and complexity of interfacing CPU with FPGA-based accelerators decrease the overall speed-up of the systems. The emergence of big-data computation exacerbated these issues. The Coherent Accelerator Processor Interface (CAPI) [91] addresses the inefficiencies and facilitates the accelerator as a part of the system memory coherence protocol [92]. The high bandwidth, low latency OpenCAPI device operates in the virtual address spaces of the applications and improves accelerator performance. Also, the Intel's QPI (Quick Path Interconnect) [93] technology provides support for virtual address translation in systems where MCPs and accelerators share access to memory and I/O devices [94].

In Table 2, the performance improvement of different memory-centric accelerator architecture running on various memory-intensive applications is shown. Further, principal memory operations performed by the accelerators are also mentioned. These architectures enhance the performance of heterogeneous systems by mitigating memory interference.

3.3 Processing-in-memory

In the Big-Data era, high-cost data movement creates a major bottleneck in system performance. Recent advances in PIM technology give a potential solution to analyse a massive dataset within a limited period by reducing the unnecessary data movement. This section surveys the different architectures and performance analysis of PIM systems for different applications.

Farmahini-Farahini *et al.* [30] proposed a DRAM accelerator architecture (DRAMA) to improve energy efficiency while transferring data between storage and processing units in data-intensive applications. In this architecture, small, low power 3D stacked coarse grain reconfigurable accelerators (CGRAs) are placed at the top of the 2D DRAM die through TSVs (through silicon via) without changing their device architecture, processor and processor–memory interface. Each CGRA is composed of functional units, small memory coupled with configurable routing switches to store intermediate data and the memory controller (MC) to schedule the memory request to the DRAM device. TSV based, high bandwidth internal I/O buses are used between DRAM and processor for normal DRAM operations. DRAMA is designed to connect each CGRA to its associated DRAM device only via input/output buffering and logic followed by TSVs and operates in parallel with other CGRA–DRAM pair. The processor broadcasts the data over the I/O interface to CGRAs without interfering with the processor-DIMM interface. Data transmission between the CGRA layer and global I/O (GIO) bus is carried out by TSVs. The processor side MC and CGRA side MC do not allow the processor and CGRA to operate simultaneously on the same dataset.

They have evaluated the efficiency of DRAMA performance by executing both embedded applications and scientific applications from the San Diego Vision suite [95] and the Parboil suite [96], respectively. CGRA executes the computationally intensive portions of both the applications, and the rest are executed by the baseline processor. This architecture increases energy efficiency while transferring data in the hierarchical memory system and improves performance speed by up to 18×. The limitation of this work is that the core processor could not utilise the internal memory bandwidth for data access and to rely on the programmer. This system requires application-specific memory mapping.

To minimise the area, power, and thermal overhead, Ahn *et al.* [34] proposed a novel PIM architecture that can work with existing programming models, cache coherence protocols and virtual memory mechanisms. This architecture monitors the locality of data access and determines where to process the PIM operations, either in memory or in the host processor. By utilising the benefits of on-chip cache locality, this architecture improves the performance where applications demand high data locality. In this paper, they have introduced a set of PIM-enabled instructions (PEIs) that can work with the designed architecture to achieve significant speed-up in computation. PEI's create an illusion as there were host processor instructions when the PIM operations are executed. In this architecture, HMC is used as baseline memory and PCU (PEI computation unit) and PMU (PEI management unit) are two major components. PCUs are integrated with each host processor and each vault of HMC individually to execute the PEIs and the PMU located adjacent to the last level shared cache manages PEI execution in different PCUs. Moreover, this architecture is independent of the properties of any memory technology and can be adopted easily by other memory technology.

Table 2 Performance comparison of different hardware accelerator architectures

Reference	Design on	System performance improvement	Evaluated application	Principal operation
Che *et al.* [22]	FPGA	Cycle count: 262K	Gaussian elimination	—
—	GPU	Cycle count: 746K	—	—
—	FPGA	Cycle count: 83	—	—
—	GPU	Cycle count: 580K	DES	—
—	FPGA	Cycle count: 20K	—	—
—	GPU	Cycle count: 30K	Needleman–Wunsch	—
Duarte and Wong [25]	Simulation only	Speed up: 2.96× to 4.61×	copy_32, copy_64, copy_32 × 2, copy_32 × 4, and glibc memcpy	Accelerating cache based memory copy
Chen *et al.* [23]	CPU + GPU	Speed-up: 7.34×	KM	Mapreduce
—	—	Speed-up: 3.91×	WC	—
—	—	Speed-up: 3.25×	NBC	—
—	—	Speed-up: 28.68×	MM	—
—	—	Speed-up: 8.96×	kNN	—
Akin *et al.* [24]	DRAM	Speed-up: 6×	FFT algorithm	Optimising DRAM memory access pattern
—	—	Power efficiency: 6.5×	—	—
Choi *et al.* [26]	Cache	Speed-up: 6.14×	Data parallel program	Parallel memory access via multi-port cache
Zhang *et al.* [85]	FPGA	Speed-up: 17.42× (1 thread)	—	Optimisation of memory access
—	—	Speed-up: 4.8× (16 thread)	—	—
Chen *et al.* [27]	ASIC	Speed up: 35 frames/s	AlexNet	Minimising data movement between on-chip and off-chip memory
—	—	Speed up: 0.7 frames/s	VGG.16	—
Cong *et al.* [87]	FPGA	Page walk latency: 58 cycles	Ideal address translation	Accelerate address translation
—	—	Performance overhead: 6.4%	—	—
Wei *et al.* [88]	FPGA	Performance improvement: 1.36×	ResNet-152, GoogleNet and inception-v4	Efficient on-chip memory allocation
Marchisio and Shafique [90]	On-chip memory	Energy reduction: 66%	CapsuleNet	Minimise the on-chip memory size and off-chip memory access

Performance evaluation of proposed architecture is carried out with ten data-intensive workloads in four different system configurations: Host-Only, PIM-Only, Ideal-Host, and Locality-Aware. These configurations are based on PEIs execution site of the architecture. In processing a large input, PIM-Only achieves better speedup over Ideal-Host, whereas the average performance of PIM-Only degrades while processing small input. Moreover, in large input processing, PIM-Only configuration consumes less off-chip bandwidth as the computation is done inside memory but the off-chip bandwidth consumption increases for small inputs as it uses large on-chip caches. In graph processing workload with medium input size, Locality-Aware configuration yields better speed-up benefits than Host-Only and PIM-Only. For all input sizes, Locality-Aware claims the best performance in energy efficiency due to its ability to adapt PEI execution to data locality. Unlike high energy consumption in PIM-Only, locality-aware minimises DRAM access by executing the PEIs at the host-side and hence the energy. Locality-Aware saves energy in large input processing by reducing off-chip traffic and execution time. Even though this work provides better bandwidth efficiency than off-chip memory, it still cannot exploit the maximum internal bandwidth available inside a DRAM chip.

Pointer chasing is a series of irregular memory access patterns used in linked data structure traversal. In data-intensive applications, pointer chasing is sensitive to latency. Hsieh *et al.* [31] present an in-memory accelerator architecture IMPICA inside the 3D-stacked main memory which accelerates the pointer chasing operation and achieves high parallelism and improved performance on address translation within DRAM. The IMPICA (In Memory Pointer Chasing Accelerator) design can eliminate high latency and transfers data sequentially between the processor and main memory. IMPICA traverse virtual memory address pointers without the help of TLB and page-walkers of CPU. This architecture has a specialised core consist of an *address engine* to generate pointer specified address and an *access engine* to access the generated address in memory. The performance of the address engine is fast enough to process multiple pointer chasing operations efficiently and thus IMPICA core can process faster than multiple cores. The pointer chasing operation is initialised by the CPU and enquiring about the request code. The address engine monitors the request queue, load and executes the pointer chasing code into its instruction RAM. IMPICA successfully addresses two fundamental threats. Firstly, the parallelism challenge is solved by separating the address generation engine from memory accesses engine. Secondly, the address translation challenge is solved by using a region-based page table and allocating data structures into virtual memory regions.

The author evaluates the performance of pointer chasing with IMPICA on the linked list, Hash table, and B-tree and found improvement by 92, 29, and 18%, respectively. To evaluate the proposed design, a gem5 full system simulator with DRAMSim2 is used. They have also evaluated the performance improvement in terms of transaction latency and throughput on real data-intensive workloads. IMPICA design can boost up the transaction throughput and lessen the access latency. Further, it also improves the energy efficiency of the system, both for the microbenches and real-time workloads. However, implementing the work at minimum possible cost is still a major challenge for commercial consumer devices.

Azarkhish *et al.* [32] propose a new PIM architecture having Smart Memory Cube (SMC) to enhance the performance of logic-based die in the HMC. Instead of OS managed TLB, the authors have implemented a simple controller for address mapping in the page table. This PIM architecture claims that the host processor can withstand cache thrashing and can utilise the memory bandwidth up to the fulfilment of its requirement. In addition, a software-based cache management scheme is developed on shared memory

for synchronisation of the host processor and PIM. Also, they have developed a simulation software SMCSim based on gem5 for system analysis. The PIM architecture has a Scratchpad Memory (SPM), a DMA engine, a Translation Look-aside Buffer (TLB) and an MMU coupled with the logic layer. TLB is responsible for initialising address translation in PIM and DMA engine to enhance PIM's memory access by transferring data from DRAM to SPM. The frequency of the host processor is the same as the frequency of the single-core processor of PIM. Further, this architecture adopted a memory virtualisation scheme that allows PIM to access virtual memory of a user without a memory copy.

To evaluate the acceleration of PIM execution, they have chosen four large-scale graph processing applications. The large scale graphical benchmarks adopted for the evaluation of PIM acceleration are Average Teenage Follower (ATF), Breadth-First Search (BFS), PageRank (PR) and Bellman-Ford Shortest Path (BF). The result shows the improvement in overall performance and energy efficiency compared to both the host processor and host-side accelerator. Only simulations were done for performance and energy evaluation of this system architecture. This architecture possesses extremely slow performance in real-world applications, as the single simulation with a large dataset typically takes hours or even days.

Transferring data over the memory channel leads to long latency and delays other data transfers. To reduce this latency and to minimise energy and performance overhead, Pattnaik *et al.* [35] proposed a PIM-based GPU architecture where 3D-stacked memory chip is interconnected to a conventional GPU chip via a memory link on the same die. There are two processors in this architecture: large and powerful primary GPU (GPU-PIC) and small and less powerful GPU cores (GPU-PIM) placed under main memory. The authors have designed two runtime code scheduling approaches that can recognise and transfer the kernels into the cores of GPU-PIM and can schedule multiple kernels simultaneously on the GPU cores. They have also developed an execution time prediction model for concurrent kernel management mechanisms to execute multiple kernels in both the GPU in PIM-assisted GPU architecture. However, in this PIM architecture, CPU cores process the PIM kernels that are part of the application function. Comprehensive evaluation report of both techniques provides a significant improvement in performance (on average by 25 and 42%) and energy efficiency (by 28 and 27%) for 25 different GPU applications. In commercial consumer devices, the addition of GPU-PIM cores to the memory is still a challenging task in terms of area and energy constraints.

Chi *et al.* [36] present a PIM architecture called PRIME based on non-volatile resistive random access memory (ReRAM). Uses of ReRAM in PRIME decreases the area overhead as it utilises the memory arrays both for storing network parameters and computing the dot product operations. To accelerate the computation of NN applications in memory, PRIME modifies the existing memory peripheral circuit and partitions the ReRAM bank into memory subarrays, full function (FF) subarrays and buffer subarrays. Instead of adding logic on 3D-stack memory, with the help of the PRIME controller, the FF subarray executes NN computation in computation mode and serves as a typical memory in memory mode. To enable accelerated NN computation researchers had described different microarchitecture and circuit design for the FF subarray, the buffer subarrays, and the PRIME controller. They have developed an interface framework for PRIME to configure FF subarrays for different NN applications and to optimise data mapping and allocation mechanisms in run time.

To evaluate the performance of PRIME design, MIBench benchmarks are used in six different NN designs. They compared the performance of PRIME with CPU-only, NPU (Network Processor Unit) co-processor and NPU PIM-processor. The performance evaluation shows that the PRIME gain almost 4× speedup benefits over NPU PIM processor and improves energy consumption by ∼895×.

In recent times, the neural network (NN) simulation plays a vital role in solving big-data computation problems. Oliveira *et al.* present a NN simulator (NIM) [33] inside 3D stacked memories to execute deep learning applications. Neural simulator explores available application parallelism with HPC devices which are composed of a mix of MCP, GPU and FPGA based accelerator. This PIM architecture introduces computational RAM and provides lower access latency and higher memory bandwidth. Further, this work is based on a reconfigurable HMC accelerator capable of vector processing directly in memory. Complex, reconfigurable functional units are implemented in this architecture to execute NN related operations. This mechanism is composed of 2048 functional units operating at 1 GHz frequency and a register unit with 8×16 registers of 32 bits per vault. To evaluate this work, a cycle-accurate HMC simulator is used. In their design, the authors have considered two different neural models for computation: Hodgkin–Huxley and Izhikevich models. To study the experimental procedure of NIM and its results, it is important to understand that as a N/L ratio (total number of neurons stimulated in an NN by the total number of neural layers) increases, the number of connections between neurons at different layers increases resulting in an excess consumption of system computational power.

Results of Izhikevich model reveal that in 1 ms time window, the NIM mechanism provides almost similar performance behaviour up to 32 N/L configurations. However, the number of network layers becomes insignificant above 64 N/L configuration of NN. The NIM mechanism of the baseline system (Intel SandyBridge processor microarchitecture) can simulate the same amount of neurons with the same NN configuration in a 2× faster time. During the Hodgkin–Huxley model simulation with the time limit of 50 µs, baseline showed a better result. Within 50 µs NTS, the baseline can simulate up to 2304 neurons (32 N/L, 72 L) whereas NIM can simulate up to 2560 neurons (64 N/L, 40 L) within the same time. In 1 ms experiments, the baseline could simulate up to 65,536 neurons (32 N/L, 2048 L), while the proposed device can simulate 49,152 neurons (64 N/L, 768 L). To measure energy consumption, the McPat tool is configured for Hodgkin–Huxley applications. Compared to the baseline system, NIM consumes less energy and NN with a high N/L ratio reduces unnecessary data movement. In contrast, as the number of neural layers increases, NIM could not provide a significant improvement in energy consumption. The presented NIM module provides acceleration for NN of significant sizes and reduces overall system energy consumption.

Prefetching is one of the well-studied sequence prediction technique that hides memory latency and enhance cache performance in modern HPC. Hardware-based data prefetching exploits the Spatio-temporal memory access pattern and anticipate what data is needed and move it to the cache ahead of time. Xu *et al.* [97] proposed a data prefetcher for PIM and CPU cores based on heterogeneous systems with the benefit of coarse-grained coherence. Coherence manages the memory region by keeping memory address recording on a global table that shared by both PIM and CPU cores. An entry is generated in the table when PIM has sent any data request and the host processor writes back the dirty data on the same memory region. Also, it that time no host processor can access data from the same memory region until the completion of the PIM's task. Although coherence has several advantages in heterogeneous systems, it degrades the system performance due to the limitation of simultaneous data access in the same memory region by PIM and host processors. Further, a cooperative prefetcher is used in this work to reduce the conflict in prefetching between host processors and PIM processors. Hardware prefetching unit PIMCH is consist of a memory access map and a predictor unit. To evaluate the PIMCH architecture, SPEC CPU 2006 benchmark suite is chosen and compares the evaluation results with the other data prefetchers. PIMCH provides 44.3 and 28.3% performance improvement in PIM based system and host-only system, respectively. Also, PMCH reduces on average 16.4% off-chip traffic and prevents miss-prefetching predictions by 84%.

Jeong *et al.* [98] proposed a multi-channel PIM architecture to mitigate the gap between the performance of processor and memory in memory-intensive applications. This architecture is suitable to communicate data among different PIM modules via several channels. Further, this work presents the optimisation of data layout, so that all channels of the PIM architecture are utilised

Table 3 Performance comparison of different PIM architectures

References	Configuration	System performance improvement	Evaluated application
Farmahini-Farahini *et al.* [30]	CGRA at the top of DRAM	Speed-up: 18.4×	SIFT, tracking (TRCK), disparity map (DISP), LBM and MRI-Gridding (MRIG)
—	—	Energy consumption: 66–95%	—
Ahn *et al.* [34]	PCU and PMC inside HMC	• Locality-Aware achieves 12 and 11% speed-up over host-only and PIM-only, respectively.	10 workloads of large-scale graph processing, in memory data analytics, machine learning and data mining
—	—	• minimise area, power and thermal overheads.	—
Hsieh *et al.* [31]	Specialised core inside 3D stack main memory	Speed-up: 92%	Linked list
—	—	Energy consumption: 41%	—
—	—	Speed-up: 29%	—
—	—	Energy consumption: 23%	Hash table
—	—	Speed-up: 18%	B-tree
—	—	Energy consumption: 10%	—
Azarkhish *et al.* [32]	Smart memory cube	Speed-up: 2× (host)	ATF, BFS, PR and BF
—	—	Speed-up: 1.5× (accelerator)	—
—	—	Energy consumption: 70% (host)	—
—	—	Energy consumption: 55% (accelerator)	—
Pattnaik *et al.* [35]	Auxiliary GPU cores in the logic layer of 3D-stacked DRAM	For kernel offloading, speed-up: 25%	25 GPU applications
—	—	Energy efficiency: 28%	—
—	—	For concurrent kernel management,	—
—	—	Speed-up: 42%	—
—	—	Energy efficiency: 27%	—
Chi *et al.* [36]	ReRAM based main memory	Speed up: 2360×	Six machine learning applications: two CNNs, three multilayer perceptrons (MLPs) and VGG-D
—	—	Energy consumption: 895×	—
Oliveira *et al.* [33]	NIM inside 3D stacked memory	Simulating capacity: 2×	Hodgkin–Huxley model and Izhikevich models
Xu *et al.* [97]	PIM cores with data prefetcher	Speed-up: 44%	SPEC CPU 2006
—	—	Off-chip traffic: 16%	—
—	—	Mis-prefetching predictions: 84%	—
Jeong *et al.* [98]	Multi-channel PIM architecture	Speed-up: 393%	—
Dai *et al.* [99]	OVBs on HMC cube	Speed-up: 5×	Large-scale graph processing

efficiently. Parallel computing is enabled by placing processing cores on each channel to operate independently. To reduce the data transfer, multi-channel PIM communication is implemented without interfering with the host processor and the memory controller. In this architecture, the memory controller is placed near memory to reduce the data transfer overheads caused by SIMD (single-instruction-multiple-data) processors. Data layout for vector operation is analysed and it depends on the channel interleaving policy. This work presents four data layouts that describe how data pages are distributed and allocated across the PIM channels. Multi-channel PIM architecture is implemented on the gem5 simulator. The evaluation report reveals that data layout where all pages are evenly distributed and channel interleaving policy is applied into four channels has achieved 393% faster speed-ups than the other data layouts. Optimised data layout addresses the communication overhead of PIM-to-PIM architecture successfully and increases execution time. However, communication overhead in address translation on four channels is expected.

Graph processing applications demand very high bandwidth requirement for data access. To meet the challenges of continuously scaled up graph computing, Dai *et al.* [99] proposed a PIM architecture GraphH for graph processing on the HMC. In this paper, the authors have addressed four crucial challenges that limit the performance of large-scale graph processing on conventional graph processing architecture. They are random access, poor locality, unbalanced workloads, and heavy conflicts. Integrated with on-chip vertex buffers (OVBs), GraphH architecture can utilise the maximum local bandwidth. This architecture does not undergo a random access pattern as the processor cores can directly access the OVBs. The locality issue is addressed by proposing a reconfigurable double-mesh connection (RDMC) which can provide exclusive bandwidth on each individual data path. Further, the index mapping interval-block (IMIB) algorithm is proposed for vertex-cut partitioning and balances the workloads of processing cores. Finally, RIP (round interval pair) scheduler reduces the data communication cost between HMCs and avoids writing conflict. To further improve the performance of GraphH, this work adopts an optimisation scheme for on-chip data reuse. Implementation of OVBs achieves 4.58× speedup benefits over direct access of DRAM layers. The experimental results on large scale graphs demonstrate that GraphH architecture obtains up to 5× speed-up over Tesseract [15] on different OVB sizes. However, the cost of communications among different HMC cubes dominates the system performance.

Table 3 summarises some PIM architectures and their performance improvement on different workloads. In big-data processing and embedded applications, the PIM approach can potentially reduce overall memory access latency by placing processing modules near memory and significantly reduce system energy consumption.

4 Discussion and open research issue

This study explores an extensive range of architecture of DRAM designs, hardware accelerators, and PIMs in the context of memory management. The main focus of this section is to discuss open research issues related to these techniques that can be used for further study.

In response to the growing disparity of processor and memory speed, new DRAM designs aim to decrease memory access time. This bottleneck is still in critical form because of (i) limited latency reduction in the last several years; (ii) improving memory density introduces more errors and increasing manufacturing process variation; (iii) prevention of adding more and/or wider buses due to limited pin count in memory chip; (iv) enormous need of data-intensive applications; (v) increase in overall system energy consumption for data movement [100]. We can successfully mitigate the challenges of memory by modifying them at low cost and low overhead. A trade-off among cost, latency, and bandwidth in DRAM designs impact the overall system performance improvement. Based on the review and discussions that we have delved into, we can explore the below-listed areas for further improvement in instructions per cycle (IPC), memory power and system power.

(i) The use of faster buses and internal bank memory array are more likely solutions in case of data intensive workloads. Partitioning of DRAM increases parallelism but decreases the bandwidth of data-bus of each subset. Current memory buses are inadequate to cover large memory systems where bandwidth is considered as more sensitive affair than latency.
(ii) Advancement in scheduling the DRAM requests can avoid bus conflicts in MCPs.
(iii) In modern DRAM designs, multiple banks serve multiple memory requests, parallelly exacerbating high latency. Taking advantage of subarray-level parallelism in a DRAM bank could be a propitious work in the future to overcome the shortcomings of DRAM bank conflicts.

In recent years, a surge of interest is found in the development of new non-volatile memory (NVM) because of its high density and low power consumption [101]. However, NVM techniques have worse write performance than DRAM and integration of NVM into memory system introduces new congestion. Therefore, NVM brings many interesting research directions to improve system performance. Architectural development in both hardware and software schemes of NVM can add new features.

In heterogeneous systems, the performance can be increased further by efficiently distributing the workloads of parallel computing to CPUs, GPUs, and FPGAs. The suitability of such hardware is task-specific. For example, GPUs are well suited for massively parallel tasks. Tasks that are largely sequential are better handled by CPUs with out of order execution, large caches, and pipelines. Hardware accelerator unit integrated with processing cores can address tasks that require a large number of data transfers to and from external memory. Computational intensive tasks are efficiently performed by FPGA based accelerators. Existing hardware accelerator architecture focuses more on the reduction of energy consumption than performance improvements. In heterogeneous computing, the following are a few shortcomings that can address efficiently in the future:

(i) Lacking in programmability and reconfiguration limits the applicability of an FPGA-based accelerator in achieving the best performance.
(ii) In the big-data processing scenario, minimisation of data transfer overhead and maximisation of parallelism could be addressed more efficiently while designing accelerator architecture.
(iii) The need for costly and time-consuming optimisation steps should also be avoided as much as possible.

Further, it is essential to explore novel memory-centric accelerator designs for complex, real-time applications with greater feasibility and intelligent scheduling policies. Optimisation in resource allocation and size of on-chip memory for data-intensive applications on the embedded platform become potential research topics. Although there is number of FPGA-based accelerators for deep learning applications, future development can be expected in the following issues:

(i) *Computational optimisation:* Very few works are focused on the faster processing of activation function. Mainstream research is based on only matrix operation.
(ii) *Data optimisation:* Bit-wise precision of network parameters by lowering the bit width may improve the performance while designing accelerator architecture.

PIM is a rapidly rising technique to mitigate the memory wall crisis. High-cost data movement between DRAM and CPU is effectively addressed by in-memory processing logic. DRAM offers low bandwidth because of the narrow memory channel between DRAM and the CPU. Whereas the logic layer of PIM architecture has access to high internal bandwidth within DRAM and at the same time, reduces system energy consumption. However, in enabling PIM at the system level, few design challenges are introduced while adding compute logic to DRAM. As the PIM logic has high latency access to common CPU structure, address translation and cache coherence issues are arising in a wide range of in-memory processing units. To determine the physical address during execution, the PIM processor has to perform address translation because there is no efficient way of accessing translation lookaside buffer by PIM logic. Also, PIM cores should support the cache coherence protocol so that both the CPU and PIM cores can utilise the updated versions of data. Prior solutions to these challenges limit the performance and energy of the system or put restrictions on the programming model. In addition, future research should upgrade the PIM architecture to work effectively with the emerging memory technologies such as phase-change memory (PCM), spin-transfer torque magnetic RAM (STT-MRAM), metal-oxide resistive RAM (RRAM), and memristors. PIM can take the potential advantages of these memories which offer much greater memory capacity and high internal memory bandwidth. Future research can focus on designing intelligent data mapping schemes. Static and adaptive data mapping maximise the benefit of PIM and facilitate easier PIM execution at runtime. Another important research direction in PIM technology is to provide runtime scheduling support. When, what, and how to execute parts of application code in PIM need focus to maximise the system performance. For easy deployment of PIM architecture in real systems, efficient and novel solutions to the aforementioned problems are required.

Another potential research direction lies in accelerating machine learning and Big-Data processing by using FPGA devices. As an interesting alternative to GPUs, FPGA-based solutions are generally energy-saving and can be embedded into smart devices. To enable the shared coherent virtual memory across CPUs and accelerators is a critical challenge in a heterogeneous system. There is a comprehensive research scope on techniques that can effectively handle the TLB misses by GPUs and FPGA based accelerators. There is a shortage of work on address translation, particularly for heterogeneous memory systems. Furthermore, research effort can be put on the study of interconnects and interfacing of FPGA accelerators with the shared memory processors and other I/O devices.

5 Conclusion

In an HPC system, the increasing ratio of processor speed and memory speed degrades overall computer performance. In addition, data movement in big-data processing is a high-cost operation as it requires a high latency and consumes a considerable amount of energy. To effectively address these challenges, many processors and memory designs, architectures, and optimisation techniques are developed. However, no such technology could efficiently balance cost, performance, power consumption, and complexity. Fast, reconfigurable and self-adaptive memory

management schemes are essential in the high speed, big-data processing scenario.

In this paper, we have analysed three promising memory management techniques to maximise the overall system performance of heterogeneous systems. These memory management techniques especially help to reduce unnecessary data movement and increase memory performance. We analyse the summary of this survey in the following two aspects: firstly, the study of DRAM, memory-centric hardware accelerator and PIM architecture on various memory intensive and embedded applications and secondly, discussing the challenges and their potential solutions. It is essential to replace the existing schemes with enhanced efficiency by leveraging emerging techniques. Further, we have discussed the impact of different DRAM designs on memory performance, integration of hardware accelerator for memory operations, and the PIM execution in heterogeneous systems. We hope that the ideas and challenges discussed in this paper can motivate other researchers to develop promising solutions that can ease the implementation of memory management.

6 References

[1] 'Memory access vs cpu speed'. Available at: http://www.oempcworld.com, accessed September 2019

[2] Mahapatra, N.R., Venkatrao, B.: 'The processor-memory bottleneck: problems and solutions', *Crossroads*, 1999, **5**, (3), pp. 1–8

[3] Li, C., Song, S.L., Dai, H., *et al.*: 'Locality-driven dynamic GPU cache bypassing'. Proc. Int. Conf. Supercomputing, Frankfurt, Germany, 2015, pp. 67–77

[4] Xie, X., Liang, Y., Wang, Y., *et al.*: 'Coordinated static and dynamic cache bypassing for gpus'. Int. Symp. High Performance Computer Architecture (HPCA), Burlingame, CA, USA, 2015, pp. 76–88

[5] Chen, X., Chang, L.W., Rodrigues, C.I., *et al.*: 'Adaptive cache management for energy-efficient GPU computing'. Proc. Int. Symp. Microarchitecture, Cambridge, UK, 2014, pp. 343–355

[6] Li, A., van den Braak, G.J., Kumar, A., *et al.*: 'Adaptive and transparent cache bypassing for GPUs'. Int. Conf. High Performance Computing, Networking, Storage and Analysis, Austin, TX, USA, 2015, pp. 1–12

[7] Li, D., Rhu, M., Johnson, D.R., *et al.*: 'Priority-based cache allocation in throughput processors'. Int. Symp. High Performance Computer Architecture (HPCA), Burlingame, CA, USA, 2015, pp. 89–100

[8] Qureshi, M.K., Jaleel, A., Patt, Y.N., *et al.*: 'Adaptive insertion policies for high performance caching', *Comput. Archit. News*, 2007, **35**, (2), pp. 381–391

[9] Rixner, S., Dally, W.J., Kapasi, U.J., *et al.*: 'Memory access scheduling', *Comput. Archit. News*, 2000, **28**, (2), pp. 128–138

[10] Ebrahimi, E., Miftakhutdinov, R., Fallin, C., *et al.*: 'Parallel application memory scheduling'. Proc. Int. Symp. Microarchitecture, Porto Alegre, Brazil, 2011, pp. 362–373

[11] Bakhoda, A., Yuan, G.L., Fung, W.W., *et al.*: 'Analyzing CUDA workloads using a detailed GPU simulator'. Int. Symp. Performance Analysis of Systems and Software (ISPASS), Boston, MA, USA, 2009, pp. 163–174

[12] Ahn, J.H., Leverich, J., Schreiber, R., *et al.*: 'Multicore DIMM: an energy efficient memory module with independently controlled DRAMs', *Comput. Archit. Lett.*, 2009, **8**, (1), pp. 5–8

[13] Ahn, J.H., Jouppi, N.P., Kozyrakis, C., *et al.*: 'Improving system energy efficiency with memory rank subsetting', *Trans. Archit. Code Optim.*, 2012, **9**, (1), pp. 4.1–4.28

[14] Chandrasekar, K., Goossens, S., Weis, C., *et al.*: 'Exploiting expendable process-margins in DRAMs for run-time performance optimization'. Proc. Conf. European Design, Automation & Test, Dresden, Germany, 2014, pp. 173–178

[15] Ahn, J., Hong, S., Yoo, S., *et al.*: 'A scalable processing-in-memory accelerator for parallel graph processing'. Int. Symp. Computer Architecture (ISCA), Portland, OR, USA, 2015, pp. 105–117

[16] Chang, K.K., Nair, P.J., Lee, D., *et al.*: 'Lowcost inter-linked subarrays (LISA): enabling fast inter-subarray data movement in DRAM'. Int. Symp. High Performance Computer Architecture (HPCA), Barcelona, Spain, 2016, pp. 568–580

[17] Son, Y.H., Seongil, O., Ro, Y., *et al.*: 'Reducing memory access latency with asymmetric DRAM bank organizations', *Comput. Archit. News*, 2013, **41**, pp. 380–391

[18] Chatterjee, N., Shevgoor, M., Balasubramonian, R., *et al.*: 'Leveraging heterogeneity in DRAM main memories to accelerate critical word access'. Int. Symp. Microarchitecture (MICRO), Vancouver, BC, Canada, 2012, pp. 13–24

[19] Seongil, O., Son, Y.H., Kim, N.S., *et al.*: 'Row-buffer decoupling: a case for low-latency dram microarchitecture'. Int. Symp. Computer Architecture (ISCA), Minneapolis, MN, USA, 2014, pp. 337–348

[20] Karam, R., Paul, S., Puri, R., *et al.*: 'Memory-centric reconfigurable accelerator for classification and machine learning applications', *J. Emerg. Technol. Comput. Syst.*, 2017, **13**, (3), pp. 34.1–34.2

[21] Peemen, M., Setio, A.A., Mesman, B., *et al.*: 'Memory-centric accelerator design for convolutional neural networks'. Int. Conf. Computer Design (ICCD), Asheville, NC, USA, 2013, pp. 13–19

[22] Che, S., Li, J., Sheaffer, J.W., *et al.*: 'Accelerating compute-intensive applications with gpus and fpgas'. Symp. Application Specific Processors (SASP), Anaheim, CA USA, 2008, pp. 101–107

[23] Chen, L., Huo, X., Agrawal, G.: 'Accelerating MapReduce on a coupled CPU-GPU architecture'. Proc. Int. Conf. High Performance Computing, Networking, Storage and Analysis, Los Alamitos, CA, USA, 2012, pp. 25–35

[24] Akin, B., Franchetti, F., Hoe, J.C.: 'Understanding the design space of dramoptimized hardware fft accelerators'. Int. Conf. Application-specific Systems, Architectures and Processors (ASAP), Zurich, Switzerland, 2014, pp. 248–255

[25] Duarte, F., Wong, S.: 'Cache-based memory copy hardware accelerator for multicore systems', *IEEE Trans. Comput.*, 2010, **59**, (11), pp. 1494–1507

[26] Choi, J., Nam, K., Canis, A., *et al.*: 'Impact of cache architecture and interface on performance and area of FPGA-based processor/parallel-accelerator systems'. Int. Symp. Field-Programmable Custom Computing Machines (FCCM), Toronto, Canada, 2012, pp. 17–24

[27] Chen, Y.H., Emer, J., Sze, V.: 'Eyeriss: a spatial architecture for energy-efficient dataflow for convolutional neural networks', *Comput. Archit. News*, 2016, **44**, (3), pp. 367–379

[28] Oliveira, J., Geraldo, F.: 'A generic processing in memory cycle accurate simulator under hybrid memory cube architecture'. Master thesis, Federal University of Rio Grande do Sul, 2017

[29] Gokhale, M., Holmes, B., Iobst, K.: 'Processing in memory: the terasys massively parallel pim array', *Computer. (Long Beach Calif.)*, 1995, **28**, (4), pp. 23–31

[30] Farmahini-Farahani, A., Ahn, J.H., Morrow, K., *et al.*: 'DRAMA: an architecture for accelerated processing near memory', *Comput. Archit. Lett.*, 2015, **14**, (1), pp. 26–29

[31] Hsieh, K., Khan, S., Vijaykumar, N., *et al.*: 'Accelerating pointer chasing in 3D-stacked memory: challenges, mechanisms, evaluation'. Int. Conf. Computer Design (ICCD), Phoenix, AZ, USA, 2016, pp. 25–32

[32] Azarkhish, E., Rossi, D., Loi, I., *et al.*: 'Design and evaluation of a processing-in-memory architecture for the smart memory cube'. Int. Conf. Architecture of Computing Systems, Nuremberg, Germany, 2016, pp. 19–31

[33] Oliveira, G.F., Santos, P.C., Alves, M.A., *et al.*: 'NIM: an HMC-based machine for neuron computation'. Int. Symp. Applied Reconfigurable Computing, Delft, The Netherlands, 2017, pp. 28–35

[34] Ahn, J., Yoo, S., Mutlu, O., *et al.*: 'PIM-enabled instructions: a low-overhead, locality-aware processing-in-memory architecture'. Int. Symp. Computer Architecture (ISCA), Portland, MA, USA, 2015, pp. 336–348

[35] Pattnaik, A., Tang, X., Jog, A., *et al.*: 'Scheduling techniques for GPU architectures with processing-in-memory capabilities'. Int. Conf. Parallel Architecture and Compilation Techniques (PACT), Haifa, Israel, 2016, pp. 31–44

[36] Chi, P., Li, S., Xu, C., *et al.*: 'PRIME: A novel processing-in-memory architecture for neural network computation in reRAM-based main memory'. Proc. Int. Symp. Computer Architecture, Seoul, Republic of Korea, 2016, pp. 27–39

[37] Zheng, Z., Wen, J., Liu, S.: 'Introduction to the special section on heterogeneous computing era', *Comput. Electr. Eng.*, 2017, **60**, pp. 45–47

[38] Carretero, J., García-Carballeira, F.: 'Introduction to special issue on heterogeneous architectures and high-performance computing', *Comput. Electr. Eng.*, 2013, **8**, (39), pp. 2551–2552

[39] AMD: 'Compute cores'. Whitepaper, 2014

[40] Ausavarungnirun, R., Chang, K.K.W., Subramanian, L., *et al.*: 'Staged memory scheduling: achieving high performance and scalability in heterogeneous systems', *Comput. Archit. News*, 2012, **40**, (3), pp. 416–427

[41] NVIDIA: 'GPU-based deep learning inference: a performance and power analysis'. Whitepaper, 2018

[42] Davis, B.T.: 'Modern DRAM architectures'. PhD thesis, University of Michigan, 2001

[43] Krishnapillai, Y., Wu, Z.P., Pellizzoni, R.: 'A rank-switching, open-row DRAM controller for time-predictable systems'. Euromicro Conf. Real-Time Systems (ECRTS), Madrid, Spain, 2014, pp. 27–38

[44] 'Dynamic random-access memory'. Available at: https://en.wikipedia.org/wiki/Dynamicrandomaccessmemory, accessed October 2017

[45] Reineke, J., Liu, I., Patel, H.D., *et al.*: 'PRET DRAM controller: bank privatization for predictability and temporal isolation'. Proc. Int. Conf. Hardware/Software Codesign and System Synthesis (CODES+ISSS), Taipei, Taiwan, October 2011, pp. 99–108

[46] Hansson, A., Agarwal, N., Kolli, A., *et al.*: 'Simulating dram controllers for future system architecture exploration'. Int. Symp. Performance Analysis of Systems and Software (ISPASS), Monterey, CA, USA, 2014, pp. 201–210

[47] Dashti, M., Fedorova, A.: 'Analyzing memory management methods on integrated CPU-GPU systems', *Memory*, 2017, **52**, (9), pp. 59–69

[48] Manish, A.: 'The architecture and evolution of CPU-GPU systems for general purpose computing'. Master thesis, University of California, 2012

[49] NVIDIA: 'The benefits of multiple CPU cores in mobile devices'. Whitepaper, 2010

[50] Technica: 'GPU accelerated computing'. Whitepaper, 2015

[51] NVIDIA: 'NVIDIAs next generation cuda compute architecture: Kepler GK110'. Whitepaper, 2012

[52] Martindale, J.: 'AMD Navi graphic cards could offer RTX 2070-like performance for $ 250'. Digitaltrends Whitepaper, 2018

[53] Han, F., Zhu, T., Meyer, R.: 'Basic performance analysis of NVIDIA GPU accelerator cards for deep learning applications'. AMAX Whitepaper, 2019

[54] Varghese, B.: 'The hardware accelerator debate: a financial risk case study using many-core computing', *Comput. Electr. Eng.*, 2015, **46**, pp. 157–175

[55] 'Hybrid memory cube specification 1.0'. Tech. Report, Hybrid Memory Cube Consortium, 2013

[56] 'Hybrid memory cube specification 2.1'. Tech. Report, Hybrid Memory Cube Consortium, 2014

[57] Ahn, J., Hong, S., Yoo, S., *et al.*: 'A scalable processing-in-memory accelerator for parallel graph processing', *Comput. Archit. News*, 2016, **43**, (3), pp. 105–117

[58] Jin, H.Q., Frumkin, M., Yan, J.: 'The openmp implementation of NAS parallel benchmarks and its performance'. NASA Ames Research Center, NAS-99-011, 1999

[59] McGhan, H.: 'SPEC cpu2006 benchmark suite'. Microprocessor Report, 2006

[60] 'SPEC'. Available at: http://www.spec.org/osg/jbb2000/, accessed September 2019

[61] Beamer, S., Sun, C., Kwon, Y.J., *et al.*: 'Re-architecting DRAM memory systems with monolithically integrated silicon photonics', *Comput. Archit. News*, 2010, **38**, pp. 129–140

[62] Chang, K.K.W., Lee, D., Chishti, Z., *et al.*: 'Improving DRAM performance by parallelizing refreshes with accesses'. Int. Symp. High Performance Computer Architecture (HPCA), Orlando, FL, USA, 2014, pp. 356–367

[63] '*SPEC CPU2006 benchmarks*' SPEC (Standard Performance Evaluation Corporation, USA, 2006)

[64] 'STREAM benchmark'. Available at: http://www.streambench.org/, accessed September 2019

[65] 'Tpc benchmarks'. Transaction Performance Processing Council, Available at: http://www.tpc.org/, accessed September 2019

[66] 'Dynograph'. Available at: https://github.com/sirpoovey/DynoGraph, accessed September 2019

[67] Woo, S.C., Ohara, M., Torrie, E., *et al.*: 'The SPLASH-2 programs: characterization and methodological considerations'. Proc. Int. Symp. Computer Architecture, Santa Margherita Ligure, Italy, 1995, pp. 24–36

[68] Bienia, C., Kumar, S., Singh, J.P., *et al.*: 'The PARSEC benchmark suite: characterization and architectural implications'. Proc. Int. Conf. Parallel Architectures and Compilation Techniques, Toronto, Canada, 2008, pp. 72–81

[69] 'All you need to know about mobile LPDRM'. Micron Technology, Available at: https://www.micron.com/products/dram/lpdram, accessed May 2019

[70] 'RLDRAM3 press release'. Micron Technology, Available at: http://www.icsi.com.tw/pdf/RLDRAM3-Press-Release.pdf, accessed May 2019

[71] Bhati, I., Chishti, Z., Lu, S.L., *et al.*: 'Flexible auto-refresh: enabling scalable and energy-efficient DRAM refresh reductions', *Comput. Archit. News*, 2015, **43**, (3), pp. 235–246

[72] Nguyen, K., Lyu, K., Meng, X., *et al.*: 'Nonblocking DRAM refresh', *IEEE Micro*, 2019, **39**, (3), pp. 103–109

[73] Douglass, A.J., Khatri, S.P.: 'Fast, ring-based design of 3-D stacked DRAM', *Trans. Very Large Scale Integr. Syst.*, 2019, **27**, (3), pp. 1–11

[74] Power, J., Basu, A., Gu, J., *et al.*: 'Heterogeneous system coherence for integrated CPU-GPU systems'. Proc. Int. Symp. Microarchitecture, Davis, CA, USA, 2013, pp. 457–467

[75] Hechtman, B.A., Che, S., Hower, D.R., *et al.*: 'Quickrelease: a throughput-oriented approach to release consistency on GPUs'. Int. Symp. High Performance Computer Architecture (HPCA), Orlando, FL, USA, 2014, pp. 189–200

[76] Singh, I., Shriraman, A., Fung, W.W., *et al.*: 'Cache coherence for GPU architectures'. Int. Symp. High Performance Computer Architecture (HPCA), Shenzhen, People's Republic of China, 2013, pp. 578–590

[77] Koukos, K., Ros, A., Hagersten, E., *et al.*: 'Building heterogeneous unified virtual memories (UVMs) without the overhead', *Trans. Archit. Code Optim.*, 2016, **13**, (1), pp. 1.1–1.21

[78] Zheng, T., Nellans, D., Zulfiqar, A., *et al.*: 'Towards high performance paged memory for GPUs'. Int. Symp. High Performance Computer Architecture (HPCA), Barcelona, Spain, 2016, pp. 345–357

[79] 'NVIDIA pascal architecture'. Available at: https://www.nvidia.com/en-us/datacenter/pascal-gpu-architecture/, accessed September 2019

[80] Ganguly, D., Zhang, Z., Yang, J., *et al.*: 'Interplay between hardware prefetcher and page eviction policy in CPU-GPU unified virtual memory'. Int. Symp. Computer Architecture (ISCA), Phoenix, AZ, USA, 2019, pp. 1–12

[81] Rao, J.N., Sundaresan, M.: 'Memory sharing via a unified memory architecture'. US Patent Application 16 /219,093, 2016

[82] Jones, S., Vivek, K., Jaroszynski, P., *et al.*: 'Unified memory systems and methods', US Patent Application 16/237,010, 2018

[83] Kanev, S., Darago, J.P., Hazelwood, K., *et al.*: 'Profiling a warehouse-scale computer'. Int. Symp. Computer Architecture (ISCA), Portland, OR, USA, 2015, pp. 158–169

[84] Dean, J., Ghemawat, S.: 'Mapreduce: simplified data processing on large clusters', *Commun. ACM*, 2008, **51**, (1), pp. 107–113

[85] Zhang, C., Li, P., Sun, G., *et al.*: 'Optimizing fpga-based accelerator design for deep convolutional neural networks'. Proc. Int. Symp. Field-Programmable Gate Arrays, Monterey, CA, USA, 2015, pp. 161–170

[86] Williams, S., Waterman, A., Patterson, D.: 'Roofline: an insightful visual performance model for multicore architectures', *Commun. ACM*, 2009, **52**, (4), pp. 65–76

[87] Hao, Y., Fang, Z., Reinman, G., *et al.*: 'Supporting address translation for accelerator-centric architectures'. Int. Symp. High Performance Computer Architecture (HPCA), Austin, Texas, USA, 2017, pp. 37–48

[88] Wei, X., Liang, Y., Cong, J.: 'Overcoming data transfer bottlenecks in FPGA-based DNN accelerators via layer conscious memory management'. Proc. Design Automation Conf (DAC), Las Vegas, NV, USA, 2019, pp. 1–6

[89] Sabour, S., Frosst, N., Hinton, G.E.: 'Dynamic routing between capsules'. Advances in Neural Information Processing Systems, 2017, pp. 3856–3866

[90] Marchisio, A., Shafique, M.: 'Capstore: energy-efficient design and management of the on-chip memory for capsulenet inference accelerators', ArXiv preprint ArXiv:190201151, 2019

[91] 'OpenCAPI consortium sc17 forum'. Available at: https://opencapi.org/wpcontent/uploads/2016/09/OpenCAPI-Overview.10.14.16.pdf, accessed September 2019

[92] Stuecheli, J., Blaner, B., Johns, C., *et al.*: 'CAPI: a coherent accelerator processor interface', *IBM J. Res. Dev.*, 2015, **59**, (1), pp. 7.1–7.7

[93] 'An introduction to the intel quickpath interconnect'. Available at: https://www.intel.in/content/dam/doc/white-paper/quick-path-interconnectintroduction-paper.pdf, accessed September 2019

[94] Abdelrahman, T.S.: 'Accelerating *K*-means clustering on a tightly-coupled processor FPGA heterogeneous system'. Int. Conf. Application-specific Systems, Architectures and Processors (ASAP), London, England, 2016, pp. 176–181

[95] Venkata, S.K., Ahn, I., Jeon, D., *et al.*: 'SD-VBS: the San Diego vision benchmark suite'. Int. Symp. Workload Characterization (IISWC), Austin, TX, USA, 2009, pp. 55–64

[96] Stratton, J.A., Rodrigues, C., Sung, I.J., *et al.*: 'Parboil: a revised benchmark suite for scientific and commercial throughput computing', *Center Reliable High-Performance Comput.*, 2012, **17**, (2), pp. 1–12

[97] Xu, S., Wang, Y., Han, Y., *et al.*: 'PIMCH: cooperative memory prefetching in processing-in-memory architecture'. Proc. Asia and South Pacific Design Automation Conf., Jeju Island, Republic of Korea, 2018, pp. 209–214

[98] Jeong, T., Choi, D., Han, S., *et al.*: 'A study of data layout in multi-channel processing-in-memory architecture'. Proc. Int. Conf. Software and Computer Applications, Kuatan, Malaysia, 2018, pp. 134–138

[99] Dai, G., Huang, T., Chi, Y., *et al.*: 'Graph: a processing-in-memory architecture for large-scale graph processing', *Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 2019, **38**, (4), pp. 640–653

[100] Mutlu, O., Ghose, S., Ausavarungnirun, R.: 'Recent advances in DRAM and flash memory architectures', ArXiv preprint ArXiv:180509127, 2018

[101] Yu, S., Chen, P.Y.: 'Emerging memory technologies: recent trends and prospects', *Solid-State Circuit Mag.*, 2016, **8**, (2), pp. 43–56