A Review of Near-Memory Computing Architectures: Opportunities and Challenges

Gagandeep Singh*, Lorenzo Chelini*, Stefano Corda*, Ahsan Javed Awan†, Sander Stuijk*, Roel Jordans*, Henk Corporaal*, Albert-Jan Boonstra‡

*Department of Electrical Engineering, Eindhoven University of Technology, Netherlands
{g.singh, l.chelini, s.corda, s.stuijk, r.jordans, h.corporaal}@tue.nl
†Department of Computing, Imperial College London, UK
ashan.awan@imperial.ac.uk
‡R&D Department, Netherlands Institute for Radio Astronomy, ASTRON, Netherlands
boonstra@astron.nl

***Abstract*—The conventional approach of moving stored data to the CPU for computation has become a major performance bottleneck for emerging scale-out data-intensive applications due to their limited data reuse. At the same time, the advancement in integration technologies have made the decade-old concept of coupling compute units close to the memory (called Near-Memory Computing) more viable. Processing right at the home of data can completely diminish the data movement problem of data-intensive applications.**

This paper focuses on analyzing and organizing the extensive body of literature on near-memory computing across various dimensions: starting from the memory level where this paradigm is applied, to the granularity of the application that could be executed on the near-memory units. We highlight the challenges as well as the critical need of evaluation methodologies that can be employed in designing these special architectures. Using a case study, we present our methodology and also identify topics for future research to unlock the full potential of near-memory computing.

***Index Terms*—near-memory computing, data centric computing, modeling of computer architecture, application characterization, survey**

I. Introduction

Over the years memory technology has not been able to keep up with advancements in processor technology in terms of latency and energy consumption which is infamously termed as the *memory wall* [1]. Earlier, system architects tried to bridge this gap by introducing memory hierarchies which mitigated some of the disadvantages of off-chip DRAMs. But the limited number of pins on the memory package is not able to meet today's bandwidth demands of multicore processors. Furthermore, with the demise of Dennard scaling [2], slowing of Moore's law, and dark silicon computer performance has reached a plateau [3].

At the same time, we are witnessing an enormous amount of data being generated across multiple areas like radio astronomy, material science, chemistry, health sciences etc [4]. In radio astronomy for example, the first phase of the Square Kilometre Array (SKA) aims at processing over 100 terabytes of raw data samples per second, yielding of the order of 300 petabytes of SKA data products annually [5]. The SKA currently is in the design phase with anticipated construction in South Africa and Australia in the first half of the coming decade. These radio-astronomy applications usually exhibit massive data parallelism and low operational intensity with limited locality. On traditional systems, these applications cause frequent data movement between the memory subsystem and the processor which has a severe impact on performance and energy efficiency. Therefore, a lot of current research is being focused on coming up with innovative manufacturing technologies and architectures to overcome these problems.

Today's memory hierarchy usually consists of multiple levels of cache, a main memory, and a storage. The traditional approach is of moving data all the way up to caches from the storage and then processing it. In contrast, *near-memory computing* (NMC) *aims at processing close to where the data resides*. This data-centric approach couples compute units close to the data and seeks to minimize the expensive data movements. Conceptually, this principle can be applied to any level of memory subsystem and with several technology options, including processing on buffer-on-board (BoB), edge-bonding small processor dies on DRAM chips etc [6]. Notably, three-dimensional stacking is touted as the true enabler of processing close to the memory. It allows the stacking of logic die and memory together by means of through-silicon via's (TSVs) which helps in reducing memory access latency, power consumption, and provides much higher bandwidth [7]. Micron's Hybrid Memory Cube (HMC) [8], High bandwidth Memory (HBM) [9] from AMD and Hynix, and Samsung's Wide I/O [10] are the memory industries competing 3D memory products.

Figure 1 depicts the system evolution based on the information referenced by a program during execution which is referred to as a *working set* [11]. Prior systems were based on a *CPU-centric* approach where data is moved to the core for processing (Fig.1 (a)-(c)), whereas now with near-memory processing (Fig.1 (d)) the processing cores are brought to the place where data resides. Computation in-memory (Fig.1 (e)) is an extreme step of completely reducing data movement by using novel devices (eg. memristors, phase change memory (PCM)) as these memories have inherent compute capability

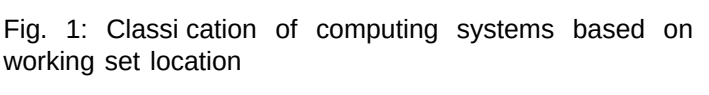

Fig. 1: Classification of computing systems based on working set location

	Property	Abbrev	Description
Memory	Hierarchy	MM	Main Memory
		SCM	Storage Class Memory
		HM	Heterogeneous Memory
	Type	C3D	Commercial 3D memory
		DIMM	Dual in-line memory module
		PCM	Phase Change Memory
		DRAM	Dynamic Random-Access Memory
		SSD	Flash SSD Memory
		LRDIMM	Load-Reduce DIMM
	Integration	US	Conventional Unstacked
		S	Stacked using 2.5D or 3D
Processing	NMC/Host Unit	CPU	Central Processing Unit
		GPU	Graphics Processing Unit
		FPGA	Field Programmable Gate Array
		CGRA	Coarse-Grained Reconfigurable Architecture
		ACC	Application Specific Accelerator
	Implementation	P	Programmable Unit
		F	Fixed function Unit
		R	Reconfigurable Unit
	Granularity	I	Instruction
		K	Kernel
		A	Application
	Host Unit		Type of Host Unit
Tool	Evaluation technique	A	Analytic
		S	Simulation
		P	Prototype/Hardware
Interoperability	Programming Interface	Y/N	Programming Interface support
	Cache Coherence	Y/N	Mechanism for Cache coherence
	Virtual Memory	Y/N	Virtual Memory Support
App. Domain	Workload		Target Application domain for the architecture

Table 1: Classification table, and legend for table 2

as well. The aim of this paper is to analyze and organize the extensive body of literature related to the novel area of near-memory computing. Figure 2 shows a high level view of our classification which is based on the level in the memory hierarchy and further split into the type of compute implementation (Programmable, Fixed-Function, or Reconfigurable). In our taxonomy we don't include disk-based systems as they can no longer offer timely response due to the high access latency and high failure rate of disks [12]. Nevertheless, there have been research efforts towards providing processing capabilities in the disk. However, it was not adopted widely by the industry due to of the marginal performance improvement that could not justify the associated cost [13], [14]. Instead, we include emerging non-volatile memories termed as storage-class memories (SCMs) [15] which are trying to fill the latency gap between DRAMs and disks. Building up on similar efforts [16], [17], we make the following contributions:

1) We analyze and organize the literature related to the landscape of near-memory computing under various dimensions;
2) We provide guidelines for design space exploration focusing on near-memory systems;
3) We present a case study to illustrate the potential of near-memory computing for memory intensive application with a comparison to the current CPU-centric approach;
4) We outline the direction for future research and highlight current challenges in the domain of near-memory computing.

The remainder of this article is structured as follows. Section II outlines the evaluation and classification scheme for NMC at main memory (Section III) and storage class memory (Section IV). Section V highlights the challenges with cache coherence, virtual memory, the lack of programming models and data mapping schemes for NMC. In Section VI we look into the tools and techniques used while performing the design space exploration for these systems. This section also illustrates the importance of application characterization for these systems. Section VII presents a high level analytic approach to model NMC systems with a comparison to a traditional CPU-centric approach. Section VIII highlights the lessons learned and future research directions.

II. Classification and Evaluation

This section introduces the classification and evaluation that is used in Section III and IV and is summarized in table 1 and table 2. For each architecture five main categories are evaluated and classified:

Memory - The decision of using what kind of memory is one of the most fundamental questions on which the near-memory architecture depends.

Processing - Type of processing unit implemented and the granularity of processing it performs plays a critical role during the design space exploration.

Tool - Any architecture's success depends heavily on the available tool support. As NMC is a fledgling field, there is lack of generic design methodologies.

Interoperability - One of the main challenges faced by NMC architectures is its inadequacy at providing support for a programming model, cache coherence, virtual memory, and efficient data mapping.

Application - Usually these architectures focus on big data applications/algorithms and this massive amount of data is produced across all the markets. Therefore, in our table we include the domain of the application.

III. Processing near Main Memory

Processing near main memory has been the most researched interpretation. It can be coupled with different processing units ranging from programmable units to fixed-functional units.

A. Programmable Unit

NDC (2014) Pugsley et al. [20] proposed a near-memory computing (NDC) architecture for MapReduce workloads, in which a central host processor with many energy efficient cores is connected to many daisy-chained 3D-stacked memory devices with simple cores in their logic layer; these cores perform Map operations. Reduce operations, however, are executed on the central host processor because it requires random access to data.

TOP-PIM (2014) Zhang et al. [21] proposed GPU-accelerated architecture. It consists of a main APU (GPU

NMC architecture		Memory			Processing				Tool	Interoperability			App. Domain
Architecture	Year	Hierarchy	Type	Integration	NMC unit	Implementation	Granularity	Host Unit	Evaluation technique	Programming Interface	Cache Coherence	Virtual Memory	Workload
SmartSSD [18]	2013	SCM	SSD	US	CPU	P	A	CPU	P	Y	Y	N	Database
WILLOW [19]	2014	SCM	SSD	US	CPU	P	K	CPU	P	Y	Y	-	Generic
NDC [20]	2014	MM	C3D	S	CPU	P	K	CPU	S	N	R	N	MapReduce
TOP-PIM [21]	2014	MM	C3D	S	GPU	P	K	CPU	S	N	Y	-	Graph and HPC
AMC [4]	2015	MM	C3D	S	CPU	P	K	CPU	S	Y	Y	Y	HPC
JAFAR [22]	2015	MM	DIMM	US	ACC	F	K	CPU	S	Y	-	Y	Database
TESSERACT [23]	2015	MM	C3D	S	CPU	F	A	CPU	S	Y	Y	N	Graph Processing
Gokhale [24]	2015	MM	C3D	S	ACC	F	K	CPU	S	Y	Y	Y	Generic
HRL [25]	2015	MM	C3D	S	CGRA+FPGA	R	A	CPU	S	N	Y	N	MapReduce
ProPRAM [26]	2015	SCM	PCM	US	CPU	P	I	-	S	Y	-	-	Data Analytics
BlueDBM [27]	2015	SCM	SSD	US	FPGA	R	K	-	P	Y	-	-	Data Analytics
NDA [28]	2015	MM	LRDIMM	S	CGRA	R	K	CPU	S	Y	Y	Y	MapReduce
PIM-enabled [29]	2015	MM	C3D	S	ACC	F	I	CPU	S	Y	Y	Y	Generic
IMPICA [30]	2016	MM	C3D	S	ACC	F	K	CPU	S	Y	Y	Y	Pointer chasing
TOM [31]	2016	MM	C3D	S	GPU	P	K	GPU	S	N	Y	Y	Generic
BISCUIT [32]	2016	SCM	SSD	US	ACC	F	K	CPU	P	Y	-	-	Database
CARIBOU [33]	2017	SCM	DRAM	US	FPGA	R	K	CPU	P	-	-	-	Database
Vermij [34]	2017	MM	C3D	S	ACC	F	A	CPU	S	Y	Y	Y	Sorting
SUMMARIZER [35]	2017	SCM	SSD	US	CPU	P	K	CPU	P	Y	-	-	Database
MONDRIAN [36]	2017	MM	C3D	S	CPU	P	K	CPU	A+S	Y	-	Y	Data Analytics

Table 2: Architectures classi cation and evaluation, refer to table 1 for a legend

+ CPU) interconnected with high-speed serial links with multiple 3D-memory modules. APU allows code portability and easier programmability. The kernels analyzed span across a huge domain from graph processing to uid and structural dynamics.

AMC (2015) Nair et al. [4] developed active memory cube (AMC). AMC is built upon the HMC adding several processing elements referred as lanes in the logic layer. The host processor and each AMC are interconnected over a bidirectional link following a daisy-chain topology. For each AMC there are 32 4-slices lanes. Each lane has is own register le, a load/store unit performing read and write operations to the memory contained in the same AMC, and a computational unit.The communications between AMCs should be coordinated by the host processor.

PIM-enabled (2015) Ahn et al. [29] created NMC architectures with the same ISA as the host. They tried to leverage the existing programming model so that the conventional architectures can exploit the PIM concept without changing their programming interface. They implemented it by adding a computation unit composed of computation logic and an SRAM operand buffer, both on host side and in the logic layer of the Hybrid Memory Cube (HMC).

TOM (2016) Hsieh et al. [31] proposed an NMC architecture consisting of main GPU interconnected via high-speed serial links with multiple 3D-stacked memories. Each memory device hosts multiple streaming multiprocessor (SMs) in the logic layer. In line with the philosophy of near-memory computing, blocks of code are of oaded to the SMs in order to reduce off-chip traf c and reduce power consumption due to data movement.

Fig. 2: Processing options in the memory hierarchy

MONDRIAN (2017) Drumond et al. [36] analyzed data analytics oeprators which are not a t for NMC. They made a case for optimizing traditional algorithms for NMC system for sequential memory access rather than cache locality.

JAFAR (2015) Xi et al. [22] proposed JAFAR, an NMC accelerator that allows to process ltering operations near the main memory. Thus only relevant data will be pushed up in caches, causing a signi cant reduction in data movement.

TESSERACT (2015) Ahn et al. [23] focus on graph processing applications. Their architecture consists of one host processor with his own memory and a HMC, with an out-of-order processor mapped to each vault. These cores can see only their own local data partition, but they can communicate to each other using a message passing method. The host

processor as well has access to the entire address space of the HMC. To exploit the high available memory bandwidth in the systems, they developed prefetching mechanisms.

IMPICA (2016) Hsieh et al. [30] accelerated the pointer-chasing operations which are ubiquitous in data structures. They proposed adding specialized units that decouple address generation from memory access in the logic layer of 3D stacked memory. These units traverse through the linked data structure in memory and return only the final node found to the CPU. They also proposed completely decoupling the page table of IMPICA from that of CPU to avoid virtual memory related issues.

B. Fixed-Function Unit

Vermij [1](2017) Vermij et al. [34] evaluated the performance of sorting algorithms in big data workloads running on a 2-socket IBM POWER8 machine. They proposed a heterogeneous system where algorithm phases with high temporal locality are executed on the CPU, while algorithm phases with poor temporal locality are executed on the NMC devices. The architecture proposed consists of a central host processor that relies on two-level memory controllers. A memory-technology agnostic controller located at the CPU side and a memory-specific controller tightly coupled with the memory. The NMC accelerators are placed in the memory-specific controllers and are assisted with an NMC manager. The NMC manager interfaces with the accelerator and the rest of the system and provides support for cache coherency, virtual memory management and communications with the host processor.

C. Re-configurable Unit

Gokhale[1] (2015) Gokhale et al. [24] proposed to place a data rearrangement engine (DRE) in the logic layer of the HMC to accelerate data accesses while still performing the computation on the main CPU. The authors targeted cache unfriendly applications with high memory latency due to irregular access patterns, e.g., sparse matrix multiplication. Each of the DRE engine consists of a scratchpad, a simple controller processor, and a data mover (aka DMA engine). In order to make use of the engines the authors developed an application program interface (API) with different operations. Each operation is issued by the main application running on the central host and served by a control program loaded by the OS on each DRE engine.

HRL (2015) Gao et al. [25] proposed a re-configurable logic architecture called Heterogeneous Re-configurable Logic (HRL) that consists of three main blocks: fine-grained configurable logic blocks (CLBs) for control unit, coarse-grained functional units (FUs) for basic arithmetic and logic operations, and output multiplexer blocks (OMBs) for branch support. Each memory module follows HMC like technology and houses multiple HRL devices in the logic layer. The central host is responsible for data partition and synchronization between NMC units.

NDA (2015) Farmahini et al. [28] proposed three different NMC architectures using coarse-grained reconfigurable arrays in commodity DRAM modules. This proposal requires minimal change to the DRAM architecture. However, programmer should identify which code will run close to memory. This leads to increased programmer effort for executing compute intensive code on the 3D stack logic layer. Also, it doesn't support direct communication between NMC stacks.

[1]Architecture has no name, first author's name is shown.

IV. Processing near Storage Class Memory

Flashes and other emerging nonvolatile memories such as phase-change memory (PCM) [37], spin-transfer torque RAM (STT-RAM) [38], memristors [39], etc., are termed as storage-class memories (SCMs) [15]. These memories are trying to fill the latency gap between DRAMs and disks. SCMs like NVRAM are even touted as a future replacement for DRAM [40]. Moving computation in SCM has some of the similar benefits to DRAM in terms of saving in bandwidth, power, latency, and energy but also because of the higher density it allows to work on much larger data-sets as compared to DRAM [41].

A. Programmable Unit

Smart SSD (2013) Kang et al. [18] proposed a model to harness the processing power of the SSD using an object-based communication protocol. They implemented the Smart SSD features in the firmware of a Samsung SSD and modified the Hadoop core and MapReduce framework to use tasklets as a map or a reduce function. To evaluate the prototype, they used a micro-benchmark and log analysis application on both a device and a host. They found that under the current SSD architecture, excessive memory accesses will make the task execution slower than in the host due to the high memory latency and low processing power.

WILLOW (2014) Seshadri et al. [19] proposed a prototype system called Willow, which has processing units called storage processor units (SPU). Each SPU runs a small operating system that manages and enforces security. On the host-side, the Willow driver creates and manages a set of objects that allow the OS and applications to communicate with SPUs. The programmable functionality is provided in the form of SSD Apps. Willow allows programmers to augment and extend the semantics of an SSD with application-specific features without compromising file system protection. The programming model for SSD Apps provides great flexibility, supports the concurrent execution of multiple SSD Apps in Willow, and supports the execution of trusted code in Willow.

ProPRAM (2015) Wang et al. [26] observed that NVM is often naturally incorporated with basic logic like data comparison write or flip-n-write module, and exploited the existing resources inside memory chips to accelerate the key non-compute intensive functions of emerging big data applications.

SUMMARIZER (2017) Koo et al. [35] designed APIs that can be used by the host application to offload a data-intensive task to the inherent ARM based cores inside a SSD processor. Using a fully functional SSD evaluation platform they performed design space exploration of the proposed approach. They evaluate static and dynamic approaches for dividing the work between the host and SSD processor.

B. Fixed-Function Unit

BISCUIT (2016) Gu et al. [32] presented Biscuit, a near-memory computing framework which allows programmers to write a data-intensive application to run on the host system and the storage system in a distributed manner. The SSD hardware incorporates a pattern matcher IP designed for NMC. When this hardware IP is applied, modified MySQL significantly improves TPC-H performance.

C. Reconfigurable Unit

BlueDBM (2015) Jun et al. [27] presented a flash-based platform, called BlueDBM, built of flash storage devices augmented with an application specific FPGA based in-storage processor. The data-sets are stored in the flash array and are read by the FPGA accelerators. Each accelerator implements an array of application-specific distance comparators, used in the high-dimensional nearest-neighbour search algorithms.

CARIBOU (2017) Zsolt et al. [33] explored offloading part of the computation in database engines directly to the storage and deploying FPGAs. They implemented a cuckoo hash table with a slab-based memory allocator to improve the handling of collisions and various values sizes in hardware-based key-value stores. Lookups and scans are performed on the same data to minimize data transferred over the network. They implemented a runtime parametrizable selection operator both for structured and unstructured data in an effort to reduce data movement further.

V. Challenges of Near-Memory Computing

One of the biggest challenges in near-memory computing is the lack of interoperability with caches and virtual memory of the host processor due to unconventional programming models. In this section we will explain the current issues with programming model, data mapping, virtual memory support and cache coherency in the context of NMC. Note there are many more design challenges, like design space exploration, reliable simulators which can deal with heterogeneous environments, etc. They are detailed in section VI.

Virtual Memory support

Support for virtual memory is achieved using paging [6] or direct segment [1]. In the former approach most NMC systems adopt a software-managed translation lookaside buffer (TLB) to provide a mapping between virtual and physical addresses [6], [42], [31], [43]. Azarkhish et al. in [44] observed that an OS managed TLB may not be the optimal solution and proposed an hardware implementation where a simple controller is responsible for fetching the rule in the page table. In [30], Hsieh et al. notice that for pointer-chasing applications, the accelerator works only on specific data structures that can be mapped onto a contiguous region of the virtual address space. As a consequence, the virtual-to-physical translation for the accelerator is limited to a smaller region. Other works such as [45], [46] decided for a simplified approach where part of the linear virtual address is mapped to physical memory using a direct segment rather than a page. This mechanism allows to remove TLB misses overhead and greatly simplifies the hardware [47].

Cache Coherence

Although in some works the cache coherence has not been treated [46], [48], we classify the approaches proposed in literature as restricted memory region and non-restricted region.

1) Restricted Memory Regions

Farahani et al. [28] divided the memory in two regions: one for the host processor and one for CGRAs, which is uncacheable. A similar approach has been used by Ahn et al. [23] for graph processing algorithms. Another strategy, proposed by Ahn et al. [29], provides a simple hardware-based solution in which the PIM operations are restricted to only one last level cache block, due to which they can monitor the cache block and request for invalidation or write-back if required.

2) Non-Restricted Memory Regions

Non-restricted memory regions can often lead to a significant amount of coherence traffic. Pattnaik et al. [49] proposed to maintain coherence between the main/host GPU and near-memory ones, flushing the L2 cache in the main GPU after kernel execution. With this approach, applications with highly irregular memory accesses lead to significant amount of overhead. Another way is to implement a look-up based approach as in the case of Hsieh et al. [31]. The NMC units record the cache line address that has been updated by the offloaded block and once when the offloaded block is processed they send this address back to the host system. Subsequently, the host system gets the latest data from memory by invalidating the reported cache lines.

Programming Model

A critical challenge in adoption of NMC is to support a heterogeneous processing environment of a host system and near-memory processing unit. It is not trivial to determine how and who identifies which part of an application should run in the near-memory processing units. Currently, this is left onto the programmer or the compiler [31] to mark sections of the code that must be processed near-memory compute units [17]. In another approach, Ahn et al. [29] use some special set of NMC instructions which invoke NMC logic units. However, this approach calls for sophisticated mechanism (e.g., taking care of the data locality) on the host side to avoid unnecessary exchange of information. Hence, there is still a lot of research required in coming up with an efficient approach to ease the programming burden.

Data mapping

The absence of effective data mapping mechanisms can severely hamper the benefits of processing close to memory. The data should be mapped in such a way that data that will be used by the near-memory processing units should be available in the vicinity. Hence, it is crucial to look into effective data mapping schemes (static and dynamic)

In TOM [31], they proposed a software/hardware cooperative method to predict which pages of the memory will be used by the offloaded code, and they tried to minimize the bandwidth consumption by placing those pages in the memory stack closest to the offloaded code. Yitbarek et al. [46] proposed a data-mapping scheme which they integrated in a

Metrics	Description
Reuse distance (temporal locality)	The number of distinct memory accesses since the last access to a given memory address
Data Stream Stride (spatial locality)	The difference in memory addresses between two consecutive memory accesses
Application bandwidth	The amount of I/O bandwidth utilized by a particular application
Instruction mix	Provides a general overview of the application
Parallelism	Inherent application parallelism such as instruction level parallelism, thread level parallelism, data level parallelism
Branch Entropy	Measures the randomness of branch behavior which is essential to provide speculation support

Table 3: Several metrics for micro-architecture independent analysis of workloads

Fig. 3: Design space exploration for system evaluation

platform that used some accelerators near 3D-stacked memory. Near-memory workloads often access data from contiguous addresses for this reason they re-mapped data at page granularity in order to have contiguous addresses in the same memory vaults.

VI. Design Space Exploration for NMC

As will be clear from the previous classification section, the design space of NMC is huge. In order to understand and evaluate this space, design space exploration (DSE) for NMC system is required, as shown in figure 3.

A. Application Characterization

More than ever, application characterization has taken a crucial role in systems design due to the increasing number of new data-intensive applications. It is used to extract information by using specific metrics, in order to decide which architecture could have the best performance and energy efficiency. In table 3, we have highlighted some of the useful metrics for near-memory architectures. Application characterization can be done in one of the following ways:

Microarchitecture-Dependent Workload Characterization

Traditionally workload characterization is done using hardware performance counters present on our current microprocessors. Awan et al. [50] use hardware performance counters to characterize the scale-out big data processing workloads into CPU-bound, memory-bound, and I/O-bound applications and propose programmable logic near DRAM and NVRAM. However, the use of hardware performance counters is limited by the impact of micro-architecture features like cache size, issue width etc [51]. To overcome this problem, recently there has been a strong push towards following a ISA independent characterization approach.

Microarchitecture-Independent Workload Characterization

Hoste et al. [52] proposed a Microarchitecture-Independent Workload Characterization (MICA) in which they used Pin [53] as basis. Also, they proposed new methodologies to analyze these characteristics, based on principal components analysis (PCA), which gives a reduction to data set's dimensionality and removes correlation from the data set, and also a genetic algorithm (GA), which reduces the data set's dimensionality, but with a retained dimension easily to understand.

Anghel et al. [54] presented a Platform-Independent Software Analysis (PISA) tool, based on the LLVM compiler. It takes a source code as input and generates a LLVM-IR bitstream which is instrumented with some function calls to a tool library that implements different workload analysis. Characteristics which PISA can extract are: Instruction Mix, Instruction-Level-Parallelism, Memory Access Patterns, Branch Entropy, Communication Patterns. Unlike other tools, PISA is also capable of analyzing multi-threaded programs.

In the same way Capalija et al. [55] used the LLVM framework for their tool. It's only for sequential code, but there are interesting metrics that are extracted such as: Data-Level-Parallelism (DLP), Thread-Level-Parallelism (TLP) and Task-Level Parallelism. This analysis is made through a Direct-Acyclic-Graph (DAG) that searches for the real data dependencies and at various granularity levels, e.g. instructions, basic blocks, functions.

Using the Intermediate Representation (IR) from the Just-In-Time (JIT) compiler Shao et al. [51] extracted number of opcodes, the value of branch entropy, the value of memory entropy, the unique number of static instructions, and the unique number of data addresses for sequential benchmarks. In

B. Performance Evaluation Techniques

Architects often make use of various evaluation techniques to navigate the design-space, avoiding the cost of chip fabrication. Based on the level of detail required, architects make use of analytic models, or more detailed functional or cycle accurate simulators, and even design prototypes. As the field of NMC does not have very mature tools and techniques, researchers often spend a lot of time building the appropriate evaluation environment. [56]

Analytic Modeling

Analytic models abstracts low level system details and provide quick performance estimates at the cost of accuracy. In

Simulator	Year	Category	NMC capabilities
Sinuca [60]	2015	Cycle-Accurate	Yes
HMC-SIM [61]	2016	Cycle-Accurate	Limited
CasHMC [62]	2016	Cycle-Accurate	No
SMC [44]	2016	Cycle-Accurate	Yes
CLAPPS [56]	2017	Cycle-Accurate	Yes

Table 4: Open source simulators

the early design stage, system architects are faced with large design choices which range from semiconductor physics, to circuit level, micro-architectural properties (eg., issue width, branch prediction), cooling concerns, and other system design constrains [57]. During the first stage of design-space exploration analytic models can provide quick estimate number. Mirzadeh et al. [58] studied a join workload, on multiple HMC like 3D-stacked DRAM devices connected together via SerDes links using a first-order analytic model. Zhang et al [59] designed an analytic model using machine learning techniques to estimate the final device performance.

Simulation Based Modeling

To achieve more accurate performance numbers, architects often resort to modeling the entire micro-architecture precisely. However, this approach can be quite slow compared to analytic techniques. Hyeokjun et al. [63] evaluated the potential of NMC for machine learning (ML) using a full- edged simulator of multi-channel SSD that can execute various ML algorithms on data stored on the SSD. Similarly, Jo et al. [64] developed the iSSD simulator based on the gem5 simulator [65].

Ranganathan et al. [66], [40] for their nano-stores architecture used a hybrid methodology which is a bottom-up approach in which they built a high level model which breaks down applications into various phases, based on compute, network, and I/O sub system activities. This takes input from the low level performance and power models regarding the performance of each phase. They used COTSon [67] for detailed micro-architectural simulation.

Fig. 4: Our 3D stacked near-memory system connected to a host system

VII. Case study

We built a high level analytic model which incorporates evaluation methodology as described in figure 3. To see the potential of computing close to the memory and when it is useful to follow a data-centric approach we make a comparison of a CPU-centric approach consisting of a multi-core system which has DDR3 as a main memory, and compared it to a data-centric approach which has HMC like 3D stacked memory instead of DDR3 and is connected to a host system. In the multi-core system each core has its private L1 cache and a shared L2 cache. Figure 4 shows our 3D stacked near-memory system connected with a host system through a serializer/deserializer (SerDes) I/O link. We embed near-memory compute units in the logic layer of 3D stack memory. The near-memory compute units are modeled as ARM Cortex-A9 units (table 5), which are placed in the logic layer of 3D stacked DRAM.

Description	Symbol	ARM
Cores	n_{cores}	4
Core clock frequency	f_{core}	3 GHz
L1 size	s_{l1}	32 KB
L2 size	s_{l2}	256 KB
DRAM size	s_{DRAM}	4 GB
L1 cache bandwidth	bw_{l1}	137 GB/s
L2 cache bandwidth	bw_{l2}	137 GB/s
DRAM bandwidth	bw_{DRAM}	17 GB/s
L1 cache hit latency	l_{l1}	1 cycles
L2 cache hit latency	l_{l2}	2 cycles
DRAM hit latency	l_{DRAM}	165 cycles
NVM hit latency	l_{ext}	800 cycles

Table 5: System parameters and example set representing ARM Cortex-A9 (ARM) processor

We consider a 4-GB HMC like design, with 8 DRAM layers. Each DRAM layer is split into 16 dual banked partitions with 4 vertical partitions forming a vault. Each vault has its own vault controller in the logic layer which is able to access the DRAM layers at a 10 GB/s bandwidth [4]. In the near-memory system we assume certain parts of computation, which are memory intensive, to be offloaded to the near-memory compute units and the other parts will be executed in the host system. The number of vaults inside the HMC and external SerDes links are kept as varying parameters which have an impact on internal and the external bandwidth of the 3D memory respectively.

Performance calculation: The model estimates the execution time as the ratio between the number of memory access required in order to move the block of data to and from the I/O system and the available I/O bandwidth of each memory subsystem. The execution time takes into account the time to process the instructions and latency due to memory access at different memory levels. The latency of different memory types have been mentioned in table 5.

Energy calculation: A power model for both CPU-centric and near-memory system were built which take into account dynamic as well as static energy. Dynamic energy is the product of the energy per memory access and number of memory accesses at each level of the memory hierarchy. It is assumed to scale linearly with utilization. Static energy is estimated as the product of time and static power for each component. It is scaled based on the number of cores and the cache size. For modeling the HMC we referred to [7] and [20]. We consider energy per bit of 3.7 pJ/b for accessing the DRAM layers and 1.5 pJ/b for the operations and data movements in the HMC logic layer. Also a static power of 0.96 W [20] was considered to model the additional components in the logic layer.

Design Space Exploration: We vary the cache miss rate, which encapsulates application characteristics, both at L1 and L2 level to see the impact it has on performance of the entire