
COMPARING IMPLEMENTATIONS OF NEAR-DATA COMPUTING WITH IN-MEMORY MAPREDUCE WORKLOADS

Seth H. Pugsley
University of Utah

Jeffrey Jestes
Cerner

Rajeev Balasubramonian
University of Utah

Vijayalakshmi Srinivasan
Alper Buyuktosunoglu
IBM Thomas J. Watson
Research Center

Al Davis
Feifei Li
University of Utah

MOVING COMPUTATION NEAR MEMORY HAS BECOME MORE PRACTICAL BECAUSE OF 3D STACKING TECHNOLOGY. THIS ARTICLE DISCUSSES IN-MEMORY MAPREDUCE IN THE CONTEXT OF NEAR-DATA COMPUTING (NDC). THE AUTHORS CONSIDER TWO NDC ARCHITECTURES: ONE THAT EXPLOITS HYBRID MEMORY CUBE DEVICES AND ONE THAT DOES NOT. THEY EXAMINE THE BENEFITS OF DIFFERENT NDC APPROACHES AND QUANTIFY THE POTENTIAL FOR IMPROVEMENT FOR AN IMPORTANT EMERGING BIG-DATA WORKLOAD.

••••• Warehouse-scale computing is dominated by systems using commodity hardware to execute workloads, processing large amounts of data distributed among many disks. Several frameworks, such as MapReduce,¹ have emerged in recent years to facilitate the management and development of big-data workloads. These systems rely on disks for most data accesses, but placing a large fraction of the data in memory is a growing trend (for example, Memcached, RAMCloud, SAS in-memory analytics, and the SAP HANA in-memory computing and in-memory database platform). Spark is a new MapReduce framework that helps programmers tag objects and cache them in DRAM main memory.² Such a framework is especially useful for iterative MapReduce computations that are popular for graph

processing, machine learning, and interactive queries.² Such in-memory workloads will be limited primarily by the bandwidth and latency to access DRAM main memory and are the targets for the new architectures proposed in this article.

A primary reason for the resurgence of interest in near-data computing (NDC) is the recent emergence of 3D-stacked memory+logic products, such as Micron's Hybrid Memory Cube (HMC).³ Such devices enable colocation of processing and memory in a single package, without impacting the manufacturing process for individual DRAM dies and logic dies. The high bandwidth between the logic and memory dies with through-silicon vias (TSVs) can enable significant speedups for memory-bound applications.

This article first describes the basic NDC framework that appeared in a paper at the 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS).⁴ It then extends that work by considering compelling alternatives. One of these alternative architectures eliminates the use of HMC devices because of their expected high purchase price. The primary contribution of this article is to generalize the NDC approach and compare the merits of these different implementations against a highly optimized non-NDC baseline.

Background

We first review the MapReduce framework and a previously described system optimized for MapReduce execution.

MapReduce workloads

MapReduce applications typically operate on disk-resident data. Large datasets (often key-value pairs) are partitioned into splits. Splits are distributed across several nodes of a commodity cluster. Users provide Map and Reduce functions to aggregate information from these large datasets. A runtime framework, such as Hadoop, is responsible for spawning Map and Reduce tasks, moving data among nodes, and handling fault tolerance.

Map tasks are spawned on all the nodes in the cluster, and each task only works on the splits in that node. A Shuffle phase takes place between Map and Reduce phases, where part of the output of every Map task is moved to another node to be processed by a Reduce task, according to a partitioning function.

The Map and Reduce tasks are both embarrassingly parallel and exhibit localized data accesses. This makes such workloads amenable to NDC. The Shuffle phase is communication intensive, but brief. For many MapReduce workloads, the Map phase takes up most of the execution time, followed by the Reduce and Shuffle phases.

In this article, we consider implementations of the Map and Reduce phases that we have written in C. These codes are run as stand-alone threads in our simulators. Our simulators cannot execute the Java Virtual Machine (JVM) and Hadoop runtimes, so

they are simply modeled as constant overheads at the start and end of each Map and Reduce task. We assume that each split is 128 Mbytes in size and resides in DRAM memory. We execute workloads that have varying bandwidth requirements, ranging from 0.47 to 5.71 bytes per instruction.

An optimized baseline for MapReduce

We now describe a highly optimized baseline for the embarrassingly parallel and bandwidth-intensive in-memory MapReduce workloads.

The processor socket is designed with many low energy-per-instruction (EPI) cores. This design will maximize the number of instructions that are executed per joule and will also maximize the number of instructions executed per unit time, within a given power budget (assuming an embarrassingly parallel workload). We therefore assume an in-order core with power, area, and performance characteristics similar to the ARM Cortex A5 that consumes 80 mW at 1 GHz. Our analysis shows that 512 80-mW cores can be accommodated on a single high-performance chip while leaving enough area and power budget to accommodate an on-chip network, shared last-level cache, and memory controllers.

This many-core processor must be equipped with the most bandwidth-capable memory interface to maximize performance. The HMC uses high-speed SerDes links to connect to the processor. Using only 128 pins, the HMC can provide 80 Gbytes per second (GBps) of bandwidth to the processor socket. The HMC also provides low energy per bit (10.48 pJ/bit versus DDR3's 70 pJ/bit). An important factor in the HMC's energy efficiency is its ability to fetch a cache line from a single DRAM die in the stack, thus limiting the overfetch that plagues a traditional DDR3 memory interface. We therefore connect our many-core processor to four HMC devices with eight high-speed links (each comprising 16 data bits in each direction). Given the limited capacity of an HMC device (4 Gbytes), we increase the board's memory capacity by connecting each HMC to other HMC devices in a daisy chain. We assume four daisy chains per processor socket, with eight HMCs in each daisy chain.

Such a baseline maximizes our workload's processing capability and bandwidth, while retaining the traditional computation model that separates computation and memory to different chips on a motherboard. We refer to this baseline as the EECore (energy-efficient core) model. Such a baseline expends low amounts of energy in computations (low EPI ARM cores) and DRAM array reads (HMC with low overfetch), but it pays a steep price for data movement (multiple high-frequency SerDes hops on the daisy chain and on-chip network navigation on the processor). Such an optimized baseline thus further exposes the communication bottleneck and stands to benefit more from NDC.

Related work

A large body of prior work on processing in memory (PIM)⁵⁻⁸ integrates logic and DRAM arrays on the same chip. Unlike prior work, the analysis in this article focuses on emerging big-data workloads and new incarnations of NDC that use 3D-stacked devices or tightly coupled processors and memory on a daughter card. Recent work in this area includes associative computing accelerators⁹ and NDC with nonvolatile memories.¹⁰⁻¹²

NDC architectures

We consider different implementations of NDC, both using previously proposed HMC-like devices, and using commodity memory arranged on modules with a more conventional memory interface.

NDC with HMCs

In this section, we briefly describe the NDC architecture that we presented at ISPASS 2014.⁴

The NDC architecture is built on a connected network of HMC devices, each of which has 16 80-mW energy-efficient cores placed on its logic chip. These are called near-data cores, or NDCores, and we refer to this augmented HMC as an NDC device.

Each NDCore is tightly coupled to one of the 16 vaults on an NDC device. The vault is a high-bandwidth connection to DRAM banks located directly above the NDCore. The vault has a capacity of 256 Mbytes, providing enough room to store one data split,

intermediate MapReduce outputs, and any code or data belonging to the MapReduce threads and the runtime. The NDCore is connected to a level-1 instruction and data (L1 I and D) cache. On a cache miss, the data is fetched directly from the DRAM arrays directly above. We employ a simple prefetcher for all evaluated systems, which prefetches five total cache lines on a cache miss. In essence, each NDCore and its vault represent a self-contained mininode that can perform a single Map or Reduce task with high efficiency.

Although the ISPASS paper uses a similar topology to the baseline (host processor sockets connected to daisy-chained NDC devices), in this article, we consider a board that uses only NDC devices and eliminates the host processor socket. This approach avoids dealing with the requirement that inactive cores be disabled to save power and enables an apples-to-apples comparison with the NDC-Module design.

We consider both mesh and ring networks to connect the many NDC devices. The mesh requires 2 times the number of SerDes links as the ring network. In both cases, the network serves only to facilitate the Shuffle phase.

NDC with modules

Our new approach to NDC replaces an HMC-based memory system with a memory system comprising commodity LPDDR2 $\times 32$ chips, which are connected via conventional, non-SerDes memory interfaces to low-power multicore processors. These processor cores have the same performance and power characteristics as NDCores. Unlike a traditional server board, where the processor socket is on the motherboard and the memory chips are on add-in DIMMs, the lightweight processor and its associated memory are both soldered onto add-in daughter card modules that are in turn arranged into networks on the motherboard (see Figure 1). We label this approach *NDC-Module*. Map and Reduce phases both again execute on the same processors, with an explicit Shuffle phase between them.

The HMC-based NDC devices are expensive, because of their logic chip and the 3D manufacturing process. LPDDR2 DRAM is cheaper, but compared to NDC-HMC, NDCModule offers less memory

bandwidth to each core, so the opportunity for high performance is not as great. We design the NDC-Module system to have the same memory capacity and number of cores as the NDC-HMC system.

There are two key insights behind the NDC-Module system. First, a standard DDRx channel exposes a small fraction of DRAM chip bandwidth to the processor, because each memory controller pin is shared by many DRAM chips. Moving computation to the DIMM, as in the NDC-Module design, enables available bandwidth to scale linearly with the number of DRAM chips. Second, similar to the HMC, an entire cache line is fetched from a single LPDDR2 chip, reducing activation energy and overfetch.

NDC-Module cards. Each NDC-Module card comprises eight nodes. Each node contains one 8-Gbit LPDDR2 x32 chip and one four-core energy-efficient CMP, similar to the memory interface and core count of an Nvidia Tegra 3 mobile processor. In total, each card contains 8 Gbytes of LPDDR2 DRAM and 32 energy-efficient cores. This is the same ratio of computation cores to DRAM as in NDC-HMC.

LPDDR2 x32 DRAM chips. The NDC-Module system uses commodity LPDDR2 DRAM chips instead of an HMC-based memory system. We use LPDDR2 x32 chips, which can transmit 1,066 Mbits per second (Mbps) per I/O wire, for a total bandwidth of 4.26 GBps per chip. This bandwidth is shared between read and write operations and services only four energy-efficient cores. Each node's bandwidth is independent of other nodes. This design gives a higher per-core theoretical bandwidth than that of the baseline EECORE system, but lower per-core bandwidth than the NDC-HMC system.

NDC-Module motherboard organization. As Figure 1 shows, we organize eight nodes on a single add-in card for a total of 8 Gbytes and 32 low-EPI cores. The NDC-Module system comprises 32 such add-in modules arranged on a single motherboard. We employ PCI-Express 3.0 $\times 16$ to connect all of the

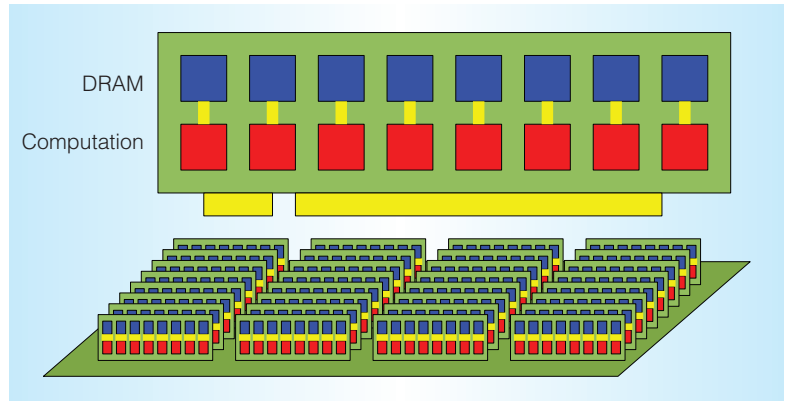


Figure 1. The NDC-Module architecture. A single card (top), and a fully populated system board (bottom).

modules, offering a total of 31.5 GBps of bandwidth with which to perform the Shuffle phase.

Evaluation

In this article, we compare NDC-HMC, with both ring- and mesh-network configurations, to NDC-Module and to our optimized socket-based baseline (EECore). All systems have equal core counts (1,024) and memory capacity (256 Gbytes, storing 1024 128-Mbyte splits) but vary in the connections between processor and memory and in how NDC devices are networked. Figure 2 summarizes all evaluated systems.

Methodology

We use a multistage CPU and memory simulation infrastructure to simulate both CPU and DRAM systems in detail.

To simulate the CPU cores (EE and NDC), we use the Simics full system simulator. To simulate the DRAM, we use the USIMM DRAM simulator,¹³ which has been modified to model an HMC architecture. We assume that the DRAM core latency (Activate + Precharge + ColumnRead) is 40 ns.

Our CPU simulations model a single Map or Reduce thread, and we assume that throughput scales linearly as more cores are used. For the EECORE system, 512-core Simics simulations are not tractable, so we use a trace-based version of the USIMM simulator to model memory contention and then feed these contention estimates into detailed single-thread Simics simulations.

EECore system	
CPU configuration	2x512 cores, 1 GHz
Core parameters	Single-issue in-order
L1 caches	32-Kbyte instruction and data caches, one cycle
NDC cores	—
NDC-HMC system	
NDC cores	1,024 cores, 1 GHz
Core parameters	Single-issue in-order
L1 caches	32-Kbyte instruction and data caches, one cycle
Interdevice connection	8x8 mesh or 1D ring
NDC-Module system	
Add-In Cards (AIC)	32
AIC configuration	8 NDC-Module nodes
NDC-Module node	4 low-EPI cores, 1 Gbyte
Total NDC cores	1,024 cores, 1 GHz
Core parameters	Single-issue in-order
L1 caches	32-Kbyte instruction and data caches, one cycle
Inter-AIC connection	PCI-Express 3.0 x 16

Figure 2. Parameters for the EECore, NDC-HMC, and NDC-Module systems we evaluated. All evaluated systems have the same number of processor cores and total computational capability.

All power and energy evaluations consider workload execution times and component activity rates.

We assume that an HMC's four SerDes links consume a total of 5.78 W per HMC, and the remainder of the logic layer consumes 2.89 W. Total maximum DRAM array power per HMC is assumed to be 4.7 W for an 8-DRAM die.³ We approximate background DRAM array power at 10 percent of this maximum value, or 0.47 W according to the Micron power calculator, and the remaining DRAM power depends on DRAM activity. Energy is consumed in the arrays on each access at the rate of an additional 3.7 pJ/bit.

For LPDDR2, bit-transport energy is 40 pJ/bit, and when operating at peak bandwidth, one LPDDR2 chip consumes 1.36 W of power. Background power for LPDDR2 chips is assumed to be 42.3 mW.

For data that is moved to the processor socket, we add 4.7 pJ/bit to navigate the global wires between the memory controller and the core. This estimate is conservative, because it ignores intermediate routing elements and favors the EECore baseline.

For the core power estimates, we assume that 25 percent of the 80-mW core peak power can be attributed to leakage (20 mW). The dynamic power for the core varies linearly between 30 and 60 mW, based on IPC

(because many circuits are switching even during stall cycles).

Workloads

We evaluate the Map and Reduce phases of five MapReduce workloads—namely, Group-By Aggregation (GroupBy), Range Aggregation (RangeAgg), Equi-Join Aggregation (EquiJoin), Word Count Frequency (WordCount), and Sequence Count Frequency (SequenceCount).

GroupBy and EquiJoin both involve sorting as part of their Map phase, but the RangeAgg workload is simply a high-bandwidth Map scan through the input split. These first three workloads use the 1998 World Cup website log as input.¹⁴

WordCount and SequenceCount find the frequency of words or sequences of words in large HTML files, and as input we use Wikipedia HTML data.¹⁵ These workloads also involve sorting, but parsing and sorting text items is more computationally intensive than sorting integers, so these workloads are more CPU-bound than the others.

When executing on the EECore system, a RangeAgg MapReduce task takes on the order of milliseconds to complete, GroupBy and EquiJoin take on the order of seconds to complete, and WordCount and SequenceCount take on the order of minutes to complete.

For each workload, we have also added 1 ms execution time overheads for beginning a new Map phase, transitioning between Map and Reduce phases, and for completing a job after the Reduce phase. This conservatively models the MapReduce runtime overheads and the cost of cache flushes between phases. Even if these estimates are off by three orders of magnitude, they would still have a negligible impact on the longer running workloads.

Performance results

We evaluate the proposed NDC systems in the context of bandwidth usage, absolute MapReduce performance, and energy efficiency.

Bandwidth

Unlike the EECore system, the NDC systems are not limited by host CPU processor socket bandwidth during Map phase execution (see Figure 3). In RangeAgg, we see that the NDC-HMC system can attain an average of 3.4 Tbytes per second (TBps) of aggregate read bandwidth, and the NDC-Module system can attain an average of 580 GBps aggregate read bandwidth, but the EECore system is limited to only using 260 GBps of aggregate read bandwidth.

We see that RangeAgg is memory bandwidth bound, that WordCount and SequenceCount are CPU-bound, and that GroupBy and EquiJoin fall somewhere in between. All of the performance advantage, and much of the energy advantage, of the NDC systems over the EECore system comes from high-bandwidth access to memory.

MapReduce performance

Figure 4 shows how execution time is split between Map, Shuffle, and Reduce phases for each workload and shows the relative execution times for each system. Map phase performance directly tracks with the memory bandwidth numbers discussed previously for each system and workload, so the NDC-HMC systems have the highest performance, followed by the NDC-Module system.

The Shuffle phase generally takes a very small amount of overall MapReduce execution time. Only when executing the GroupBy

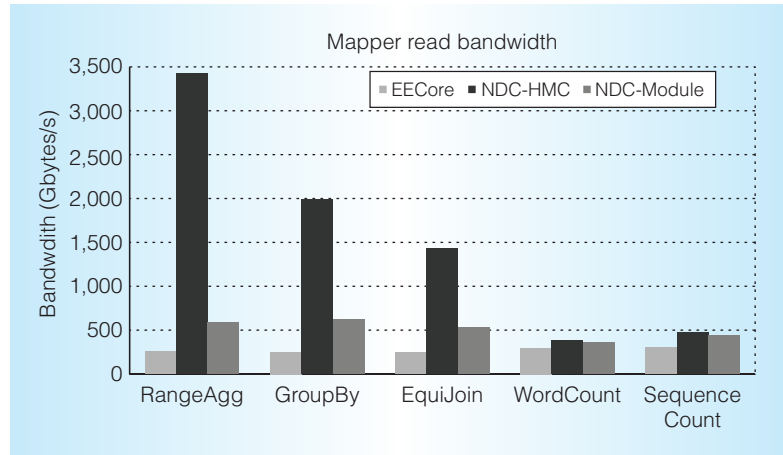


Figure 3. Average read bandwidth usage during Map phase for an entire system. There are no shared memory interfaces in NDC-HMC, so bandwidth usage is limited only by processor performance.

and EquiJoin workloads does the Shuffle phase have a nontrivial impact on performance. However, even in the worst case, performance for EquiJoin is hurt by only 8.4 percent when using NDC-HMC Ring, versus NDC-HMC Mesh. The EECore system does not use the Shuffle phase at all. Instead, the EE system uses long-latency random accesses to memory during the Reduce phase.

Compared to the EECore baseline, NDC-HMC Mesh and NDC-HMC Ring both reduce execution time by 23.7 percent (WordCount) to 92.7 percent (RangeAgg). NDC-Module reduces execution time over the EECore system by 16.5 percent (WordCount) to 55.5 percent (RangeAgg).

Energy consumption

We consider both static and dynamic energy in evaluating the energy and power consumption of EE and NDC systems. Figure 5 shows the breakdown in energy consumed by the memory subsystem and the processing cores. All three NDC systems improve on the energy efficiency of the EE baseline, in large part because of improved task execution times.

NDC-HMC Ring improves on the energy efficiency of NDC-HMC Mesh because it uses fewer SerDes links. The gap between the Mesh and Ring systems is narrowest for GroupBy and EquiJoin, which spend longer in their Shuffle phase—the only

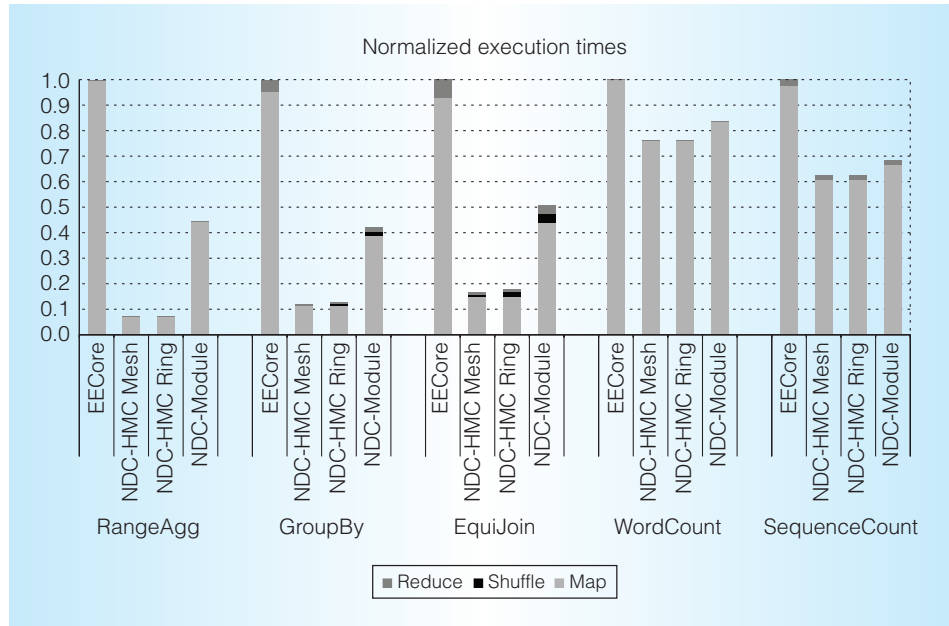


Figure 4. Execution time for an entire MapReduce job normalized to the EE system. Shuffle time does not have a large impact on performance, even in systems with low Shuffle bandwidth.

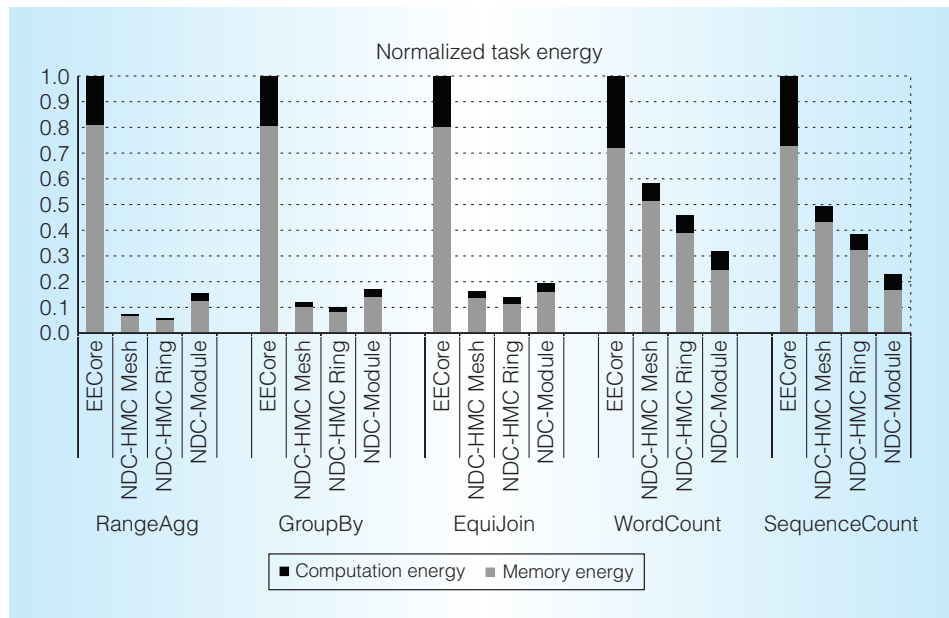


Figure 5. Energy consumed by the memory and processing resources, normalized to the EE system. All three NDC systems improve on the EE baseline's energy efficiency.

phase where having more SerDes links helps performance.

The NDC-Module system uses LPDDR2, which has a higher energy cost per access,

compared to an HMC-based memory, but lower background power because of its lack of a SerDes interface. The CPU-bound WordCount and SequenceCount

workloads have many opportunities for the LPDDR2 interface to be idle, so NDC-Module uses the least energy in these two workloads.

Data parallel workloads like MapReduce benefit when useable memory bandwidth can be extracted from every memory chip in the system simultaneously. In this article, we have focused only on executing a single MapReduce task on a single, isolated system, rather than on a cluster of connected systems. Future work will seek to verify that NDC performance scales to larger clusters, and that neither communication nor software overheads outweigh the benefit offered by NDC. Also, the use of commodity memory in the NDC-Module architecture could lead to substantial cost savings compared to NDC-HMC; however, this benefit has not yet been quantified. Finally, iterative MapReduce, and other non-MapReduce application types, offer additional challenges and opportunities for NDC architectures that are unaddressed by this article, but which are worth further exploration.

MICRO

References

1. J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Proc. 6th Conf. Operating Systems Design & Implementation*, 2004, article 10.
2. M. Zaharia et al., "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing," *Proc. 9th USENIX Conf. Networked Systems Design and Implementation*, 2012, article 2.
3. J. Jeddelloh and B. Keeth, "Hybrid Memory Cube—New DRAM Architecture Increases Density and Performance," *Proc. Symp. VLSI Technology*, 2012, pp. 87-88.
4. S. Pugsley et al., "NDC: Analyzing the Impact of 3D-Stacked Memory + Logic Devices on MapReduce Workloads," to be published in *Proc. IEEE Int'l Symp. Performance Analysis of Systems and Software (ISPASS 14)*, 2014.
5. M. Hall et al., "Mapping Irregular Applications to DIVA, a PIM-Based Data-Intensive Architecture," *Proc. ACM/IEEE Conf. Supercomputing*, 1999, p. 57.
6. D. Patterson et al., "A Case for Intelligent DRAM: IRAM," *IEEE Micro*, vol. 17, no. 2, 1997, pp. 34-44.
7. Y. Kang et al., "FlexRAM: Toward an Advanced Intelligent Memory System," *Proc. Int'l Conf. Computer Design*, 1999, pp. 192-201.
8. T. Kgil et al., "PicoServer: Using 3D Stacking Technology to Enable a Compact Energy Efficient Chip Multiprocessor," *Proc. 12th Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, 2006, pp. 117-128.
9. Q. Guo et al., "AC-DIMM: Associative Computing with STT-MRAM," *Proc. 40th Ann. Int'l Symp. Computer Architecture*, 2013, pp. 189-200.
10. P. Ranganathan, "From Microprocessors to Nanostores: Rethinking Data-Centric Systems," *Computer*, Jan. 2011, pp. 39-48.
11. A. Caulfield, L. Grupp, and S. Swanson, "Gordon: Using Flash Memory to Build Fast, Power-Efficient Clusters for Data-Intensive Applications," *Proc. 14th Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, 2009, pp. 217-228.
12. A. Gutierrez et al., "Integrated 3D-Stacked Server Designs for Increasing Physical Density of Key-Value Stores," *Proc. 19th Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, 2014, pp. 485-498.
13. N. Chatterjee et al., *USIMM: The Utah Simulated Memory Module*, tech. report UUCS-12-002, Univ. of Utah, 2012.
14. M. Arlitt and T. Jin, *1998 World Cup Web Site Access Logs*, Aug. 1998, <ftp://ita.ee.lbl.gov/html/contrib/WorldCup.html>.
15. F. Ahmad, "PUMA Benchmarks and Dataset Downloads," <https://sites.google.com/vls/farazahmad/pumadatasets>.

Seth H. Pugsley is a PhD candidate in computer science at the University of Utah. His research focuses on memory system design, including caching policies, hardware prefetching, main memory system organization, and near-data processing. Pugsley has a

BS in computer science from the University of Utah.

Jeffrey Jestes is a senior software engineer at Cerner. His research interests include summarizing massive data in distributed and parallel frameworks; ranking, monitoring, and tracking big data; and scalable query processing in large databases. Jestes has a PhD in computing from the University of Utah.

Rajeev Balasubramonian is an associate professor in the School of Computing at the University of Utah. His research focuses on memory systems and interconnects. Balasubramonian has a PhD in computer science from the University of Rochester. He is a member of IEEE.

Vijayalakshmi Srinivasan is a research staff member at the IBM Thomas J. Watson Re-

search Center. Her research focuses on computer architecture, particularly processor and memory microarchitecture, data-center architectures, and big data analytics. Srinivasan has a PhD in computer science from the University of Michigan. She is a senior member of IEEE.

Alper Buyuktosunoglu is a research staff member in the Reliability and Power-Aware Microarchitecture department at the IBM Thomas J. Watson Research Center. His research focuses on high-performance and power- and reliability-aware computer architectures. Buyuktosunoglu has a PhD in electrical and computer engineering from University of Rochester. He is an IBM Master Inventor, a senior member of IEEE, and an editorial board member of *IEEE Micro*.

Al Davis is a professor in the School of Computing at the University of Utah. He is also a part-time visiting scholar at HP Laboratories. His research interests include computer architecture, VLSI, and photonics. Davis has a PhD in computer science from the University of Utah.

Feifei Li is an associate professor in the School of Computing at the University of Utah. His research interests include database and data management systems and big data analytics. Li has a PhD in computer science from Boston University.

Direct questions and comments about this article to Seth H. Pugsley, 50 S. Central Campus Drive, Room 3190, Salt Lake City, Utah 84112; pugsley@cs.utah.edu.

ADVERTISER SALES INFORMATION

Advertising Personnel

Marian Anderson
Sr. Advertising Coordinator
Email: manderson@computer.org
Phone: +1 714 816 2139
Fax: +1 714 821 4010

Sandy Brown
Sr. Business Development Mgr.
Email: sbrown@computer.org
Phone: +1 714 816 2144
Fax: +1 714 821 4010

Advertising Sales Representatives (display)

Central, Northwest, Far East:
Eric Kincaid
Email: e.kincaid@computer.org
Phone: +1 214 673 3742
Fax: +1 888 886 8599

Northeast, Midwest, Europe,
Middle East:
Ann & David Schissler
Email: a.schissler@computer.org,
d.schissler@computer.org
Phone: +1 508 394 4026
Fax: +1 508 394 1707

Southwest, California:
Mike Hughes
Email: mikehughes@computer.org
Phone: +1 805 529 6790


Southeast:
Heather Buonadies
Email: h.buonadies@computer.org
Phone: +1 973 585 7070
Fax: +1 973 585 7071

Advertising Sales Representative (Classified Line)

Heather Buonadies
Email: h.buonadies@computer.org
Phone: +1 973 304 4123
Fax: +1 973 585 7071

Advertising Sales Representative (Jobs Board)

Heather Buonadies
Email: h.buonadies@computer.org
Phone: +1 973 304 4123
Fax: +1 973 585 7071

 Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.