NDC: Analyzing the Impact of 3D-Stacked Memory+Logic Devices on MapReduce Workloads

Seth H Pugsley[1], Jeffrey Jestes[1], Huihui Zhang[1], Rajeev Balasubramonian[1], Vijayalakshmi Srinivasan[2], Alper Buyuktosunoglu[2], Al Davis[1], and Feifei Li[1]

[1]University of Utah, {pugsley, jestes, huihui, rajeev, ald, lifeifei}@cs.utah.edu
[2]IBM T.J. Watson Research Center, {viji, alperb}@us.ibm.com

Abstract

While Processing-in-Memory has been investigated for decades, it has not been embraced commercially. A number of emerging technologies have renewed interest in this topic. In particular, the emergence of 3D stacking and the imminent release of Micron's Hybrid Memory Cube device have made it more practical to move computation near memory. However, the literature is missing a detailed analysis of a killer application that can leverage a Near Data Computing (NDC) architecture. This paper focuses on in-memory MapReduce workloads that are commercially important and are especially suitable for NDC because of their embarrassing parallelism and largely localized memory accesses. The NDC architecture incorporates several simple processing cores on a separate, non-memory die in a 3D-stacked memory package; these cores can perform Map operations with efficient memory access and without hitting the bandwidth wall. This paper describes and evaluates a number of key elements necessary in realizing efficient NDC operation: (i) low-EPI cores, (ii) long daisy chains of memory devices, (iii) the dynamic activation of cores and SerDes links. Compared to a baseline that is heavily optimized for MapReduce execution, the NDC design yields up to 15X reduction in execution time and 18X reduction in system energy.

1. Introduction

A large fraction of modern-day computing is performed within warehouse-scale computers. These systems execute workloads that process large amounts of data, scattered across many disks and many low-cost commodity machines. A number of frameworks, such as MapReduce [20], have emerged in recent years to facilitate the management and development of big-data workloads. While current incarnations of these systems rely on disks for most data accesses, there is a growing trend towards placing a large fraction of the data in memory, e.g., Memcached [5], RAMCloud [49], and Spark [62]. The RAMCloud project shows that if a workload is limited more by data bandwidth than by capacity, it is better to store the data in a distributed memory platform than in a distributed HDD/SSD platform. There are therefore many big-data workloads that exhibit lower cost and higher performance with in-memory storage [49]. This is also evident in the commercial world, e.g., SAS in-memory analytics [58], SAP HANA in-memory computing and in-memory database platform [57], and BerkeleyDB [14] (used in numerous embedded applications that require a key-value-like storage engine). Let us consider SAP HANA as a concrete example. It employs a cluster of commodity machines as its underlying storage and computation engine, and it relies on the collective DRAM memory space provided by all nodes in the cluster to store large data entirely in memory. Each node can provide terabytes of memory, and collectively, they deliver an in-memory storage space that can hold up to hundreds of terabytes of data, depending on the size of the cluster [26, 25].

In-memory storage of big-data is also being made possible by technology innovations such as 3D-stacked memory and memory blades, and emerging non-volatile cells that focus on improving capacity and persistence. For example, the recent emergence of 3D-stacked memory products [54, 51, 60, 24] will likely benefit such in-memory big-data workloads.

There is great interest in designing architectures that are customized for emerging big-data workloads. For example, a recent paper [43] designs a custom core and NIC for Memcached. In this work, we make a similar attempt for in-memory MapReduce workloads. We take advantage of emerging 3D-stacked memory+logic devices (such as Micron's Hybrid Memory Cube or HMC [27]) to implement a Near Data Computing (NDC) architecture, which is a realizable incarnation of processing-in-memory (PIM). A single board is designed to accommodate several daisy-chained HMC-like devices, each of which includes a few low-power cores that have highly efficient access to a few giga-bytes of data on the 3D stack. MapReduce applications are embarrassingly parallel and exhibit highly localized memory access patterns, and are therefore very good fits for NDC.

While Memcached has the same behavior on every invocation and can benefit from customization [43], MapReduce functions can take several forms and therefore require programmability. We show that efficiency and cost are optimized by using low Energy Per Instruction general-purpose cores, by implementing long daisy chains of memory devices, and by dynamically activating cores and off-chip SerDes links. Our work is the first thorough quantitative study analyzing the effect of HMC-like devices for in-memory Map-Reduce workloads. We lay the foundation for future studies that can further explore the NDC architecture and apply it to other big-data workloads that have high degrees of parallelism and memory locality.

2. MapReduce Background

MapReduce workloads are applied to databases, and are comprised of two phases. The Mapper is assigned to work on a slice

of the database, called a database split. In this paper, we assume 128 MB database splits. The Mapper performs its function on this split and prepares the result to be handed off to a Reducer. Each Reducer takes as its input a portion of the output from each Mapper, and performs its function on this set to produce the final output. Next, we go into some detail about each step.

2.1. Mapper

Map. The Mapper applies the Map function to all records in the input split, typically producing key-value pairs as output from this stage. This is a linear scan of the input split, so this phase is highly bandwidth intensive. The computational complexity varies across workloads.

Sort. The Mapper next sorts the set of key-value pairs by their keys, with an in-place quick sort.

Combine. The Combine phase is applied to the local output of the Mapper, and can be viewed as a local Reduce function. This phase involves a linear scan through the sorted output data, applying the Reduce function to each key (with its associated set of values) in the output set.

Partition. The Mapper's final action is to divide its sorted-and-combined output into a number of partitions equal to the number of Reducers in the system. Each Reducer gets part of the output from each Mapper. This is done by another linear scan of the output, copying each item into its correct partition.

2.2. Reducer

Shuffle and Sort. The Reducer's first job is to gather all of its input from the various Mapper output partitions, which are scattered throughout the system, into a single, sorted input set. This is done by a merge of all the already-sorted partitions into a single input set (a merge sort).

Reduce. Finally, the Reducer applies the Reduce function to all of the keys (with their associated sets of values) in its sorted input set. This involves a linear scan of its input, applying the Reduce function to each item.

2.3. Computational Requirements

Mappers and Reducers have different computational and bandwidth needs. The Map phase is largely bandwidth constrained, and consumes the bulk of the execution time for our workloads. It would therefore be beneficial to execute the Mapper on processors that have high levels of memory bandwidth, and not necessarily high single-thread performance.

3. Memory System Background

3.1. Moving from DDR3 to HMC

In a conventional memory system, a memory controller on the processor is connected to dual in-line memory modules (DIMMs) via an off-chip electrical DDR3 memory channel (bus). Modern processors have as many as four memory controllers and four DDR3 memory channels [4]. Processor pin counts have neared scaling limits [35]. Efforts to continually boost processor pin bandwidth lead to higher power consumption and limit per-pin memory capacity, thus it is hard to simultaneously support higher memory capacity and memory bandwidth.

	Pins	Bandwidth	Power
DDR3	143	12.8 GB/s	6.2 W
DDR4	148	25.6 GB/s	8.4 W
HMC	128	80.0 GB/s	13.4 W

Table 1: Memory Technology Comparison.

Recently, Micron has announced the imminent release of its Hybrid Memory Cube (HMC) [51]. The HMC uses 3D die-stacking to implement multiple DRAM dies and an interface logic chip on the same package. TSVs are used to ship data from the DRAM dies to the logic chip. The logic chip implements high-speed signaling circuits so it can interface with a processor chip through fast, narrow links.

3.2. Analyzing an HMC-Based Design

In Table 1, we provide a comparison between DDR3, DDR4, and HMC-style baseline designs, in terms of power, bandwidth and pin-count. This comparison is based on data and assumptions provided by Micron [27, 36].

HMC is optimized for high-bandwidth operation and targets workloads that are bandwidth-limited. HMC has better bandwidth-per-pin, and bandwidth-per-watt characteristics than either DDR3 or DDR4. We will later show that the MapReduce applications we consider here are indeed bandwidth-limited, and will therefore best run on systems that maximize bandwidth for a given pin and power budget.

4. Related Work

2D Processing-in-Memory: Between 1995-2005, multiple research teams built 2D PIM designs and prototypes (e.g., [32, 50, 38, 48]) and confirmed that there was potential for great speedup in certain application classes, such as media [29, 38], irregular computations [15, 32], link discovery algorithms [10], query processing [32, 47, 38], etc.

None of this prior work has exploited 3D stacking. While a few have examined database workloads, none have leveraged the MapReduce framework to design the application and to map tasks automatically to memory partitions. MapReduce is unique because the Map phase exhibits locality and embarrassing parallelism, while the Reduce phase requires high-bandwidth random memory access. We show that NDC with a 3D-stacked logic+memory device is a perfect fit because it can handle both phases efficiently. We also argue that dynamic activation of cores and SerDes links is beneficial because each phase uses a different set of cores and interconnects. This compelling case for NDC is made possible by the convergence of emerging technology (3D stacking), workloads (big-data analytics), and mature programming models (MapReduce).

3D Stacking: A number of recent papers have employed 3D stacking of various memory chips on a processor chip (e.g.,

[44, 45, 46, 22, 59, 61, 28, 40]) to reduce memory latencies. Even a stack of 4 DRAM chips can only offer a maximum capacity of 2 GB today. Hence, in the high-performance domain, such memory chips typically serve as a cache [37] and must be backed up by a traditional main memory system. Loh [44] describes various design strategies if the memory chips were to be used as main memory. Kim et al. [40] and Fick et al. [28] build proof-of-concept 3D-stacked devices that have 64 cores on the bottom die and small SRAM caches on the top die. These works do not explore the use of similar future devices for big-data processing. None of this prior work aggregates several 3D-stacked devices on a single board to cost-effectively execute big-data workloads. The 3D-Maps prototype has measured the bandwidth and power for some kernels, including the histogram benchmark that resembles the data access pattern of some Map phases [40]. Industrial 3D memory prototypes and products include those from Samsung [55, 54], Elpida [23, 24], Tezzaron [60], and Micron [3, 51]. Many of these employ a logic controller at the bottom of the stack with undisclosed functionality. Tezzaron plans to use the bottom die for self-test and soft/hard error tolerance [60]. Micron has announced an interest in incorporating more sophisticated functionality on the bottom die [9].

Custom Architectures for Big-Data Processing: Some papers have argued that cost and energy efficiency are optimized for cloud workloads by using many "wimpy" processors and replacing disk access with Flash or DRAM access [42, 11, 49, 16]. Chang et al. [52, 17] postulate the Nanostore idea, where a 3D stack of non-volatile memory is bonded to a CPU. They evaluate specific design points that benefit from fast NVM access (relative to SSD/HDD) and a shallow memory hierarchy. Lim et al. [43] customize the core and NIC to optimize Memcached execution. Guo et al. [31, 30] design associative TCAM accelerators that help reduce data movement costs in applications that require key-value pair retrieval. The design relies on custom memory chips and emerging resistive cells. Phoenix [53] is a programming API and runtime that implements MapReduce for shared-memory systems. The Mars framework does the same for GPUs [33]. DeKruijf and Sankaralingam evaluate MapReduce efficiency on the Cell Processor [21]. A recent IBM paper describes how graph processing applications can be efficiently executed on a Blue Gene/Q platform [19].

5. Near Data Computing Architecture

5.1. High Performance Baseline

A Micron study [36] shows that energy per bit for HMC access is measured at 10.48 pJ, of which 3.7 pJ is in the DRAM layers and 6.78 pJ is in the logic layer. If we assume an HMC device with four links that operate at their peak bandwidth of 160 GB/s, the HMC and its links would consume a total of 13.4 W. About 43% of this power is in the SerDes circuits used for high-speed signaling [36, 56]. In short, relative to DDR3/DDR4 devices, the HMC design is paying a steep power penalty for its superior bandwidth. Also note that SerDes links cannot be easily powered down because of their long wake-up times. So the HMC will dissipate at least 6 W even when idle.

Energy Efficient / ND Core	
Process	32 nm
Power	80 mW
Frequency	1 GHz
Core Type	single-issue in-order
Caches	32 KB I and D
Area (incl. caches)	0.51 mm^2

EE Core Chip Multiprocessor	
Core Count	512
Core Power	41.0 W
NOC Power	36.0 W
LLC and IMC	20.0 W
Total CMP Power	97.0 W

Table 2: Energy Efficient Core (EECore) and baseline system

We begin by considering a server where a CPU is attached to 4 HMC devices with 8 total links. Each HMC has a capacity of 4 GB (8 DRAM layers each with 4 Gb capacity). This system has a memory bandwidth of 320 GB/s (40 GB/s per link) and a total memory capacity of 16 GB. Depending on the application, the memory capacity wall may be encountered before the memory bandwidth wall.

Memory capacity on the board can be increased by using a few links on an HMC to connect to other HMCs. In this paper, we restrict ourselves to a daisy-chain topology to construct an HMC network. Daisy chains are simple and have been used in other memory organizations, such as the FB-DIMM. We assume that the processor uses its eight links to connect to four HMCs (two links per HMC), and each HMC connects two of its links to the next HMC in the chain (as seen in Figure 1b). While daisy chaining increases the latency and power overhead for every memory access, it is a more power-efficient approach than increasing the number of system boards.

For power-efficient execution of embarrassingly-parallel workloads like MapReduce, it is best to use as large a number of low energy-per-instruction (EPI) cores as possible. This 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. According to the analysis of Azizi et al. [13], at low performance levels, the lowest EPI is provided by a single-issue in-order core. This is also consistent with data on ARM processor specification sheets. We therefore assume an in-order core similar to the ARM Cortex A5 [1].

Parameters for a Cortex A5-like core, and a CMP built out of many such cores, can be found in Table 2. Considering that large server chips can be over 400 mm^2 in size. We assume that 512 such cores are accommodated on a server chip (leaving enough room for interconnect, memory controllers, etc.). To construct this table, we calculated the power consumed by on-chip wires to support its off-chip bandwidth, not including the overheads for the intermediate routers [39], we calculated the total power consumed by the on-chip network [41], and fac-

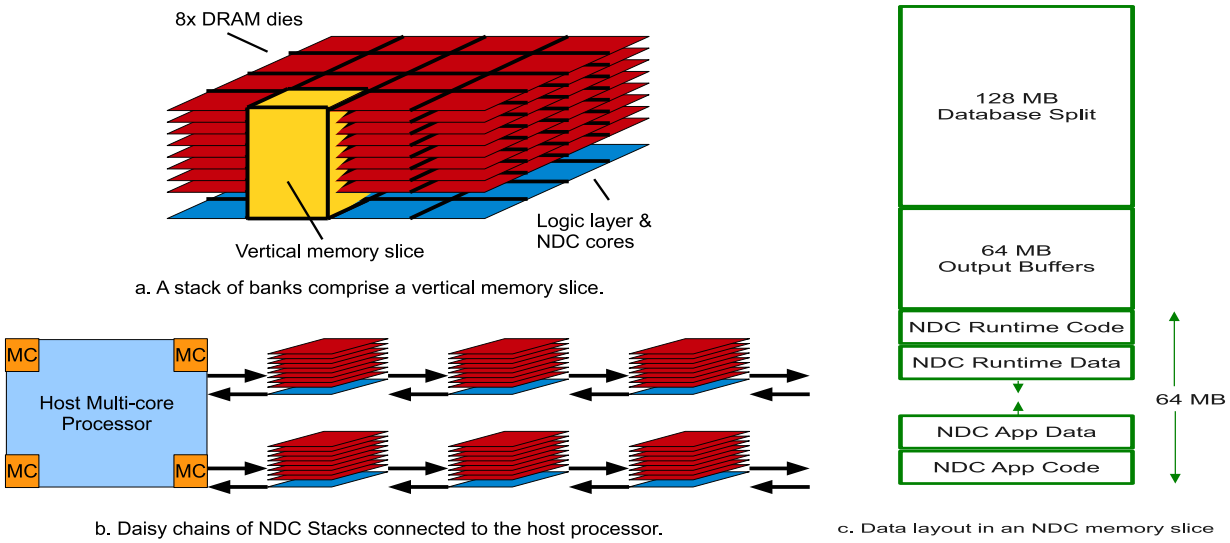

Figure 1: The Near Data Computing Architecture.

tored in the power used by the last level caches and memory controllers [34]. This is a total power rating similar to that of other commercial high-end processors [4]

The processor can support a peak total throughput of 512 BIPS and 160 GB/s external read memory bandwidth, i.e., a peak bandwidth of 0.32 read bytes/instruction can be sustained. On such a processor, if the application is compute-bound, then we can build a simpler memory system with DDR3 or DDR4. Our characterization of MapReduce applications shows that the applications are indeed memory-bound. The read bandwidth requirements of our applications range from 0.47 bytes/instruction to 5.71 bytes/instruction. So the HMC-style memory system is required.

We have designed a baseline server that is optimized for in-memory MapReduce workloads. However, this design pays a significant price for data movement: (i) since bandwidth is vital, high-speed SerDes circuits are required at the transmitter and receiver, (ii) since memory capacity is vital to many workloads, daisy-chained devices are required, increasing the number of SerDes hops to reach the memory device, (iii) since all the computations are aggregated on large processor chips, large on-chip networks have to be navigated to reach the few high-speed memory channels on the chip.

5.2. NDC Hardware

We next show that a more effective approach to handle MapReduce workloads is to move the computation to the 3D-stacked devices themselves. We refer to this as Near Data Computing to differentiate it from the processing-in-memory projects that placed logic and DRAM on the same chip and therefore had difficulty with commercial adoption.

While the concept of NDC will be beneficial to any memory bandwidth-bound workload that exhibits locality and high parallelism, we use MapReduce as our evaluation platform in this study. Similar to the baseline, a central host processor with many EECores is connected to many daisy-chained memory devices augmented with simple cores. The Map phases of MapReduce workloads exhibit high data locality and can be executed on the memory device; the Reduce phase also exhibits high data locality, but it is still executed on the central host processor chip because it requires random access to data. For random data accesses, average hop count is minimized if the requests originate in a central location, i.e., at the host processor. NDC improves performance by reducing memory latency and by overcoming the bandwidth wall. We further show that the proposed design can reduce power by disabling expensive SerDes circuits on the memory device and by powering down the cores that are inactive in each phase. Additionally, the NDC architecture scales more elegantly as more cores and memory are added, favorably impacting cost.

3D NDC Package. As with an HMC package, we assume that the NDC package contains 8 4 Gb DRAM dies stacked on top of a single logic layer. The logic layer has all the interface circuitry required to communicate with other devices, as in the HMC. In addition, we introduce 16 simple processor cores (Near-Data Cores, or NDCores).

3D Vertical Memory Slice. In an HMC design, 32 banks are used per DRAM die, each with capacity 16 MB (when assuming a 4 Gb DRAM chip). When 8 DRAM die are stacked on top of each other, 16 banks align vertically to comprise one 3D vertical memory slice, with capacity 256 MB, as seen in Figure 1a. Note that a vertical memory slice (referred to as a "vault" in HMC literature) has 2 banks per die. Each 3D vertical memory slice is connected to an NDCore below on the logic layer by Through-Silicon Vias (TSVs). Each NDCore operates exclusively on 256 MB of data, stored in 16 banks directly above it. NDCores have low latency, high bandwidth access to their 3D slice of memory. In the first-generation HMC, there are 1866 TSVs, of which, 512 are used for data transfers at 2 Gb/s each [36].

NDCores. Based on our analysis earlier, we continue to use low-EPI cores to execute the embarrassingly parallel Map phase. We again assume an in-order core similar to the ARM Cortex A5 [1]. Each core runs at a frequency of 1 GHz and consumes

80 mW, including instruction and data caches. We are thus adding only 1.28 W total power to the package (and will shortly offset this with other optimizations). Given the spatial locality in the Map phase, we assume a prefetch mechanism that fetches five consecutive cache lines on a cache miss. We also apply this prefetching optimization to all baseline systems tested, not just NDC, and it helps the baseline systems more than the NDC system, due to their higher latency memory access time.

Host CPUs and 3D NDC Packages.

Because the host processor socket has random access to the entire memory space, we substitute the Shuffle phase with a Reduce phase that introduces a new level of indirection for data access. When the Reduce phase touches an object, it is fetched from the appropriate NDC device (the device where the object was produced by a Mapper). This is a departure from the typical Map, Shuffle, and Reduce pattern of MapReduce workloads, but minimizes data movement when executing on a central host CPU. The Reduce tasks are therefore executed on the host processor socket and its 512 EECores, with many random data fetches from all NDC devices. NDC and both baselines follow this model for executing the Reduce phase.

Having full-fledged traditional processor sockets on the board allows the system to default to the baseline system in case the application is not helped by NDC. The NDCores can remain simple as they are never expected to handle OS functionality or address data beyond their vault. The overall system architecture therefore resembles the optimized HMC baseline we constructed in Section 5.1. Each board has two CPU sockets. Each CPU socket has 512 low-EPI cores (EECores). Each socket has eight high-speed links that connect to four NDC daisy-chains. Thus, every host CPU core has efficient (and conventional) access to the board's entire memory space, as required by the Reduce function.

Power Optimizations. Given the two distinct phases of MapReduce workloads, the cores running the Map and Reduce phases will never be active at the same time. If we assume that the cores can be power-gated during their inactive phases, the overall power consumption can be kept in check.

Further, we maintain power-neutrality within the NDC package. This ensures that we are not aggravating thermal constraints in the 3D package. In the HMC package, about 5.7 W can be attributed to the SerDes circuits used for external communication. HMC devices are expected to integrate 4-8 external links and we've argued before that all of these links are required in an optimal baseline. However, in an NDC architecture, external bandwidth is not as vital because it is only required in the relatively short Reduce phase. To save power, we therefore permanently disable 2 of the 4 links on the HMC package. This 2.85 W reduction in SerDes power offsets the 1.28 W power increase from the 16 NDCores.

The cores incur a small area overhead. Each core occupies 0.51 mm^2 in 32 nm technology. So the 16 cores only incur a 7.6% area overhead, which could also be offset if some HMC links were outright removed rather than just being disabled.

Regardless of whether power-gating is employed, we expect that the overall system will consume less energy per workload task. This is because the energy for data movement has been greatly reduced. The new design consumes lower power than the baseline by disabling half the SerDes circuits. Faster execution times will also reduce the energy for constant components (clock distribution, leakage, etc.).

5.3. NDC Software

User Programmability. Programming for NDC is similar to the programming process for MapReduce on commodity clusters. The user supplies Map and Reduce functions. Behind the scenes, the MapReduce runtime coordinates and spawns the appropriate tasks.

Data Layout. Each 3D vertical memory slice has 256 MB total capacity, and each NDCore has access to one slice of data. For our workloads, we populate an NDCore's 256 MB of space with a single 128 MB database split, 64 MB of output buffer space, and 64 MB reserved for code and stack space, as demanded by the application and runtime. Each of these three regions is treated as large superpages. The first two superpages can be accessed by their NDCore and by the central host processor. The third superpage can only be accessed by the NDCore. The logical data layout for one database split is shown in Figure 1c.

MapReduce Runtime. Runtime software is required to orchestrate the actions of the Mappers and Reducers. Portions of the MapReduce runtime execute on the host CPU cores and portions execute on the NDCores, providing functionalities very similar to what might be provided by Hadoop. The MapReduce runtime can serve as a lighweight OS for an NDCore, ensuring that code and data do not exceed their space allocations, and possibly re-starting a Mapper on a host CPU core if there is an unserviceable exception or overflow.

6. Evaluation

6.1. Evaluated Systems

In this work, we compare an NDC-based system to two baseline systems. The first system uses a traditional out-of-order (OoO) multi-core CPU, and the other uses a large number of energy-efficient cores (EECores). Both of these processor types are used in a 2-socket system connected to 256 GB of HMC memory capacity, which fits 1024 128 MB database splits. All evaluated systems are summarized in Table 3.

6.1.1. OoO System On this system, both the Map and Reduce phases of MapReduce run on the high performance CPU cores on the two host sockets. Each of the 16 OoO cores must sequentially process 64 of the 1024 input splits to complete the Map phase. As a baseline, we assume perfect performance scaling for more cores, and ignore any contention for shared resources, other than memory bandwidth, to paint this system configuration in the best light possible.

6.1.2. EECore System Each of the 1024 EECores must compute only one each of the 1024 input splits in a MapReduce workload. Although the frequency of each EECore is much lower than an OoO core, and the IPC of each EECore is lower

Out-of-Order System	
CPU configuration	2x 8 cores, 3.3 GHz
Core parameters	4-wide out-of-order
	128-entry ROB
L1 Caches	32 KB I and D, 4 cycle
L2 Cache	256 KB, 10 cycle
L3 Cache	2 MB, 20 cycle
NDC Cores	—
EECore System	
CPU configuration	2x 512 cores, 1 GHz
Core parameters	single-issue in-order
L1 Caches	32 KB I and D, 1 cycle
NDC Cores	—
NDC System	
CPU configuration	2x 512 cores, 1 GHz
Core parameters	single-issue in-order
L1 Caches	32 KB I and D, 1 cycle
NDC Cores	1024

Table 3: System parameters.

than an OoO core, the EECore system still has the advantage of massive parallelism, and we show in our results that this is a net win for the EECore system by a large margin.

6.1.3. NDCore System We assume the same type and power/frequency cores for NDCores as EECores. The only difference in their performance is the way they connect to memory. EECores must share a link to the system of connected HMCs, but each NDCore has a direct link to its dedicated memory, with very high bandwidth, and lower latency. This means NDCores will have higher performance than EECores.

In order to remain power neutral compared to the EECore system, each HMC device in the NDC system has half of its 4 data links disabled, and therefore can deliver only half the bandwidth to the host CPU, negatively impacting Reduce performance.

6.2. Workloads

We evaluate the Map and Reduce phases of 5 different 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 a sort, a combine, and a partition in their Map phase, in addition to the Map scan, but the *RangeAgg* workload is simply a high-bandwidth Map scan through the 64 MB database split. These first three workloads use 50 GB of the 1998 World Cup website log [12]. *WordCount* and *SequenceCount* each find the frequency of words or sequences of words in large HTML files, and as input we use 50 GB of Wikipedia HTML data [7]. These last two workloads are more computationally intensive than the others because they involve text parsing and not just integer compares when sorting data.

6.3. Methodology

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

To simulate the CPU cores (OoO, EE, and NDC), we use the Simics full system simulator [8]. To simulate the DRAM, we use the USIMM DRAM simulator [18], which has been modified to model an HMC architecture. We assume that the DRAM core latency (Activate + Precharge + ColumnRead) is 40 ns. Our simulations model a single Map or Reduce thread at a time and we assume that throughput scales linearly as more cores are used. While NDCores have direct access to DRAM banks, EECores must navigate the memory controller and SerDes links on their way to the HMC device. Since these links are shared by 512 cores, it is important to correctly model contention at the memory controller. A 512-core Simics simulation is not tractable, so we use a trace-based version of the USIMM simulator. This stand-alone trace-based simulation models contention when the memory system is fed memory requests from 512 Mappers or 512 Reducers. These contention estimates are then fed into the detailed single-thread SIMICS simulation.

We wrote the code for the Mappers and Reducers of our five workloads in C, and then compiled them using GCC version 3.4.2 for the simulated architecture. The instruction mix of these workloads is strictly integer-based. 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.

We evaluate the power and energy consumed by our systems taking into account workload execution times, memory bandwidth, and processor core activity rates. We calculate power for the memory system as being equal to the the sum of the power used by each logic layer in each HMC, including SerDes links, the DRAM array background power, and power used to access the DRAM arrays for reads and writes. We assume that the four SerDes links consume a total of 5.78 W per HMC, and the remainder of the logic layer consumes 2.89 W [56]. Total maximum DRAM array power per HMC is assumed to be 4.7 W for 8 DRAM die [36]. We approximate background DRAM array power at 10% of this maximum value [6], or 0.47 W, and the remaining DRAM power is dependent on DRAM activity. Energy is consumed in the arrays on each access at the rate of an additional 3.7 pJ/bit (note that the HMC implements narrow rows and a close page policy [36]). 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 [39]. This is a conservative estimate because it ignores intermediate routing elements, and favors the EECore baseline. For the core power estimates, we assume that 25% of the 80 mW core peak power can be attributed to leakage (20 mW). The dynamic power for the core varies linearly between 30 mW and 60 mW, based on IPC (since many circuits are switching even during stall cycles).

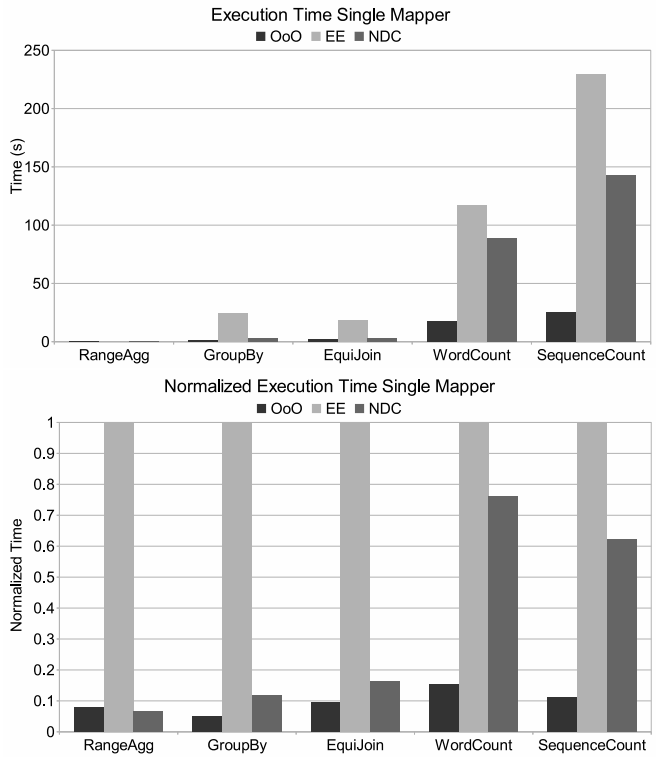

Figure 2: Execution times of a single Mapper task, measured in absolute time (top), and normalized to EE execution time (bottom).

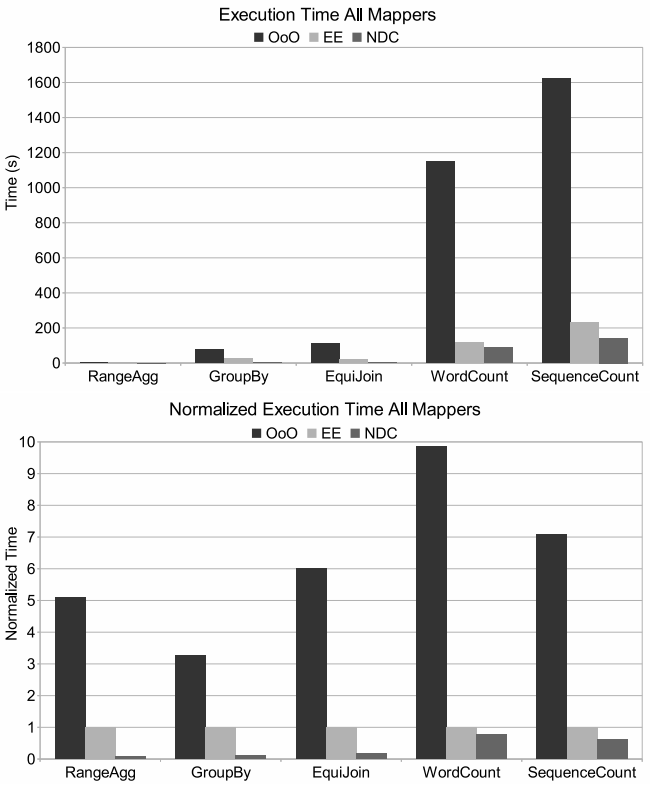

Figure 3: Execution times of all Mapper tasks, measured in absolute time (top), and normalized to EE execution time (bottom).

7. Performance Results

7.1. Individual Mapper Performance

We first examine the performance of a single thread working on a single input split in each architecture. Figure 2 shows the execution latency of a single mapper for each workload.

We show both normalized and absolute execution times to show the scale of each of these workloads. When executing on an EECore, a *RangeAgg* Mapper 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.

RangeAgg, *GroupBy*, and *EquiJoin* have lower compute requirements than *WordCount* and *SequenceCount*, so in these workloads, because of its memory latency advantage, an NDCore is able to nearly match the performance of an OoO core. The EECore system falls behind in executing a single Mapper task compared to both OoO and NDCores, because its HMC link bandwidth is maxed out for some workloads, as seen in Section 7.3.

7.2. Map Phase Performance

Map phase execution continues until all Mapper tasks have been completed. In the case of the EE and NDC systems, the number of Mapper tasks and processor cores is equal, so all Mapper tasks are executed in parallel, and the duration of the Map phase is equal to the time it takes to execute one Mapper task. In the case of the OoO system, Mapper tasks outnumber processor cores 64-to-1, so each OoO processor must sequentially execute 64 Mapper tasks.

Because of this, the single-threaded performance advantage of the OoO cores becomes irrelevant, and both EE and NDC systems are able to outperform the OoO system by a wide margin. As seen in Figure 3, compared to the OoO system, the EE system reduces Map phase execution times from 69.4% (*RangeAgg*), up to 89.8% (*WordCount*). The NDC system improves upon the EE system by further reducing execution times from 23.7% (*WordCount*), up to 93.2% (*RangeAgg*).

7.3. Bandwidth

The NDC system is able to improve upon the performance of the OoO and EE systems because it is not constrained by HMC link bandwidth during the Map phase. Figure 4 shows the read and write bandwidth for each 2-socket system, as well as a bar representing the maximum HMC link bandwidth, which sets an upper bound for the performance of the OoO and EE systems.

The OoO system is unable to ever come close to saturating the available bandwidth of an HMC-based memory system. The EE system is able to effectively use the large amounts of available bandwidth, but because the bandwidth is a limited resource, it puts a cap on the performance potential of the EE system. The NDC system is not constrained by HMC link bandwidth, and is able to use an effective bandwidth many times that of the other systems. While the two baseline systems are limited to a maximum read bandwidth of 320 GB/s, the NDC system has a maxi-

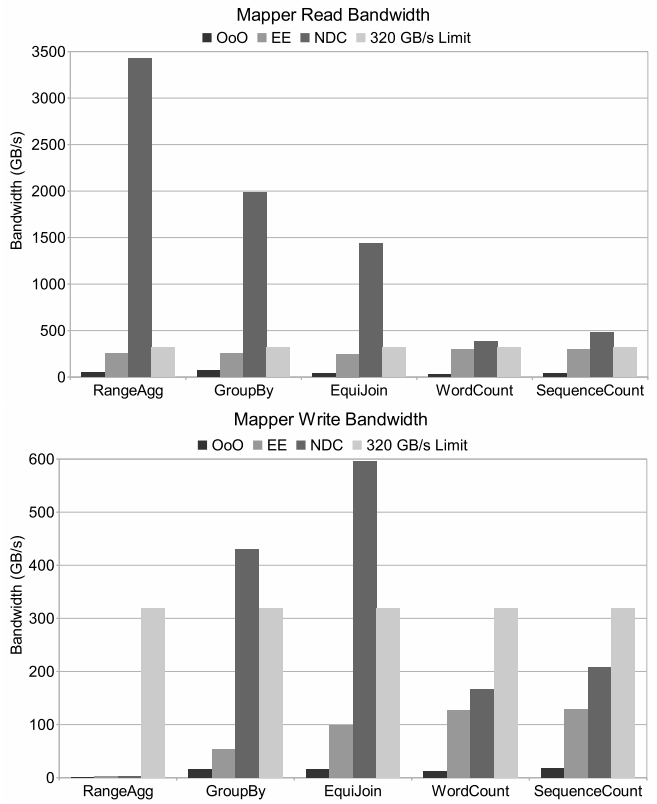

Figure 4: Bandwidth usage during Map phase for an entire 2-socket system. Maximum HMC link read and write bandwidth are each 320 GB/s for the system.

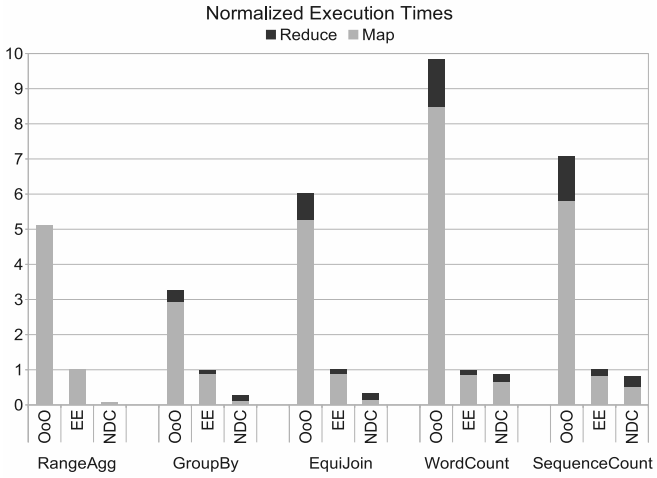

Figure 5: Execution time for an entire MapReduce job normalized to the EE system.

mum aggregate TSV bandwidth of 8 TB/s. In fact, this is the key attribute of the NDC architecture – as more memory devices are added to the daisy-chain, the bandwidth usable by NDCores increases. On the other hand, as more memory devices are added to the EE baseline, memory bandwidth into the two processor sockets is unchanged.

7.4. MapReduce Performance

So far we have focused on the Map phase of MapReduce workloads, because this phase typically dominates execution time, and represents the best opportunity for improving the overall execution time of MapReduce workloads. Figure 5 shows how execution time is split between Map and Reduce phases for each workload, and shows the relative execution times for each system.

The OoO and EE systems use the same processing cores for both Map and Reduce phases, but the NDC system uses NDCores for executing the Map phase, and EECores for executing the Reduce phase. Performance improves for both Map and Reduce phases when moving from the OoO system to the EE system, but only Map phase performance improves when moving from the EE system to the NDC system. Reduce phase performance degrades slightly for the NDC system since half the SerDes links are disabled (to save power).

Overall, compared to the OoO system, the EE system is able to reduce MapReduce execution time from 69.4% (*GroupBy*), up to 89.8% (*WordCount*). NDC further reduces MapReduce execution times compared to the EE system from 12.3% (*WordCount*), up to 93.2% (*RangeAgg*).

7.5. Energy Consumption

We consider both static and dynamic energy in evaluating the energy and power consumption of EE and NDC systems. Figure 6 shows the breakdown in energy consumed by the memory subsystem, and the processing cores. Figure 6a shows the energy savings when moving from an EE system to an NDC system that uses a full complement of HMC links (NDC FL). Compared to the EE system, the NDC FL system reduces energy consumed to complete an entire MapReduce task from 28.2% (*WordCount*), up to 92.9% (*RangeAgg*). The processor and memory energy savings primarily come from completing the tasks more quickly.

Figure 6b assumes NDC FL as a baseline and shows the effect of various power optimizations. NDC Half Links is the NDC system configuration we use in all of our other performance evaluations, and is able to reduce energy consumed by up to 23.1% (*RangeAgg*) compared to NDC Full Links. Disabling half the links reduces performance by up to 22.6% because it only affects the Reduce phase (as seen in the dark bars in Figure 5). NDC-PD is a model that uses all the SerDes links, but places unutilized cores in power-down modes. So NDCores are powered down during the Reduce phase and EECores are powered down during the Map phase. We assume that a transition to low-power state incurs a 1.0 ms latency and results in core power that is 10% of the core peak power. Note that the transition time is incurred only once for each workload and is a very small fraction of the workload execution time, which ranges between dozens of milliseconds to several minutes. This technique is able to reduce overall system energy by up to 10.0% (*SequenceCount*). Finally, combining the Half Links optimization with core power-down allows for energy savings of 14.7% (*GroupBy*) to 28.3% (*RangeAgg*).

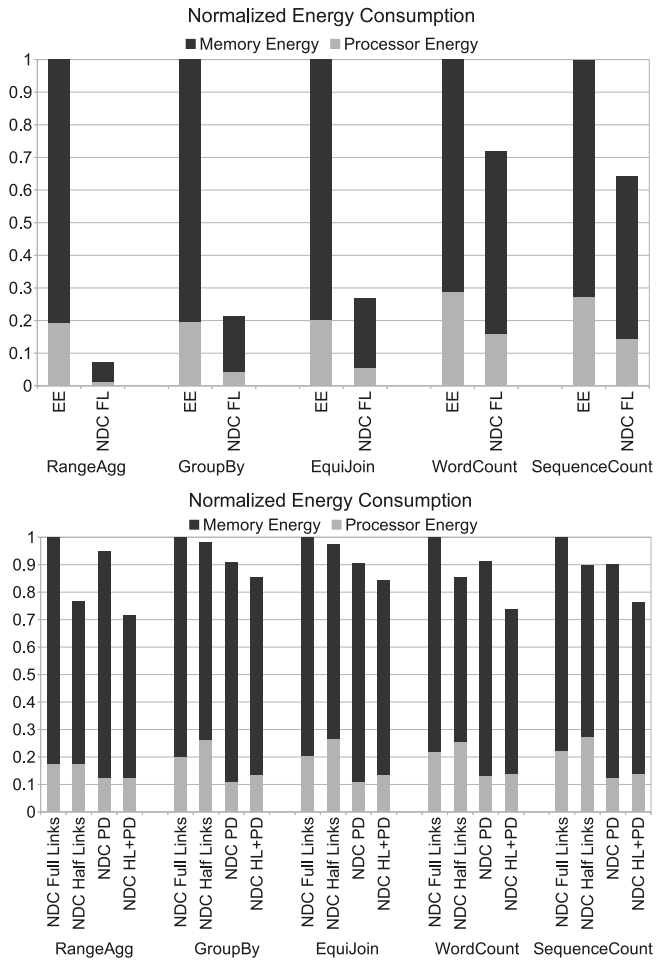

Figure 6: Energy consumed by the memory and processing resources. Top figure normalized to EE processor energy; bottom figure normalized to NDC FL processor energy.

7.6. HMC Power Consumption and Thermal Analysis

In addition to a system-level evaluation of energy consumption, we also consider the power consumption of an individual HMC device. In the EE system, the HMC device is comprised of a logic layer, including 4 SerDes links, and 8 vertically stacked DRAM dies. An NDC HMC also has a logic layer and 8 DRAM dies, but it only uses 2 SerDes links and also includes 16 NDC cores. As with the energy consumption evaluation, we consider core and DRAM activity levels in determining HMC device power. Figure 7 shows the contribution of HMC power from the logic layer, the DRAM arrays, and NDC cores, if present.

The baseline HMCs do not have any NDC cores, so they see no power contribution from that source, but they do have twice the number of SerDes links, which are the single largest consumer of power in the HMC device.

The NDC design saves some power by trading 2 SerDes links for 16 NDCores. However, we also see an increase in DRAM array power in NDC. In the EECore baseline, host processor pin bandwidth is shared between all HMCs in the chain, and no one HMC device is able to realize its full bandwidth potential. This leads to a low power contribution coming from DRAM array

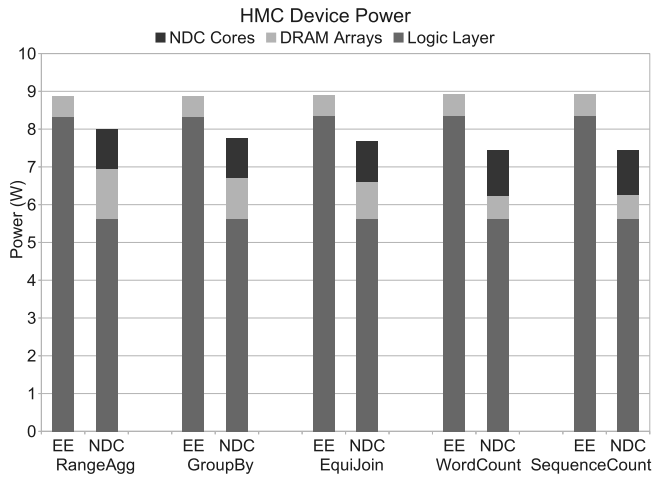

Figure 7: Breakdown of power consumed inside an HMC stack for both EE and NDC systems. The HMC in the EE system contains no NDC cores, and the HMC in the NDC system uses half the number of data links.

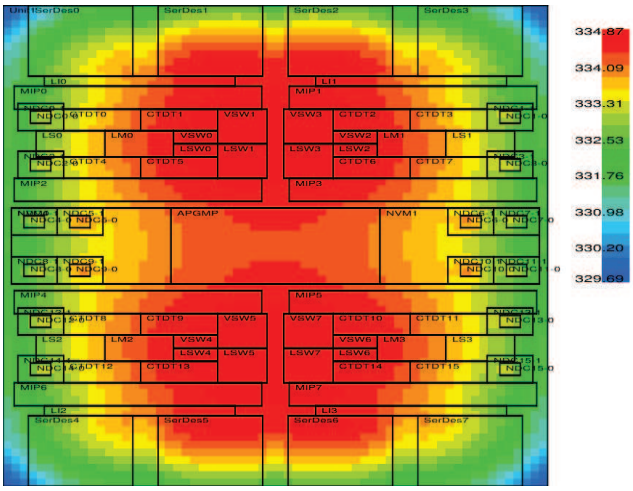

Figure 8: Heatmap of the logic layer in the NDC system (best viewed in color).

activity, because each HMC device can contribute on average only 1/8th the bandwidth supported by the SerDes links. The NDC architecture, on the other hand, is able to keep the DRAM arrays busier by utilizing the available TSV bandwidth. Overall, the NDC HMC device consumes up to 16.7% lower power than the baseline HMC device.

We also evaluated the baseline HMC and NDC floorplans with Hotspot 5.0 [2], using default configuration parameters, an ambient temperature of 45° C inside the system case, and a heat spreader of thickness 0.25 mm. We assumed that each DRAM layer dissipates 0.59 W, spread uniformly across its area. The logic layer's 8.67 W is distributed across various units based on HMC's power breakdown and floorplan reported by Sandhu [56]. We assumed that all 4 SerDes links were active. For each NDCore, we assumed that 80% of its 80 mW power is dissipated in 20% of its area to model a potential hotspot within the NDCore. Our analysis showed a negligible increase in device peak temperature from adding NDCores. This is shown by the logic layer heatmap in Figure 8; the SerDes units have

much higher power densities than the NDCore, so they continue to represent the hottest units on the logic chip. We carried out a detailed sensitivity study and observed that the NDCores emerge as hotspots only if they consume over 200 mW each. The DRAM layers exceed 85° C (requiring faster refresh) only if the heat spreader is thinner than 0.1 mm.

8. Conclusions

This paper argues that the concept of Near-Data Computing is worth re-visiting in light of various technological trends. We argue that the MapReduce framework is a good fit for NDC architectures. We present a high-level description of the NDC hardware and accompanying software architecture, which presents the programmer with a MapReduce-style programming model. We first construct an optimized baseline that uses daisy-chains of HMC devices and many energy-efficient cores on a traditional processor socket. This baseline pays a steep price for data movement. The move to NDC reduces the data movement cost and helps overcome the bandwidth wall. This helps reduce overall workload execution time by 12.3% to 93.2%. We also employ power-gating for cores and disable SerDes links in the NDC design. This ensures that the HMC devices consume less power than the baseline and further bring down the energy consumption. Further, we expect that NDC performance, power, energy, and cost will continue to improve as the daisy chains are made deeper.

Acknowledgments

We thank the anonymous reviewers for their many useful suggestions. This work was supported in part by NSF grant CNS-1302663 and IBM Research.

References

[1] "Cortex-A5 Processor," http://www.arm.com/products/processors/cortex-a/cortex-a5.php.
[2] "HotSpot 5.0," http://lava.cs.virginia.edu/HotSpot/index.htm.
[3] "Hybrid Memory Cube, Micron Technologies," http://www.micron.com/innovations/hmc.html.
[4] "Intel Xeon Processor E5-4650 Specifications," http://ark.intel.com/products/64622/.
[5] "Memcached: A Distributed Memory Object Caching System," http://memcached.org.
[6] "Micron System Power Calculator," http://www.micron.com/products/support/power-calc.
[7] "PUMA Benchmarks and dataset downloads," http://web.ics.purdue.edu/~fahmad/benchmarks/datasets.htm.
[8] "Wind River Simics Full System Simulator," http://www.windriver.com/products/simics/.
[9] "Open-Silicon and Micron Align to Deliver Next-Generation Memory Technology," 2011, http://www.open-silicon.com/news-events/press-releases/open-silicon-and-micron-align-to-deliver- next-generation-memory-technology.html.
[10] J. Adibi, T. Barrett, S. Bhatt, H. Chalupsky, J. Chame, and M. Hall, "Processing-in-Memory Technology for Knowledge Discovery Algorithms," in *Proceedings of DaMoN Workshop*, 2006.
[11] D. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan, "FAWN: A Fast Array of Wimpy Nodes," in *Proceedings of SOSP*, 2009.
[12] M. Arlitt and T. Jin, "1998 World Cup Web Site Access Logs," http://www.acm.org/sigcomm/ITA/, August 1998.
[13] O. Azizi, A. Mahesri, B. Lee, S. Patel, and M. Horowitz, "Energy-Performance Tradeoffs in Processor Architecture and Circuit Design: A Marginal Cost Analysis," in *Proceedings of ISCA*, 2010.
[14] BerkeleyDB, "Berkeley DB: high-performance embedded database for key/value data," http://www.oracle.com/technetwork/products/berkeleydb/overview/index.html.
[15] J. Brockman, S. Thoziyoor, S. Kuntz, and P. Kogge, "A Low Cost, Multithreaded Processing-in-Memory System," in *Proceedings of WMPI*, 2004.
[16] A. Caulfield, L. Grupp, and S. Swanson, "Gordon: Using Flash Memory to Build Fast, Power-efficient Clusters for Data-Intensive Applications," in *Proceedings of ASPLOS*, 2009.
[17] J. Chang, P. Ranganathan, D. Roberts, T. Mudge, M. Shah, and K. Lim, "A Limits Study of the Benefits from Nanostore-based Future Data Centric System Architectures," in *Proceedings of Computing Frontiers*, 2012.
[18] N. Chatterjee, R. Balasubramonian, M. Shevgoor, S. Pugsley, A. Udipi, A. Shafiee, K. Sudan, M. Awasthi, and Z. Chishti, "USIMM: the Utah SImulated Memory Module," University of Utah, Tech. Rep., 2012, UUCS-12-002.
[19] F. Checconi, F. Petrini, J. Willcock, A. Lumsdaine, A. Choudhury, and Y. Sabharwal, "Breaking the Speed and Scalability Barriers for Graph Exploration on Distributed-memory Machines," in *Proceedings of SC*, 2012.
[20] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Proceedings of OSDI*, 2004.
[21] M. deKruijf and K. Sankaralingam, "MapReduce for the Cell B.E. Architecture," *IBM Journal of Research and Development*, vol. 53(5), 2009.
[22] X. Dong, Y. Xie, N. Muralimanohar, and N. Jouppi, "Simple but Effective Heterogeneous Main Memory with On-Chip Memory Controller Support," in *Proceedings of SC*, 2010.
[23] Elpida Memory Inc., "News Release: Elpida Completes Development of Cu-TSV (Through Silicon Via) Multi-Layer 8-Gigabit DRAM," http://www.elpida.com/pdfs/pr/2009-08-27e.pdf, 2009.
[24] ——, "News Release: Elpida, PTI, and UMC Partner on 3D IC Integration Development for Advanced Technologies Including 28nm," http://www.elpida.com/en/news/2011/05-30.html, 2011.
[25] F. Färber, S. K. Cha, J. Primsch, C. Bornhövd, S. Sigg, and W. Lehner, "SAP HANA Database: Data Management for Modern Business Applications," *SIGMOD Record*, vol. 40, no. 4, pp. 45–51, 2011.
[26] F. Färber, N. May, W. Lehner, P. Große, I. Müller, H. Rauhe, and J. Dees, "The SAP HANA Database – An Architecture Overview," *IEEE Data Eng. Bull.*, vol. 35, no. 1, pp. 28–33, 2012.
[27] T. Farrell, "HMC Overview: A Revolutionary Approach to System Memory," 2012, exhibit at Supercomputing.
[28] D. Fick *et al.*, "Centip3De: A 3930 DMIPS/W Configurable Near-Threshold 3D Stacked System with 64 ARM Cortex-M3 Cores," in *Proceedings of ISSCC*, 2012.
[29] J. Gebis, S. Williams, C. Kozyrakis, and D. Patterson, "VIRAM-1: A Media-Oriented Vector Processor with Embedded DRAM," in *Proceedings of DAC*, 2004.
[30] Q. Guo, X. Guo, Y. Bai, and E. Ipek, "A Resistive TCAM Accelerator for Data-Intensive Computing," in *In Proceedings of MICRO*, 2011.
[31] Q. Guo, X. Guo, R. Patel, E. Ipek, and E. Friedman, "AC-DIMM: Associative Computing with STT-MRAM," in *Proceedings of ISCA*, 2013.
[32] M. Hall, P. Kogge, J. Koller, P. Diniz, J. Chame, J. Draper, J. LaCoss, J. Granacki, J. Brockman, A. Srivastava, W. Athas, V. Freeh, J. Shin, and J. Park, "Mapping Irregular Applications to DIVA, a PIM-based Data-Intensive Architecture," in *Proceedings of SC*, 1999.
[33] B. He, W. Fang, Q. Luo, N. Govindaraju, and T. Wang, "Mars: A MapReduce Framework on Graphics Processors," in *Proceedings of PACT*, 2008.
[34] J. Howard *et al.*, "A 48-Core IA-32 Message-Passing Processor with DVFS in 45nm CMOS," in *Proceedings of ISSCC*, 2010.
[35] ITRS, "International Technology Roadmap for Semiconductors, 2009 Edition."
[36] J. Jeddeloh and B. Keeth, "Hybrid Memory Cube – New DRAM Architecture Increases Density and Performance," in *Symposium on VLSI Technology*, 2012.
[37] X. Jiang, N. Madan, L. Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, Y. Solihin, and R. Balasubramonian, "CHOP: Adaptive Filter-Based DRAM Caching for CMP Server Platforms," in *Proceedings of HPCA*, 2010.
[38] Y. Kang, M. Huang, S. Yoo, Z. Ge, D. Keen, V. Lam, P. Pattnaik, and J. Torrellas, "FlexRAM: Toward an Advanced Intelligent Memory System," in *Proceedings of ICCD*, 1999.
[39] S. Keckler, "Life After Dennard and How I Learned to Love the Picojoule," Keynote at MICRO, 2011.
[40] D. Kim *et al.*, "3D-MAPS: 3D Massively Parallel Processor with Stacked Memory," in *Proceedings of ISSCC*, 2012.
[41] P. Kundu, "On-Die Interconnects for Next Generation CMPs," in *Workshop on On- and Off-Chip Interconnection Networks for Multicore Systems (OCIN)*, 2006.
[42] K. Lim *et al.*, "Understanding and Designing New Server Architectures for Emerging Warehouse-Computing Environments," in *Proceedings of ISCA*, 2008.
[43] K. Lim, D. Meisner, A. Saidi, P. Ranganathan, and T. Wenisch, "Thin Servers with Smart Pipes: Designing Accelerators for Memcached," in *Proceedings of ISCA*, 2013.

[44] G. Loh, "3D-Stacked Memory Architectures for Multi-Core Processors," in *Proceedings of ISCA*, 2008.
[45] G. Loi, B. Agrawal, N. Srivastava, S. Lin, T. Sherwood, and K. Banerjee, "A Thermally-Aware Performance Analysis of Vertically Integrated (3-D) Processor-Memory Hierarchy," in *Proceedings of DAC-43*, June 2006.
[46] N. Madan, L. Zhao, N. Muralimanohar, A. N. Udipi, R. Balasubramonian, R. Iyer, S. Makineni, and D. Newell, "Optimizing Communication and Capacity in a 3D Stacked Reconfigurable Cache Hierarchy," in *Proceedings of HPCA*, 2009.
[47] R. Murphy, P. Kogge, and A. Rodrigues, "The Characterization of Data Intensive Memory Workloads on Distributed PIM Systems," in *Proceedings of Workshop on Intelligent Memory Systems*, 2000.
[48] M. Oskin, F. Chong, and T. Sherwood, "Active Pages: A Model of Computation for Intelligent Memory," in *Proceedings of ISCA*, 1998.
[49] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazieres, S. Mitra, A. Narayanan, G. Parulkar, M. Rosenblum, S. Rumble, E. Stratmann, and R. Stutsman, "The Case for RAMClouds: Scalable High-Performance Storage Entirely in DRAM," *SIGOPS Operating Systems Review*, vol. 43(4), 2009.
[50] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and C. Yelick, "A Case for Intelligent DRAM: IRAM," *IEEE Micro*, vol. 17(2), April 1997.
[51] T. Pawlowski, "Hybrid Memory Cube (HMC)," in *HotChips*, 2011.
[52] P. Ranganathan, "From Microprocessors to Nanostores: Rethinking Data-Centric Systems," *IEEE Computer*, Jan 2011.
[53] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis, "Evaluating MapReduce for Multi-Core and Multiprocessor Systems," in *Proceedings of HPCA*, 2007.
[54] Samsung, "Samsung to Release 3D Memory Modules with 50% Greater Density," 2010, http://www.computerworld.com/s/article/9200278/Samsung_to_release_3D_memory_modules_with_50_greater_density.
[55] Samsung Electronics Corporation, "Samsung Electronics Develops World's First Eight-Die Multi-Chip Package for Multimedia Cell Phones," 2005, (Press release from `http://www.samsung.com`).
[56] G. Sandhu, "DRAM Scaling and Bandwidth Challenges," in *NSF Workshop on Emerging Technologies for Interconnects (WETI)*, 2012.
[57] SAP, "In-Memory Computing: SAP HANA," http://www.sap.com/solutions/technology/in-memory-computing-platform.
[58] SAS, "SAS In-Memory Analytics," http://www.sas.com/software/high-performance-analytics/in-memory-analytics/.
[59] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen, "A Novel Architecture of the 3D Stacked MRAM L2 Cache for CMPs," in *Proceedings of HPCA*, 2009.
[60] Tezzaron Semiconductor, "3D Stacked DRAM/Bi-STAR Overview," 2011, http://www.tezzaron.com/memory/Overview_3D_DRAM.htm.
[61] D. H. Woo *et al.*, "An Optimized 3D-Stacked Memory Architecture by Exploiting Excessive, High-Density TSV Bandwidth," in *Proceedings of HPCA*, 2010.
[62] M. Zaharia, M. Chowdhury, M. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," in *Proceedings of HotCloud*, 2010.