INVITED PAPER

Evolution of Memory Architecture

Problems to be tackled efficiently and new applications have a strong say in defining how memory architectures must evolve. With large data as a defining theme, this paper discusses how processor and system architecture is likely to continue to change to move to a form where rapid retrievability will become a critical characteristic.

By Ravi Nair, *Fellow IEEE*

ABSTRACT | Computer memories continue to serve the role that they first served in the electronic discrete variable automatic computer (EDVAC) machine documented by John von Neumann, namely that of supplying instructions and operands for calculations in a timely manner. As technology has made possible significantly larger and faster machines with multiple processors, the relative distance in processor cycles of this memory has increased considerably. Microarchitectural techniques have evolved to share this memory across ever-larger systems of processors with deep cache hierarchies and have managed to hide this latency for many applications, but are proving to be expensive and energy-inefficient for newer types of problems working on massive amounts of data. New paradigms include scale-out systems distributed across hundreds and even thousands of nodes, in-memory databases that keep data in memory much longer than the duration of a single task, and near-data computation, where some of the computation is off-loaded to the location of the data to avoid wasting energy in the movement of data. This paper provides a historical perspective on the evolution of memory architecture, and suggests that the requirements of new problems and new applications are likely to fundamentally change processor and system architecture away from the currently established von Neumann model.

KEYWORDS | Approximate memories; disk; dynamic random access memory (DRAM); flash memory; main memory; memory hierarchy; near-data processing; non-von Neumann architectures; processing-in-memory; storage-class memory; von Neumann architecture

Manuscript received November 7, 2014; revised April 14, 2015; accepted May 6, 2015. Date of publication July 7, 2015; date of current version July 15, 2015.
The author is with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (e-mail: nair@us.ibm.com).

Digital Object Identifier: 10.1109/JPROC.2015.2435018

I. INTRODUCTION

The central processing unit (CPU) of a computer is often considered the heart of the computer, the part of the computer through which data are streamed and transformed in the course of a calculation. As such it is also given pride of place, with the arithmetic power of a system often being used as a metric of performance of the system. However, the computational capability of a system is dependent not just on the number of calculations that it can perform in a second, but also on the capability and efficiency of staging data to these calculations. These data include not only the input data and parameters needed to solve the problem but also auxiliary tables that need to be referenced, as well as data produced in the course of calculations that need to be coursed back through the computational units.

There is another important item needed in completing calculations, and that is the recipe precisely describing the calculations needed, their order, and their dependence on the results of previous calculations. This recipe for the successful completion of a task, or program, is specified as a sequence of steps called instructions, typically saved in some external medium and brought into the computer along with other items of input data when the program needs to be executed.

All this information needed by a computation is today brought into a central place called the memory of the computer, much as in the early electronic discrete variable automatic computer (EDVAC) [1]. Access to instructions and data must be fast, so that the expensive calculating units are well utilized during the course of the program. Indeed, in the early days of computing, access to this memory was as fast, if not faster, as the rate of performing the actual calculations. But as computers became faster and as the size of problems grew both in input size and in the complexity of instructions, access to memory became steadily slower than the rate of computations. This led to

proposals, such as data-flow architectures [2], where the recipe for the calculation was not a sequential set of steps and where the data were not centralized, but rather were distributed in parallel among the computational elements of the computer. Such proposals did not succeed commercially, as computer engineers found ways of predicting the instructions and data that the computational elements would need, and located them appropriately in time for them to be processed on the various units of the machine. Thus, today's computers supplement the main memory of the computer with a set of local registers and one or more levels of cache in a memory hierarchy, where the cache level closest to the calculating engine is small and fast, but contains only a subset of the contents of memory.

The resulting machines have become quite sophisticated—and complex—and the community is once again asking whether it is efficient to keep data in a central location and move it to processing units which keep getting further away. Cost and energy have not been serious concerns as long as Moore's law [3] kept giving us steadily increasing transistor densities and as long as Dennard scaling [4] kept power requirements low. With the end of Dennard scaling, the focus of concern is shifting to the energy expended in moving data back and forth from memory shared by multiple processors across large systems. Thus, there is a trend toward partitioning the memory across distinct compute nodes and performing multiple tasks in parallel, analogous to the old data-flow proposals. This paradigm appears to particularly suit the computation needs of workloads dealing with large volumes of unstructured data.

This paper will describe this new trend and examine specifically the evolution of the role of main memory. The organization of the paper is as follows. Section II presents a brief history to understand how the organization of the computer evolved to the form it is today. Section III describes the memory hierarchy developed by processor designers to combat the problem of increasing memory latency. Section IV discusses memory reliability, while Section V introduces storage-class memories which bridge the gap between DRAM and disks. This is then followed in Section VI by a description of the change in the volume and nature of data and the changing role of data in the modern world. Section VII describes near-data computation, the trend toward offloading computation to processing units nearer the data, while Section VIII describes how advances in 3-D technology are reviving interest in processing-in-memory. Section IX argues that dense memories in new technologies are likely to be exploited better when applications tolerate approximate solutions. Section X examines the possibilities of integrating memories and computing more tightly in a post-von Neumann world.

II. EVOLUTION OF MAIN MEMORY

In 1945, during the early days of computing, John von Neumann wrote a draft of the design of a computer called the EDVAC [1] which had a memory unit attached to an arithmetic unit and a control unit. The memory unit was in addition to the teletype, magnetic tape, punched card, or other storage unit that stored the problem code, parameters of the problem, and tables needed in the solution of the problem. Von Neumann stated that memory elements "that only circulate in the interior of the device, and never be recorded for human sensing" were needed in the machine for the following reasons:

1) to remember intermediate results during complex arithmetic operations like multiplication and division;
2) to store the instructions for a complex calculation;
3) to provide tables for functions in order to avoid repeated complex calculations;
4) to remember initial conditions and boundary conditions for solutions of partial differential equations;
5) to store results of one or more iterations in a successive approximation algorithm for use in later iterations.

Von Neumann argued that even though the nature of operations needed for the different tasks was different and even though the locations in the machine where they were needed were different, it was useful to have a single type of memory with interchangeable parts to accommodate the varying needs in different applications of the various functions mentioned above. When it was finally built in 1949, the EDVAC had a memory of 1024 44-b words in addition to its external tape unit. The memory was a sequential access mercury delay-line memory with an average delay of 192 μs and a throughput of one word every 48 μs from each of 128 delay lines. In comparison, additions were performed at the rate of one every 864 μs [5].

Such high-speed memories were expensive and finicky and hence unsuitable for commercial use. Commercial systems instead adopted a more affordable form of memory, the drum memory, where the drum was a metal cylinder coated with a magnetic recording material. The first IBM 650 machines [6] could be configured with up to 2000 ten-digit words of drum memory and with an average access time of 2.4 ms, limited by the 12 500-rpm rotational speed of the drum. It could perform 200 ten-digit additions per second. A drum memory with 4608 words was added to the EDVAC in 1954 to supplement its high-speed delay-line memory. Drum memories had a short run as main memory because of their rotational latency, but they were still in widespread use into the 1960s as secondary memory or storage.

The 1960s and 1970s saw the prolific use of the magnetic core memory which, unlike drum memories, had no moving parts and provided random access to any word in memory. Early core memories had a cycle time of roughly 6 μs, nearly 1000 times faster than drum memories [7]. Early core memory systems had only around 4000 words, but their capacities increased rapidly. The capacity of the CDC 6600 with a clock cycle of 100 ns in 1965 was around

128 000 words with an access time of 1 μs [8]. Besides its superior access time, a big advantage of magnetic core memory was its reliability.

Core memory was replaced by semiconductor memory in the 1970s. The memory industry was revolutionized by the invention of the dynamic random access memory (DRAM), specially the single-transistor DRAM cell [9]. Semiconductor memories provided several times better density than core memories, but had disadvantages. Unlike core memories which were nonvolatile, semiconductor memories consumed power to retain their state. DRAMs had even more disadvantages being particularly vulnerable to soft errors and needing to be refreshed periodically. Despite these disadvantages, DRAMs became the staple form of main memory as they rode the technology curve, increasing in density and decreasing in cost per bit every generation.

Early DRAMs, like the MK4006 launched around 1970, had a capacity of 1 kb [10] and an access time of 400 ns. By the end of the decade, the capacity of a single DRAM chip went up to 64 kb in the MK4164 with an access time of 100 ns. Technology scaling allowed processor clock frequencies to increase much more rapidly during the same period, and the resulting increased latency in cycles to memory was ameliorated by the introduction of static RAM (SRAM) caches closer to the processor. Temporal locality and spatial locality in programs were exploited using a cache that had only a fraction of the capacity of the DRAM main memory. Caches were transparent to the programmer, with hardware determining which parts of memory should be placed at any given time in the cache. As technology enabled more and more transistors to be placed on a die, many implementations chose to separate the instruction cache from the data cache, while the system still maintained the unified memory model.

Meanwhile disks emerged in the early 1970s replacing drums as the preferred nonvolatile backup for DRAM main memory [11]. Compared to the fixed head-per-track of drums, disks had movable heads, and hence the transition from drums to disks involved tradeoffs in cost, performance, and complexity. However, disk technology kept improving at impressive rates bringing the price per bit down the same curve as that of DRAM. The importance of nonvolatile disks grew with the widespread adoption of memory virtualization [12], which allowed programs to address a virtual memory larger than the size of main memory of the computer. Virtual memory was made transparent to the user, with the operating system, assisted by hardware, deciding which part of the virtual memory should be in main memory and which part in the larger disk storage.

Thus, by the 1970s, the various parts of the memory hierarchy as seen in systems today were already in place.

III. THE MEMORY HIERARCHY

Surprising as it may seem, the basic principles that guided the design of the EDVAC are still employed in today's machines. These basic principles constitute what is termed the von Neumann architecture. In this architecture, a set of instructions is fetched from memory and executed in a processing unit. Often in the course of executing these instructions, values need to be fetched from memory using load instructions and stored back into memory using store instructions. For longer term access both instructions and data are stored in file systems on disk.

The most significant change that has been made in instruction-set architectures since the early days of computing is that today's reduced instruction-set computers (RISCs) perform calculations on registers which are initialized with values explicitly loaded from memory, or which hold results of computations performed by the computational unit. Results that need to be stored back to memory are performed using explicit stores. This is in contrast to the early computers where many of the arithmetic and logical instructions had operands that addressed contents in memory. This two-step process was necessitated by the increased latency to memory compared to the latency of computation, as described in Section I. Memory latency increases as memory size increases; the latency of memory access in a modern server runs into the hundreds of cycles [13]. The utilization of the computational units can be greatly increased if operands can be fetched from memory into registers earlier than when they are needed in the computation—registers provide a location close to the computation for staging the operands. In addition, registers are useful when there is temporal locality in a program; reuse of a value from a register is far less costly than bringing the value back from DRAM.

Main memories today are composed of DRAM chips that are typically packaged in modules called dual inline memory modules (DIMMs), with several DIMMs making up the total main memory of the system. The capacity of a DRAM chip has steadily increased from the 1 kb in the first DRAM chip mentioned earlier to 8 Gb today, an eight million fold increase in 40 years. Such a density allows the total main memory of a system to be as large as 16 TB in a 16-socket system [14]. Commercial database applications that run on such systems work on data that need to be persistent and of very large size. The low cost per bit of disks and their inherent nonvolatile nature makes disks the natural place to maintain these databases, but the latency of access to data on disks is limited by their mechanical characteristics. A large memory in such systems helps significantly in improving the performance of queries on these databases, as it provides a staging area for the data immediately relevant to the computation, analogous to the role of registers mentioned above.

The part of the memory hierarchy that has undergone the most change over the years is the layer between the register file close to the processor and the main memory. The principal role of this layer is to reduce the effective latency of a load from memory. As mentioned earlier, caches are used to anticipate the parts of memory that

	Power5	Power6	Power7	Power7+	Power8
Technology	130 nm	65 nm	45 nm	32 nm	22 nm
Cores per chip	2	2	8	8	12
L1 cache per core	32 + 32 KB	64 + 64 KB	32 + 32 KB	32 + 32 KB	64 + 32 KB
L2 cache per core	⇓	⇓	256 KB	256 KB	512 KB
Last-level cache per chip	1.9 MB	8 MB	32 MB	80 MB	96 MB
Off-chip cache per socket	36 MB	32 MB	None	None	128 MB
Max. Memory per socket	64 GB	192 GB	256 GB	512 GB	1 TB
Bandwidth to Memory	15 GB/s	30 GB/s	100 GB/s	100 GB/s	230 GB/s

Fig. 1. ***Capacities of cache levels and memory in IBM's Power series of machines.***

would be needed by a program. As memory sizes have increased and as the absolute latency of memory access has increased over the years, an increasing number of cache layers has been introduced between the register file and memory. Today it is not unusual to find at least three levels of cache in processors. The first level of cache is typically about 16–64 kB in size with access latencies that are 2×–4× the cycle time of the processor. This is often backed up by a second level of cache that may be between 128 kB and 2 MB in size and with latencies that are 7×–15× the cycle time of the processor, and may be shared among multiple processors on the same die. In both POWER7 [13] and POWER8 [14], the last-level of cache on chip is actually an implementation of DRAM technology on the processor chip called eDRAM [15], that provides about 4× the density of SRAM caches and burns significantly less power. The latency of access to this layer is in the range of 100 processor cycles, while its capacity is in the range of 50–100 MB. In order to further close the gap between this layer and the terabytes of main memory that may be needed in a system, large POWER8 configurations also provide the capability for yet another layer of cache, an eDRAM L4 cache, embedded in the memory controllers external to the processor with up to 128-MB capacity. Fig. 1 shows the trend in capacities of caches and memory of IBM's Power.[1]

Such deep memory hierarchies are not without their costs. As mentioned earlier, the programmer knows only about registers and memory. It is the hardware that hides the details of caching data in the appropriate layers. Besides the directories that keep track of which parts of memory are in which layer of cache, there is a significant amount of logic that goes into the organization of these caches (e.g., associativity) and in the replacement algorithms to determine which blocks should be replaced to make room for a new block. In addition, when multiple users are running simultaneously on the system, it is also important to keep track of the specific cache blocks and memory pages that a user is allowed to access. All these checks add to the complexity of the processor and to the overhead incurred by software, particularly in the operating system. The hierarchy is effective when the locality in an application conforms to the structure of the hierarchy, but when it does not, it hurts both performance and energy efficiency.

Finally, when multiple processors share the same memory, it is important to ensure that accesses to memory are performed in a manner that is consistent with the expectation of the programmer. These expectations are embodied in specifications called the memory coherence and memory consistency specifications. The more stringent the specification the easier it is for the programmer to reason about programs, but the more complex the hardware needed to enforce the specification. Unfortunately, with few exceptions, the programming model for general purpose machines is largely unchanged from the programming model of the early von Neumann machines, and interfaces for parallel programs are designed to ease the burden for programmers attuned to thinking in sequential steps. The continued pervasive use of these interfaces has led to features in hardware specifically to optimize the performance for such interfaces, and thereby led to added complexity in the hardware.

The memory hierarchy as described above evolved from the needs of servers and workstations. As technology advanced, this hierarchy trickled down into low-end systems, including tablets, smartphones, and embedded systems. But these latter systems also emphasize power consumption of their memory subsystems considerably more than latency. They save power by operating at lower voltages, refreshing less often, and providing deeper power-down modes. With energy consumption gaining importance on higher end systems, many of the latter power-saving techniques are likely to find their way to higher end systems also [16].

IV. RELIABILITY CONSIDERATIONS

DRAMs have always been vulnerable to soft errors, either due to contaminants in packaging or because of cosmic rays [17]. As a result, there has always been additional logic either incorporated into DRAMs or in DRAM memory systems to make them reliable. In the beginning, this additional logic was just parity, with, for example, one additional DRAM chip in a DIMM to store the parity of the bytes contained in the other eight chips. The parity bit is written when the memory is written and compared with the regenerated parity when memory is read. Detection of errors through parity is not sufficient for large commercial memory systems; the frequency of errors increases as the size of memory, and software handling of detected errors gets cumbersome. The availability of these systems is

[1]Trademark, service mark, or registered trademark of International Business Machines Corporation in the United States, other countries, or both.

improved by incorporating some degree of error correction in addition to error detection. Error correction however involves logic that is more complex, compared with single-bit parity schemes, and adds to the cost of memory systems.

Many high-end systems implement a single-error correcting, double-error detecting (SECDED) code by using a full byte of code for every 64 b of memory. Lower end systems may not use additional bits for error detection, or may use simply a parity bit. However, with greater use of embedded devices in critical applications, reliability expectations for lower end systems have become higher; even these devices today tend to use error-correcting codes like SECDED in their memory systems.

The cost of memory systems is also increased by the redundancy incorporated to improve the manufacturing yield of DRAM chips. Rather than ensure that all bits in a DRAM are functioning according to specification, it is less expensive to provide spare rows, columns, and decoders or even redundant chips in a system, and configure the addressing logic to bypass failed structures [18]. The amount and type of redundancy needed is determined by the types of failure that need to be tolerated and by statistical data on the contribution to the yield of these failure types.

Going beyond memory chips themselves, errors in communicating the contents between a memory module and the processor may be detected using cyclic redundancy or checksum codes on the entire content block, typically a cache line. A general survey of the various reliability features incorporated in contemporary DRAMs may be found in [11].

V. STORAGE-CLASS MEMORIES

The difference in cost per bit of DRAMs and disks and the difference in latencies of access between the two have led to a perpetual quest for a type of memory, preferably with no moving parts, which bridges the gap between the two, or even eliminates it. The technology currently attempting to fill that gap is flash memory [19]. The density efficiency of flash memory is between that of disks and DRAM, and its access latency, particularly READ latency, is much better than that of disks. Moreover, its cost per bit is dropping rapidly thanks to its prolific use in mobile devices and even laptop computers, where its nonvolatile nature allows flash memory to replace disk file systems [20], and where the absence of moving parts makes it more rugged and reliable.

Whereas its READ time is not too slow compared to that of DRAM, the WRITE time of flash memory is far slower than that of DRAM. The process of changing the contents of a flash cell is more complex than writing a DRAM cell and consumes a significant amount of energy. The stresses introduced during WRITES also tend to limit the number of times that a flash cell can be written into—typically between 10 000 and 100 000 times. This is not a significant limitation for lower end uses of flash memory, especially in mobile devices with their limited platform lifetimes. Schemes have been proposed [21], [22] to get around the WRITE endurance problem, and flashes are beginning to be used in hybrid configurations along with low cost disks to stage file systems and database storage [23]. In such use, flashes and other emerging semiconductor nonvolatile memories are termed storage-class memories (SCMs).

While flash memories are finding a place in the storage subsystems of modern machines, a higher density option to replace DRAMs in main memory has not yet emerged. Flash memories are inadequate in such a role both because of their limited WRITE capability as well as because of their high WRITE latency. There have been proposals for using flash, not directly as main memory, but in a hybrid configuration with DRAM used as a staging buffer for data stored in flash [24], [25]. Other emerging technologies, such as phase-change memory (PCM) [26] and spin-transfer torque RAM (STT–RAM) [27] have been mooted as possible alternatives for hybrid configurations.

VI. NATURE OF DATA

Many of the microarchitectural techniques developed in the 1980s and 1990s were in response to the needs of transaction-processing systems which continued to grow large and handled large volumes of data. Databases, especially relational databases, formed the important workload of that period. Databases were important in all industries and hence had to satisfy stringent requirements from the point of view of reliability and correctness. These requirements were encapsulated in the atomicity, consistency, isolation, and durability (ACID) properties [28] that most databases strived to satisfy.

From an architecture point of view, the dominant multiprocessor architecture that databases favored has been the shared memory multiprocessing (SMP) [29] architecture. As database requirements kept increasing, machines with this architecture were scaled up, i.e., built with ever larger number of processors sharing a common memory. In order to satisfy a query or perform analytics, data was fetched from the database residing on disks and moved through the various levels of the memory hierarchy to a processor or a set of processors that performed the needed operations, such as filtering, sorting, and merging. The results returned through the same layers in the memory hierarchy back to disk or to an interactive user. Thus, a significant amount of data movement is involved, often of data not needed by the computation, and this movement is alleviated only a little by making the memory and caches large enough to contain the immediately useful data.

With the proliferation of new types of transactions such as those involved in Internet commerce and social networking, such scale-up SMP systems became impractical. Design complexity and cost considerations forced the move toward more scale-out distributed systems with memory partitioned across nodes connected in a cluster, and not shared among them. In such systems, the task of ensuring

consistency in light of possibly conflicting parallel accesses to data is left to the programmer or to the system software.

Since the 1990s, distributed systems have been playing a role in large supercomputers where cost and scalability are important requirements. Today, distributed clusters form the dominant type of supercomputer, accounting for 85% of the top 500 supercomputers in the world [30]. Supercomputers are designed for scientific simulation experiments which generally model physical phenomena. In these applications, the computation itself is cleanly partitionable into local regions that often correspond closely to partitions of the physical world being simulated. Correspondingly, the interconnection between nodes in the system is designed to favor communication from a node to a fixed set of neighboring nodes, an n-dimensional torus interconnect [31] being an example. The solution technique commonly employed in these systems is a bulk-synchronous model, where all nodes iteratively alternate between a parallel computing phase and a parallel communication phase. The programmer continues to use von Neumann programming models to program each node and uses special interfaces such as message passing interface (MPI) [32] to program the exchange of information between nodes during the communication phase. The key goal for such systems is maximum utilization of the system. Toward this end, efforts are made by the programmer and the system designer to carefully balance the compute load on all nodes, minimize activity from the operating system during problem execution, and compute almost exclusively on the contents in a node's memory, minimizing disk accesses. These efforts have been largely successful and the modest communication overhead between nodes has enabled a partitioned von Neumann model to serve the purposes of this community.

The commercial database world has also encountered the problem of scale, with high-capacity, highly available transactional machines becoming very expensive. As in the case of scientific supercomputers, cost considerations are leading to the partitioning of databases in distributed nodes on a network. But unlike most supercomputer applications, redundant copies of data often need to be maintained across nodes both to improve resilience and to reduce latency of interactive transactional operations. The challenge then is to maintain a consistent view of the database at all times across the entire system without impacting the availability of the system and hence its usefulness. This was encapsulated in Brewer's consistency, availability, and partition tolerance (CAP) theorem [33] which stated that when a record is replicated across partitions, it is not possible to make consistent changes to the record while making the record immediately available across the system, if communication between the partitions is not reliable.

An implication of the CAP theorem is that in a distributed system, consistency requirements may need to be relaxed in order to gain faster responses in a more scalable manner [34]. Relaxing consistency requirements, while not an option for many traditional databases, such as financial databases, is often acceptable for databases associated with new types of transactions. In a database that contains reviews of items on a shopping website, for example, there is little impact if a shopper does not see the latest review of a product. By allowing the database to continue working with stale data while it gets updated with new values, the latency to responses to queries can be reduced, often considerably. Such databases are said to follow the basically available, soft-state, eventually consistent (BASE) principles [35], as opposed to the ACID principles cited earlier. Relaxation of consistency allows the exploitation of less expensive, more scalable systems, and is especially useful for READ-mostly databases, where WRITES and UPDATES are not common. In such cases, a node serving a portion of the database could have its working set in memory, and thus ensure that a significantly large fraction of the transactions at that node are satisfied with minimum latency. Search engines, for example, keep an index of web keywords in memory and satisfy search queries based on this index, only occasionally updating the index to include new documents or changed documents on the web.

The lower cost of scale-out systems, the enormous reduction in latency of query processing achieved by keeping a useful subset of data in memory, and the predominance of READ-type operations in new workload have led to new implementations of in-memory databases, such as SAP HANA [36]. In-memory databases are particularly useful in business analytics, the analysis of transaction data to drive business planning. Most such analytics work on snapshots of the state of a large database. When the size of the system memory is large enough to locate the snapshots in memory rather than on disk, the speed of analytics can be improved considerably and can help businesses react faster to conditions in the marketplace.

This trend toward moving the locus of data from disks to DRAM, with disks relegated to a backup role, has been the idea behind proposals such as RAMCloud [37]. RAMCloud postulates that when the DRAM of a distributed network of servers is the place where all data are stored rather than a farm of disks, the resulting decrease in access latency can translate to a 100–1000$\times$ gain in the throughput of database operations. There is undoubtedly a cost tradeoff. Disks are considerably cheaper in cost per bit compared to DRAM memory. Fig. 2, adapted from [38], shows the three-year total cost of ownership (TCO) for a database in disk, in flash memory, and in main memory. The figure shows that when query rates are high, in-memory databases have the best TCO even for moderately large databases. This is because, at the boundary, say between flash and DRAM, the cost of flash is limited by the queries per second needed whereas the cost of DRAM is limited by the capacity needed. Thus, for a point within the lower right green region, more flash has to be bought than is really needed, increasing the TCO compared to that of a DRAM providing similar bandwidth.

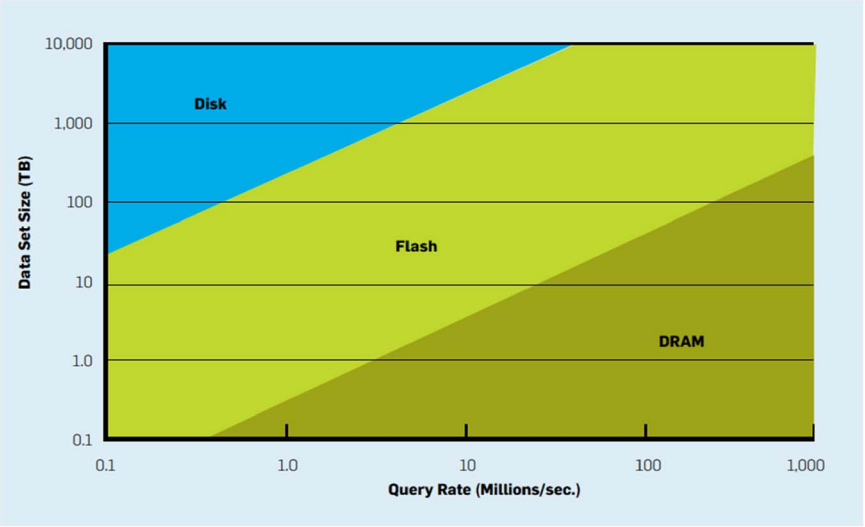

Fig. 2. *Solution space for lowest three-year TCO as a function of data set size and query rate (from [34] and [35]).*

The use of DRAM main memory to assist in query processing is especially important in what is commonly referred to as "not-only" SQL (NoSQL) databases [39]. Examples of NoSQL databases are key-value stores such as Redis [40], and key-value caches such as Memcached [41]. In the case of Redis, the entire key-value table is kept in memory and a disk access is performed only to log changes or to save checkpoint state for reliability. In the case of Memcached, the value of a key previously accessed from disk is cached in main memory. As long as there is reuse of this cached key, and as long as the value of the key remains unchanged, no accesses are required to the larger database on disk. It is not uncommon in social media for queries to be exactly the same for a window of time, or within a domain. Memcached can be useful in creating a hashed key from even complex queries and caching the results of such queries in a key-value table located in the memory of a local node.

These examples demonstrate that there is a distinct trend toward using large amounts of memory in systems. New programming paradigms and new distributed memory platforms are emerging with a view to providing new types of service for a world that is both producing and consuming ever larger bodies of data.

VII. NEAR-DATA PROCESSING

The principle of locality of computation has long been the basis of design decisions in the computing realm. Algorithms are generally designed with a view to placing needed data close to where they are used, both in a spatial sense and in a temporal sense. Hardware implementations are optimized for the case when algorithms exhibit such locality. As the amount of data used by an algorithm increases, and as accesses to data follow less regular patterns, it becomes difficult to satisfy locality requirements both at the algorithm level and at the hardware level. The prediction logic needed to place data at different levels of a deep memory hierarchy becomes too complex and too power-hungry, and is often ineffective. The question then arises whether, instead of having the data move through all the levels of the memory hierarchy from disk to processor and back to disk, it would not be more efficient in time and energy to move the computation to where the data are located.

Commercial systems have recently started exploiting this concept. An example is the IBM Netezza Database Appliance [42]. As illustrated in Fig. 3, this system performs many types of computations close to disk, and hence transfers processed information rather than raw data from the disk to the processor. The types of computations performed on the disk are varied, ranging from simple filtering operations or data compression operations to sophisticated query processing. The benefits of performing such operations closer to the disk are manifold: energy is saved by not having to move data through the memory hierarchy, memory bandwidth requirements are reduced, the latency of query responses is reduced, and the host processor is freed to perform other tasks.

Rows of tables are distributed across all disks automatically by the system. When the processor receives a complex query, the parts of queries that can be distributed and done in parallel near the disks are sent to each disk. The Netezza system includes field-programmable gate arrays (FPGAs) in the disk controller, as shown in Fig. 4, to allow the "computation near data" to be customized for each application. The partial results of such localized processing are sent back to the main processor to be combined with other similar results from other disks, to perform additional operations such as aggregation, join, or sort operations needed to complete the query processing.

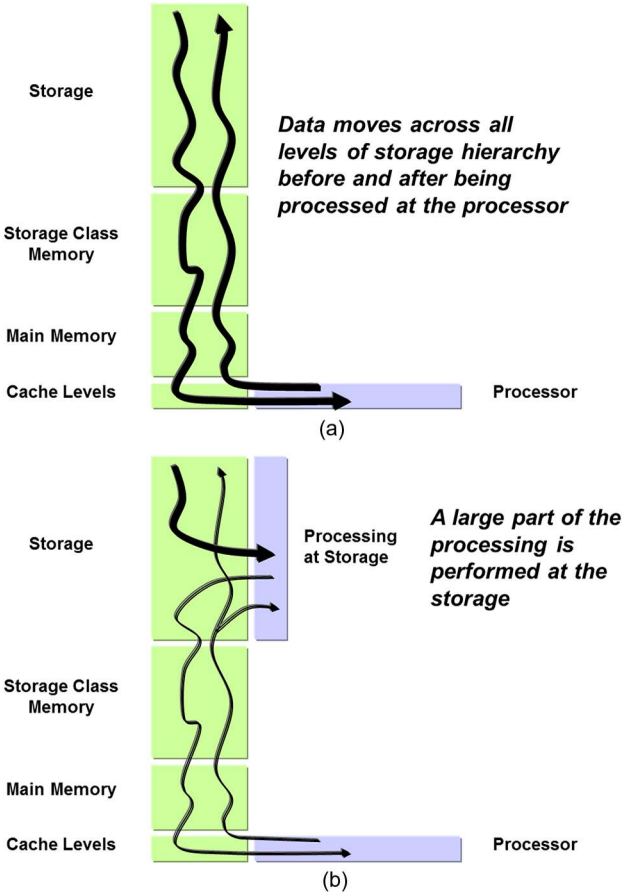

Fig. 3. *Moving computation to data. (a) Traditional processing. (b) The Netezza paradigm.*

Such techniques can be applied also to processing near solid-state drives (SSDs) like the flash storage mentioned earlier. The lower latency to SSDs compared to disks makes it possible to perform even more complex functions closer to the SSD. Moving computation away from processors closer to SSDs can provide savings in bandwidth, power, and energy, can be more trusted, and incurs much lower latency [44]. Multipart, atomic WRITE operations done at the SSD can reduce latency for transactional updates by up to 65% and almost eliminate the bandwidth overheads incurred by logging schemes typical in transactional databases [45]. Implementing portions of key-value stores in an SSD has been shown to improve throughput between 7.5 and 9.6 times [46]. Embedded information about file system data structures can be processed close to the SSD, so that the CPU can be freed from performing simple metadata updates.

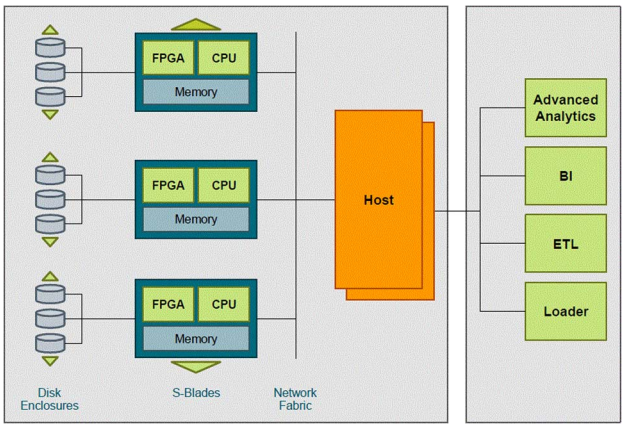

Fig. 4. *Netezza TwinFin appliance architecture (from [43]).*

The industry is currently at an inflection point where on the one hand, the nature of technology available to continue scaling the capabilities of computing systems is different from that of the past, and, on the other hand, the nature of problems that need to be solved have characteristics different from those of the past. Near-data processing is a paradigm that shows significant promise in helping to transport us into the new computing landscape [47].

VIII. PROCESSING-IN-MEMORY

While the speed of a basic addition operation and the clock frequency of processors improved steadily in the 1980s and 1990s, the improvement in performance of the system as a whole was limited by the speed of communication between the processor and other components in the system. In particular, the latency of access to memory and the bandwidth to memory did not improve commensurately. This was referred to as the memory wall [48]. One of the solutions proposed to combat the memory-wall problem in the 1990s was processing-in-memory (PIM). Early PIM architectures, such as Terasys [49], Execube [50], DIVA [51], and VIRAM [52], either proposed the addition of specialized logic to DRAM chips so that some processing could be done directly in memory or assumed that all memory needed by the processor could be placed on the same chip as the processor. The principal idea was to take advantage of the bandwidth available by collocating memory and processing on the same chip rather than be constrained by the limited bandwidth through the pin interfaces of separate chips.

Many of these proposals showed promising performance and energy efficiency, but they did not make an impact, for two main reasons. The first was that DRAMs were built using a DRAM-technology process that was optimized for providing high DRAM density at low cost, and was different from the logic-technology process focused on providing fast transistors for processor chips. Incorporating processor logic in DRAM technology resulted in processors that were considerably slower and less dense compared with the same design in logic technology. With technology scaling rapidly improving the number and speed of processors, deeper SRAM cache hierarchies mitigated the bandwidth and latency disadvantages of off-chip DRAMs.

The second reason was that the popular SMP paradigm, mentioned earlier, allowed large centralized pools of memory to be accessed uniformly and coherently by any single processor in the system, and for such pools to be flexibly allocated among multiple processors; this was not possible with PIMs where the amount of memory visible to a processor on a single die was quite limited.

Instead of moving computation all the way to memory, one could imagine moving the computation to layers in the system between the processor and memory. One such location is the memory controller, traditionally a separate chip implemented in logic technology that performs functions in support of the DRAM memory chips, such as refreshing the DRAM cells, steering the address bits to avoid faulty cells on the chip, performing error detection and error correction, and performing self-test. There have also been proposals to integrate other types of functions in the memory controller [53]. Simple atomic operations are also performed at the memory controller in more advanced designs. High-end processor systems often also add a level of cache in the memory controller [54].

With the maturing of 3-D stacking technology, processing in or near memory is an area that is about to see significant progress. Three-dimensional stacking addresses almost all the deficiencies that have been associated with PIM in the past. By exploiting a third dimension, stacking has the potential to provide a compact footprint for a large amount of DRAM. Three-dimensional stacking also allows dies of different technologies to be connected with through silicon vias (TSVs) [55], which can be placed considerably closer than input/output (I/O) pins in a package. Thus, 3-D technology promises to provide substantially increased bandwidth between the DRAM and logic dies located on the same stack. Micron's hybrid memory cube (HMC) [56] is one such memory platform that dramatically increases the capacity of memory using 3-D stacking of several (4–8) layers of DRAM dies, and connects this stack to an additional base logic layer using TSVs (Fig. 5). The base layer uses complementary metal–oxide–semiconductor (CMOS) technology optimized for logic, and contains the memory controller, built-in self-test (BIST) logic, and interfaces to a host processor or other HMC stacks.

The memory dies in an HMC stack are partitioned into blocks, and all blocks in the same physical location on all dies are grouped together into an entity called a vault. A corresponding vault controller located in the logic layer individually manages each vault. All vault controllers are interconnected to one another and, through link controllers, to external I/O links via an internal network.

The logic layer in an HMC structure provides a convenient point where data transformations may be performed on the contents of the memory stack. Such transformations could filter data before they are provided to a host processor for further processing, thereby reducing bandwidth requirements and latency at the interface between the host processor and memory. In other cases, such transformations can remove the need for the host processor to READ or WRITE any part of the data. In both cases, the reduction of data movement between chips results both in improved performance and in power savings. Several proposals have been made for personalization of the logic layer in an HMC such as near-data computing (NDC) [57], throughput-oriented programmable processing in memory (TOP–PIM) [58], and the active memory cube (AMC) [59].

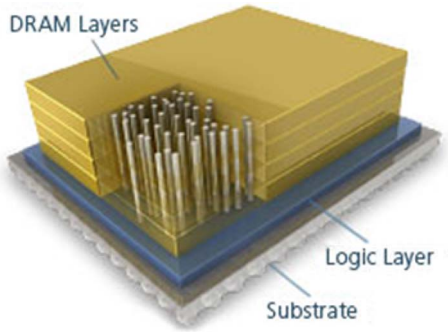

Fig. 5. *Structure of Micron's hybrid memory cube (from [56]).*

The AMC is a proposed personalization of the logic layer in an HMC targeted for energy-efficient implementation of future exascale supercomputers. As mentioned before, the vault controllers, interconnect network, and link controllers are located on a logic layer separate from the DRAM dies. On this layer, the AMC adds several processing elements, referred to as lanes, connected to an enhanced interconnect network, as shown in Fig. 6. By using the same interconnect network and the same transaction protocols between a lane or a link and the vault controllers, a lane can address memory locations within its AMC just as an external processor can. The link controller can send and receive READ and WRITE commands using special memory-mapped input/output (MMIO) addresses to and from any lane. This allows an external host processor to communicate directly with a lane, and for a lane to indicate some event to the host without involving the vault controllers.

By retaining the memory interface of the HMC, an AMC can appear to a host processor as ordinary memory accessible through standard memory READ and WRITE commands, while providing added capability of independently transforming the contents of memory. Special commands to memory have been provided on systems in the past, for example, in atomic READ–MODIFY–WRITE operations; the AMC extends the concept significantly further, allowing commands to specify arbitrary programs to perform the transformations. As described in [59], complete kernel programs performing a mix of instructions, including scalar and vector operations on the contents of an AMC, and branches, can be loaded into the instruction buffers of each lane of the AMC and executed on demand by the host through memory-mapped commands.

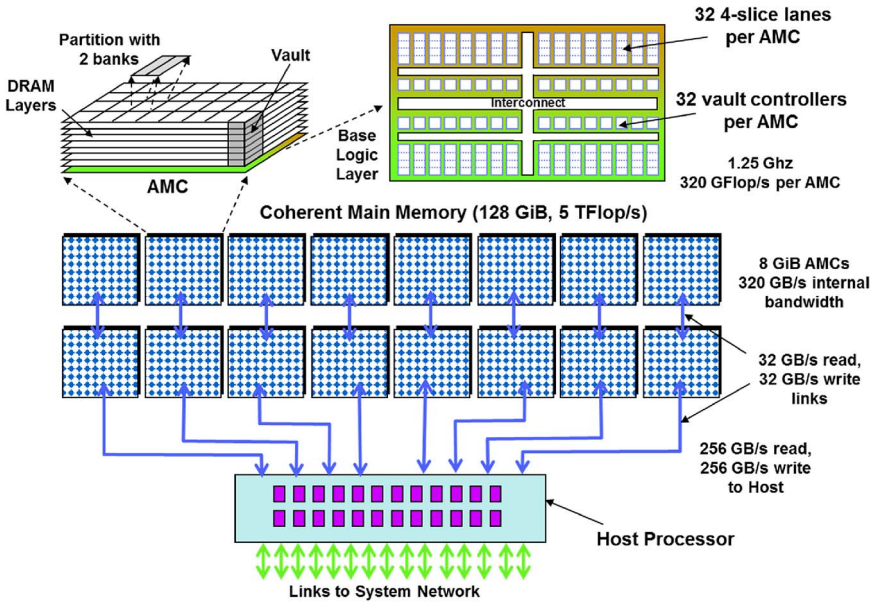

Fig. 6. *A 14-nm proposal for the AMC (from [59]).*

Die-stacking technologies are maturing to enable the integration of compute and data in a way that was not possible in previous merged logic-with-DRAM solutions. Besides the increased bandwidth between compute and memory, such integration offers the potential for significant savings in the energy of data movement. With the renewed focus on energy efficiency there is reason to believe that PIM could finally find widespread adoption.

IX. APPROXIMATE MEMORIES

It may no longer be possible for hardware that guarantees the high reliability traditionally enjoyed by the computing community to be built in future with reasonable cost. The massive number of components in future systems and the low voltages that will be needed to operate them efficiently collude to lower the inherent mean-time between failures (MTBFs) of future large systems. The operating system, runtime system, or even the application itself will need to step in to fill the impending reliability gap. Checkpointing schemes have been used in large supercomputers to allow an application to roll back to some known state when a fault is detected. Such schemes will become more common in other systems. Such schemes are also most effective when the application programmer provides hints about the tolerance to failures in various data structures of the program or in various phases of the program. An interesting example of this is the work described in [60], where compiler technology is used to transform an application to make it self-checkpointing and self-restartable.

Checkpointing is effective when there is a way to detect that a fault has occurred. However, it is very hard to anticipate all the ways in which faults can occur, and hence very hard to design detection capability for all types of faults. Once a failure that is not anticipated by the assumed fault model occurs, there is a chance that the computation will complete with corrupted results. While not all applications can tolerate corrupted results, it is worthwhile asking whether there are applications, or regions of applications, that can tolerate some degree of corruption in their results.

There is reason to be optimistic. During the early 2000s, there was a tremendous growth in data produced by humans, computers, and sensors. Along with this, there was a tremendous commercial interest in processing these data to glean actionable information. These data were different from those traditionally associated with transactional databases. Moore's law [3] and the resulting low cost of digital hardware led to the transition of all forms of media data, image, video, audio, sensory data, into a digital form rather than their erstwhile analog form [61]. In the early days, the new data represented digitally converted versions of existing analog data, but, as depicted in Fig. 7, the converted data are dwarfed today by the massive amount of new data being produced directly in digital form. The characteristics of this type of data and the type of analysis performed on such data are vastly different from those for transactional data. Transactional data, also called structured data, tend to be organized and searchable through simple search engines. On the other hand, unstructured data like email, audio files, presentations, videos, images, and instant messages are more difficult to perform an analysis on.

In the meantime, the types of problems that are being solved by computers is changing rapidly. The nature of computation needed on new data does not always require

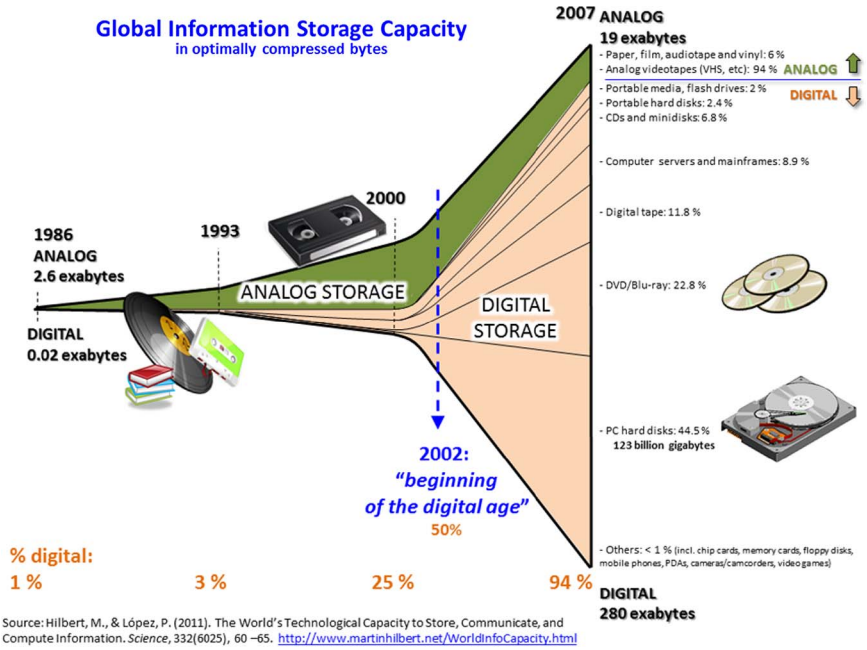

Fig. 7. ***Historical trend of growth of information storage capacity (from [62]).***

the preciseness that is implemented on today's computers. With the data having unknown provenance and unknown precision, and with errors possibly introduced into the data during transmission, it is more reasonable to relax the preciseness of the computation on such data, if it could provide faster answers or consume less energy. This is the idea behind approximate computing, a burgeoning research topic.

Approximate computing has implications on the memory system also. In one proposal, Sampson *et al.* [63] consider multilevel memories, such as multilevel phase-change memories [64], where the density is enhanced by storing more than one bit in a single memory cell. Multiple WRITEs may be needed to ensure that the correct value is stored in the cell and hence a WRITE latency cost and an energy cost is incurred in a multilevel WRITE. Performance can be improved by not incurring the latency of multiple WRITEs, but at a cost in the degradation of the final quality of the results. The user is allowed to specify that certain data structures in the program can be placed in not-so-reliable memory. The system partitions main memory into two regions, one with high reliability and another with high capacity but lower reliability, and allows the user to place each data structure in the appropriate target location. On a selected suite of programs, Sampson *et al.* were able to demonstrate an average performance gain of 1.7× with the results degrading in quality by less than 10%.

X. NON-VON NEUMANN ARCHITECTURES

The physical separation of memory and compute as seen in today's computers is a legacy of early machines and is a principal characteristic of the so-called von Neumann architecture. Over the years, this physical separation was also accompanied by a large difference between the latency of simple arithmetic computations and the latency of access to memory. Several changes have been made under the covers in the microarchitecture and hardware implementation to close this latency gap without changing the basic execution model. Mainstream systems have not strayed significantly from the original von Neumann architecture even though there have been several examples of specialized non-von Neumann systems for specific applications that demonstrate better performance, power efficiency, or area efficiency.

The changed nature of data and the changed nature of computation over the data provides an opportunity to bring new computational paradigms into the mainstream. There is a proliferation of proposals for application-specific accelerators attached to a general purpose host. These proposals typically consist of closed hardware solutions for a single application or a domain of applications, aiming to eliminate computational inefficiencies inherent in standard general purpose processors. Such accelerators can hide the detailed computing paradigm implemented within them by presenting just a software library interface to a program running on the host. Most of them, like GPUs [65] and Cell [66], tend to be simply application domain-specific von-Neumann architectures, still maintaining the separation of memory and compute.

The idea of closely coupling computation with memory is inherent in architectures like systolic arrays and cellular automata. These architectures have had a long history, but are getting renewed attention [67], because of their suitability to

problems involving today's new data and because of the emergence of new technologies to implement them.

Content-addressable memories (CAMs) and ternary content-addressable memories (TCAMs) present simple examples of an architecture that implements a close combination of memory and computation. In a CAM, comparison logic is included with each memory cell that makes it possible to select a chunk of memory, not by its location as in standard memory, but by its contents. However, the additional logic decreases the density of memory considerably compared to standard DRAM and adds to its power consumption. TCAMs [68] differ from CAMs in that, instead of a 0 or 1 specification for each bit required to be matched, TCAMs allow three states for each digit, 0, 1, and x, where x indicates that a match occurs irrespective of whether that digit is 0 or 1 ("don't care" bit). A common use for CAMs and TCAMs is in routing networks, where the address field of a network packet is used to rapidly determine how a packet should be routed to its destination.

The concept of content addressability and the idea of combining memory with computation was taken one step further in the connection machine of thinking machines [69] which incorporated a single-bit processing element with each 4 kB of memory. The machine consisted of 65 536 such elements interconnected in a hypercube configuration. This machine was able to exploit the data parallelism inherent in many scientific applications. In molecular dynamics, for example, one processing element is assigned to an individual atom, and force values from neighboring atoms are used to compute the net force on the atom in a cell and thereby its instantaneous trajectory. All calculations in all processors are performed in parallel, thus ensuring a significant speedup of the force calculation phase. The connection machine was an implementation of a concept developed by Danny Hillis in his Ph.D. dissertation [70] motivated by his desire for a computer that worked like the human brain. There is limited understanding of how the brain works, but the common belief is that, unlike von Neumann machines, computation and memory are much more closely integrated in the brain.

Several proposals have been made for computer structures that have functional elements similar to the functional elements of the brain. The most popular of these is the artificial neural network model [71]. The artificial neural network consists of layers of computational elements called neurons programmed to perform tasks such as image recognition. Neurons of one layer are connected to neurons of adjacent layers through an interconnection network. Each neuron gets a set of binary inputs from a previous layer and consists of a computational element implementing a nonlinear function whose result is transmitted to neurons in the next layer. To make the neural network general, a set of weights is associated with the function computed in the computational element. These weights form the "memory" spread across the computational elements in the network. The memory in an artificial neural network is local to a neuron, not shared between neurons. The amount of memory associated with each computational element increases with the sophistication of the task performed.

The parameters for the firing of neurons and their connectivity to other neurons is learned by exposing the model to a set of inputs with known outputs. This is very different from the rule-based programming of conventional general purpose computers which are programmed using the programmer's knowledge of how the task should be completed. A neural network is not capable of performing all the functions that a general purpose computer can. Unlike general-purpose computers, they are data driven; they produce actionable results based on previous learning in reaction to presented data.

The artificial neural network is a central element in the architecture used by Esmailzadeh *et al.* [72] to demonstrate the significant advantages in cost, power, and latency through approximate computing. The programming model proposed in the paper allows a region of a program to be deemed approximable by the programmer. The compiler produces a version of the approximable code targeted for a separate accelerator that performs the computation faster and with lower energy, but with results that may be different from the results produced by the host. On a variety of programs that tolerate approximation the authors show impressive performance gains and power reduction for a tolerable loss of quality by using an artificial neural network specially trained to the approximable code region, instead of running the region on a general purpose host processor.

There continue to be new models inspired by the brain, particularly for analyzing large bodies of streaming data. One such model is the hierarchical temporal memory [73] model, which is inspired by the neocortex region of the human brain in its functionality, and also to some extent in its structure. The HTM model is the engine behind Numenta's system for analyzing large bodies of data to detect anomalies [74].

Attention is also shifting toward the construction of machines, or machine elements, that have brain-like functionality. The SyNAPSE program [75] has developed a chip [76] that incorporates one million programmable neurons, 256 million programmable synapses, and 46 billion synaptic operations per second per watt. The 5.4 billion transistor CMOS chip implements an on-chip 2-D mesh network of 4096 digital, distributed neurosynaptic cores, each core integrating memory, computation, and communication, and operating in an event-driven, parallel, and fault-tolerant fashion. The combination of such a data-driven architecture with a hybrid asynchronous–synchronous design methodology and a 28-nm technology that provides low-leakage transistors together contribute to the chip's very low power density of 20 mW/cm^2 and very low average power consumption of 70 mW.

There is still a lot of work that needs to be done in making non-von Neumann systems as easy to program and

use as the current generation of general purpose machines. The early uses of such machines will likely be in special purpose applications and as appliances attached to more familiar computers. But there are signs that the demands of the data-driven world combined with the looming end of Moore's law will accelerate the quest for such new technologies and new architectures.

XI. CONCLUDING REMARKS

Memories have played an important role in computers since the early days of computing. Throughout, the principal use of memories has been to help complete a calculation, whether by storing the input needed for simulation or for transaction processing, holding the intermediate results produced during the course of the calculation, or saving the results of a calculation before they are archived or presented to a consumer.

Advances in lithography and circuit design have allowed memory densities to increase in an affordable manner. Supplemented with ingenuity in processor design, this increasing density has allowed larger and more complex problems to be solved efficiently without giving up on the von Neumann stored-program concept and the separation of computation and memory.

This free ride is ending. It is getting increasingly difficult to build and operate nanometer-scale device structures in a reliable manner, threatening the end of CMOS scaling. The workhorse of memory, the DRAM, is already demonstrating this difficulty with scaling. Newer devices like flash, PCM, and STT–RAM hold promise, but currently do not have all the attributes needed to replace DRAMs. Thus, the quest will continue for a technology that provides a memory that is nonvolatile, has density better than that of DRAM, and has access times no worse than that of DRAM.

Meanwhile, over the years, computing has become ubiquitous. The needs of computing can no longer be gauged by the requirements of a single computer or even a single system. Rather, computing today involves processing of data in a large connected world. The very nature of problems solved by computers has changed. Paradigms for using and programming computers have also changed to keep up with these new requirements, and so also has the role of memory.

This change is most visible at the application level, where traditional disk-based relational databases are giving way to new forms of in-memory databases, in-memory tables, and in-memory computing. The role of memory is changing from being a place to store information for a transient calculation to a place where rapidly retrievable information is stored over long periods of time. Today, this change is visible only at the highest levels of the system stack, but as these paradigms take hold, the change is likely to trickle down through the other layers and even into the processor and memory hardware.

Another change that is occurring is the realization that there are inefficiencies in moving data across the memory hierarchy, and across nodes in large distributed systems. This is engendering techniques to move computation to the data wherever it is located, whenever it is efficient to do so. This will also affect the design of memories—memories will no longer be passive repositories; they will be called upon to serve data in either raw or processed forms not only from compute elements local to their node, but from compute elements located anywhere on a large distributed system.

Finally, a change is occurring in the nature of tasks expected from computers. Unlike the simulation tasks that dominated the early computers and the transactional processing tasks that dominated the more recent servers, a significant amount of computation in the future will involve tasks that process multiple kinds of data, sometimes of unknown quality, to provide actionable information not only to individuals, but to institutions, corporations, and even governments. Many of these tasks will have low latency and cost requirements that can be met only by compromising somewhat on the precision and accuracy expected from traditional simulation and transaction processing tasks. New paradigms in approximate computing will emerge, and with it, new organizations for memories that bring computation and memory closer together at finer granularities.

The industry is at the threshold of a new post-von Neumann era. New technologies and new paradigms will emerge to solve new types of problems as well as to solve large-scale versions of old problems in a cost- and energy-efficient manner. Memory is clearly going to play an even more important role in this new era than it has in the past. ■

Acknowledgment

The author would like to thank D. Prener, H. Hunter, J. Moreno, J. Kahle, S. Iyer, and the anonymous reviewers for valuable comments and feedback on the manuscript.

REFERENCES

[1] J. Von Neumann, "First draft of a report on the EDVAC," *IEEE Ann. History Comput.*, vol. 15, no. 4, pp. 27–75, 1993.

[2] J. B. Dennis and D. P. Misunas, "A preliminary architecture for a basic data-flow processor," *ACM SIGARCH Comput. Architect. News*, vol. 3, no. 4, pp. 126–132, Jan. 1975.

[3] R. R. Schaller, "Moore's law: Past, present and future," *IEEE Spectrum*, vol. 34, no. 6, pp. 52–59, Jun. 1997.

[4] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted MOSFET's with very small physical dimensions," *IEEE J. Solid-State Circuits*, vol. SSC-9, no. 5, pp. 256–268, Oct. 1974.

[5] Electronic computers within the Ordnance Corps, chapter III—EDVAC. [Online]. Available: http://ftp.arl.mil/~mike/comphist/61ordnance/chap3.html

[6] IBM Archives, Magnetic drum data processing machine announcement, press release. [Online]. Available: http://www-03.ibm.com/ibm/history/exhibits/650/650_pr1.html

[7] Wikipedia, Magnetic-core memory. [Online]. Available: http://en.wikipedia.org/wiki/Magnetic-core_memory

[8] The CDC 6600 arrives at CERN. [Online]. Available: http://timeline.web.cern.ch/the-cdc-6600-arrives-at-cern

[9] R. H. Dennard, "Field-effect transistor memory," U.S. Patent 3 387 286, 1968.

[10] Mostek, Circuits and systems product guide, 1980. [Online]. Available: http://bitsavers.

trailing-edge.com/pdf/mostek/_dataBooks/1980_Mostek_Circuits_and_Systems_Product_Guide.pdf

[11] B. Jacob, S. Ng, and D. Wang, *Memory System: Cache, DRAM, Disk*. San Francisco, CA, USA: Morgan-Kaufmann, 2007.

[12] P. J. Denning, "Virtual memory," *ACM Comput. Surv.*, vol. 2, no. 3, pp. 153–189, 1970.

[13] IBM Corporation, Under the hood: Of POWER7 processor caches. [Online]. Available: http://www-03.ibm.com/systems/resources/systems_power_software_i_perfmgmt_underthehood.pdf

[14] IBM Corporation, IBM power system E880 data sheet. [Online]. Available: http://www.ibm.com/common/ssi/cgi-bin/ssialias?subtype=SP&infotype=PM&appname=STGE_PO_PO_USEN&htmlfid=PO-D03105USEN

[15] J. Barth *et al.*, "A 500 MHz random cycle, 1.5 ns-latency, SOI embedded DRAM macro featuring a three transistor micro sense amplifier," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2007, pp. 486–617.

[16] S. Mittal, "A survey of architectural techniques for DRAM power management," *Int. J. High Performance Syst. Architect.*, vol. 4, no. 2, pp. 110–119, 2012.

[17] T. J. O'Gorman, "The effect of cosmic rays on the soft error rate of a DRAM at ground level," *IEEE Trans. Electron Devices*, vol. 41, no. 4, pp. 553–557, Apr. 1994.

[18] S. E. Schuster, "Multiple word/bit line redundancy for semiconductor memories," *IEEE J. Solid-State Circuits*, vol. 13, no. 5, pp. 698–703, Oct. 1978.

[19] S. L. Min and E. H. Nam, "Current trends in flash memory technology," in *Proc. Asia South Pacific Design Autom.*, Jan. 2006, pp. 332–333.

[20] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings," in *Proc. 14th Int. Conf. Architect. Support Programm. Lang. Oper. Syst.*, 2009, pp. 229–240.

[21] Micron, "Wear leveling in micron NAND memory," Tech. Note TN-29-61. [Online]. Available: http://www.micron.com/~/media/Documents/Products/Technical%20Note/NAND%20Flash/tn2961_wear_leveling_in_nand.pdf

[22] M. Murugan and D. H. C. Du, "Rejuvenator: A static wear leveling algorithm for NAND flash memory with minimized overhead," in *Proc. IEEE 27th Symp. Mass Storage Syst. Technol.*, 2011, DOI: 10.1109/MSST.2011.5937225.

[23] T. Bisson and S. A. Brandt, "Reducing hybrid disk write latency with flash-backed I/O requests," in *Proc. 15th IEEE Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst.*, 2007, DOI: 10.1109/MASCOTS.2007.57.

[24] K. Sudan, A. Badam, and D. Nellans, "NAND-flash: Fast storage or slow memory?" presented at the Non-Volatile Memory Workshop, San Diego, CA, USA, Mar. 2012.

[25] J. C. Mogul, E. Argollo, M. Shah, and P. Faraboschi, "Operating system support for NVM+DRAM hybrid main memory," in *Proc. 12th Conf. Hot Topics Oper. Syst.*, May 2009, pp. 14–18.

[26] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *Proc. 36th Annu. Int. Symp. Comput. Architect.*, Jun. 2009, pp. 24–33.

[27] M. Hosomi *et al.*, "A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-RAM," in *IEEE Int. Electron Devices Meeting Tech. Dig.*, Dec. 2005, pp. 459–462.

[28] T. Haerder and A. Reuter, "Principles of transaction-oriented database recovery," *ACM Comput. Surv.*, vol. 15, no. 4, pp. 287–317, 1983.

[29] D. E. Culler, A. Gupta, and J. P. Singh, *Parallel Computer Architecture: A Hardware/Software Approach*. San Francisco, CA, USA: Morgan Kaufmann, 1997.

[30] Top 500 Supercomputer Sites. [Online]. Available: http://www.top500.org/

[31] The IBM Blue Gene Team, "The blue gene/Q compute chip," Hot Chips 23, 2011. [Online]. Available: http://www.hotchips.org/wp-content/uploads/hc_archives/hc23/HC23.18.1-manycore/HC23.18.121.BlueGene-IBM_BQC_HC23_20110818.pdf

[32] A. Skjellum, E. Lusk, and W. Gropp, *Using MPI: Portable Parallel Programming with the Message Passing Interface*. Cambridge, MA, USA: MIT Press, 1999.

[33] E. A. Brewer, "Towards robust distributed systems," presented at the Symp. Principles Distrib. Comput., Portland, OR, USA, Jul. 2000, Invited Talk.

[34] E. A. Brewer, "CAP twelve years later: How the 'rules' have changed," *Computer*, vol. 45, no. 2, pp. 22–29, 2012.

[35] D. Pritchett, "BASE: An ACID alternative," *ACM Queue*, vol. 6, no. 3, pp. 48–55, May/Jun. 2008.

[36] F. Färber *et al.*, "The SAP HANA database—An architecture overview," *IEEE Data Eng. Bull.*, vol. 35, no. 1, pp. 28–33, 2012.

[37] J. Ousterhout *et al.*, "The case for RAMClouds: Scalable high-performance storage entirely in DRAM," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 4, pp. 92–105, Jan. 2010.

[38] D. G. Andersen *et al.*, "FAWN: A fast array of wimpy nodes," in *Proc. ACM SIGOPS 22nd Symp. Oper. Syst. Principles*, Oct. 2009, DOI: 10.1145/1629575.1629577.

[39] J. Pokorny, "NoSQL databases: A step to database scalability in web environment," in *Proc. 13th Int. Conf. Inf. Integr. Web-Based Appl. Services*, 2011, pp. 278–283.

[40] S. Sanfilippo and P. Noordhuis, *Redis*. [Online]. Available: http://redis.io/

[41] B. Fitzpatrick, "Distributed caching with memcached," *Linux J.*, vol. 2004, no. 124, pp. 72–76, 2004.

[42] IBM Netezza, "IBM Netezza data warehouse appliances." [Online]. Available: http://www-01.ibm.com/software/data/netezza/

[43] P. Francisco, "The Netezza data appliance architecture: A platform for high performance data warehousing and analytics," in *IBM Redbooks*, 2011.

[44] S. Swanson, "Near-data computation: It's not (just) about performance," presented at the First Workshop Near Data Process, Davis, CA, USA, Dec. 8, 2013. [Online]. Available: http://www.cs.utah.edu/wondp/WoNDP-steve.pdf

[45] J. Coburn *et al.*, "From ARIES to MARS: Transaction support for next-generation solid-state drives," in *Proc. 24th ACM Symp. Oper. Syst. Principles*, 2013, pp. 197–212.

[46] A. De *et al.*, "Minerva: Accelerating data analysis in next-generation SSDs," in *Proc. IEEE 21st Int. Symp. Field-Programm. Custom Comput. Mach.*, 2013, pp. 9–16.

[47] R. Balasubramonian *et al.*, "Near-data processing: Insights from a MICRO-46 workshop," *IEEE Micro*, vol. 34, no. 4, pp. 36–42, 2014.

[48] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *ACM SIGARCH Comput. Architect. News*, vol. 23, no. 1, pp. 20–24, 1995.

[49] M. Gokhale, B. Holmes, and K. Iobst, "Processing in memory: The Terasys massively parallel PIM array," *Computer*, vol. 28, no. 4, pp. 23–31, 1995.

[50] P. M. Kogge, "EXECUBE—A new architecture for scaleable MPPs," in *Proc. Int. Conf. Parallel Process.*, Aug. 1994, vol. 1, pp. 77–84.

[51] J. Draper *et al.*, "The architecture of the DIVA processing-in-memory chip," in *Proc. 16th Int. Conf. Supercomput.*, Jun. 2002, pp. 14–25.

[52] C. E. Kozyrakis *et al.*, "Scalable processors in the billion-transistor era: IRAM," *Computer*, vol. 30, no. 9, pp. 75–78, 1997.

[53] J. Carter *et al.*, "Impulse: Building a smarter memory controller," in *Proc. 5th Int. Symp. High-Performance Comput. Architect.*, Jan. 1999, pp. 70–79.

[54] J. Friedrich *et al.*, "The POWER8 processor: Designed for big data, analytics, cloud environments," in *Proc. IEEE Int. Conf. IC Design Technol.*, May 2014, DOI: 10.1109/ICICDT.2014.6838618.

[55] M. Motoyoshi, "Through-silicon via (TSV)," *Proc. IEEE*, vol. 97, no. 1, pp. 43–48, Jan. 2009.

[56] J. T. Pawlowski, *Hybrid memory cube (HMC): Breakthrough DRAM performance with a fundamentally re-architected DRAM subsystem*, Hot Chips 23, 2011. [Online]. Available: http://www.hotchips.org/wp-content/uploads/hc_archives/hc23/HC23.18.3-memory-FPGA/HC23.18.320-HybridCube-Pawlowski-Micron.pdf

[57] S. Pugsley *et al.*, "ND: Analyzing the impact of 3D-stacked memory+logic devices on map reduce workloads," in *Proc. Int. Symp. Performance Anal. Syst. Softw.*, 2014, pp. 190–200.

[58] D. Zhang *et al.*, "TOP-PIM: Throughput-oriented programmable processing in memory," in *Proc. 23rd Int. Symp. High-Performance Parallel Distrib. Comput.*, 2014, pp. 85–98.

[59] R. Nair *et al.*, "Active memory cube: A processing-in-memory architecture for exascale systems," *IBM J. Res. Develop.*, vol. 59, no. 2/3, pp. 17:1–17:14, 2015.

[60] M. Schulz *et al.*, "Implementation and evaluation of a scalable application-level checkpoint-recovery scheme for MPI programs," in *Proc. 2004 ACM/IEEE Conf. Supercomput.*, Nov. 2004, pp. 38–51.

[61] M. Hilbert and P. López, "The world's technological capacity to store, communicate, compute information," *Science*, vol. 332, no. 6025, pp. 60–65, 2011.

[62] Wikipedia, Information society. [Online]. Available: http://en.wikipedia.org/wiki/Information_society

[63] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, "Approximate storage in solid-state memories," in *Proc. 46th Annu. IEEE/ACM Int. Symp. Microarchitect.*, Dec. 2013, pp. 25–36.

[64] F. Bedeschi *et al.*, "A bipolar-selected phase change memory featuring multi-level cell storage," *IEEE J. Solid-State Circuits*, vol. 44, no. 1, pp. 217–227, Jan. 2009.

[65] J. D. Owens *et al.*, "GPU computing," *Proc. IEEE*, vol. 96, no. 5, pp. 879–889, May 2008.

[66] J. A. Kahle *et al.*, "Introduction to the Cell multiprocessor," *IBM J. Res. Develop.*, vol. 49, no. 4/5, pp. 589–604, Jul. 2005.

[67] P. Dlugosch, D. Brown, P. Glendenning, M. Leventhal, and H. Noyes, "An efficient and scalable semiconductor architecture for parallel automata processing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 12, pp. 3088–3098, Dec. 2014.

[68] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE J. Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, Mar. 2006.

[69] L. W. Tucker and G. G. Robertson, "Architecture and applications of the connection machine," *Computer*, vol. 21, no. 8, pp. 26–38, 1988.

[70] W. D. Hillis, *The Connection Machine*. Cambridge, MA, USA: MIT Press, 1989.

[71] J. J. Hopfield, "Artificial neural networks," *IEEE Circuits Devices Mag.*, vol. 4, no. 5, pp. 3–10, 1988.

[72] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchitect.*, Dec. 2012, pp. 449–460.

[73] J. Hawkins and D. George, *Hierarchical temporal memory: Concepts, theory and terminology*," Numenta Inc., White Paper, 2006. [Online]. Available: http://numenta.com/assets/pdf/whitepapers/hierarchical-temporal-memory-cortical-learning-algorithm-0.2.1-en.pdf

[74] Numenta, Inc., The path to machine intelligence. [Online]. Available: http://numenta.com/assets/pdf/whitepapers/Numenta%20-%20Path%20to%20Machine%20Intelligence%20White%20Paper.pdf

[75] DARPA, The SyNAPSE program. [Online]. Available: http://www.darpa.mil/Our_Work/DSO/Programs/Systems_of_Neuromorphic_Adaptive_Plastic_Scalable_Electronics_%28SYNAPSE%29.aspx

[76] P. A. Merolla *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, Aug. 2014.

ABOUT THE AUTHOR

Ravi Nair (Fellow, IEEE) received the B.Tech. degree in electronics and electrical communications engineering from the Indian Institute of Technology, Kharagpur, India, in 1974 and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana-Champaign, Urbana, IL, USA, in 1976 and 1978, respectively.

Since 1978, he has been with IBM at the Thomas J. Watson Research Center, Yorktown Heights, NY, USA. During 1987–1988, he was on sabbatical leave at the Department of Computer Science, Princeton University, Princeton, NJ, USA. He has also taught at Columbia University, New York, NY, USA. He has worked on processing-in-memory, approximate computing, supercomputer systems, multiprocessor virtualization, processor architecture and microarchitecture, performance evaluation, computer synthesis and analysis of VLSI layout, parallel machines for physical design, fault-tolerant systems, and testing. He has several patents, publications, and awards for work in these areas. He has coauthored the book *Virtual Machines: Versatile Platforms for Systems and Processes* (San Francisco, CA, USA: Morgan Kaufamnn, 2005) with Prof. J. Smith. His current interests include large-scale system design, processor microarchitecture, and approximate computing.

Dr. Nair is a Distinguished Research Staff Member at IBM, and a member of the IBM Academy of Technology.