Performance Evaluation and Feasibility Study of Near-data Processing on DRAM Modules (DIMM-NDP) for Scientific Applications

Matthias Gries, Pau Cabré, Julio Gago

Performance Evaluation and Feasibility Study of Near-data Processing on DRAM Modules (DIMM-NDP) for Scientific Applications

Matthias Gries , Pau Cabré, Julio Gago

Abstract—As the performance of DRAM devices falls more and more behind computing capabilities, the limitations of the memory and power walls are imminent. We propose a practical Near-Data Processing (NDP) architecture DIMM-NDP for mitigating the effects of the memory wall in the nearer-term targeting server applications for scientific computing. DIMM-NDP exploits existing but unused DRAM bandwidth on memory modules (DIMMs) and takes advantage of a subset of the forthcoming JEDEC NVDIMM-P protocol in order to integrate application-specific, programmable functionality near memory. DIMM-NDP works on shared memory with the host CPU by definition, takes advantage of abundant memory capacity in the main memory subsystem and remains economic by relying on standard, unmodified DRAM devices.

We evaluate DIMM-NDP with a range of bandwidth, latency and compute-bound workloads from the domains of predictive data analytics and machine learning that depend on dense and sparse linear algebra. Simulation results show up to 6.3x better performance for bandwidth-limited applications, representing 79% of the theoretical peak, and up to 3x improved energy efficiency. We complement the evaluation with feasibility checks for DIMM-like form factors to offer 32GB to 128GB capacity per DIMM, hardware overhead costs (below 20%), and power envelopes for standard (13W) and custom DIMMs (40W). A sensitivity analysis of interface properties for comparison with traditional accelerator coupling over PCIe, as well as a case study on porting software kernels, showing in the order of one month programming effort per application, outline reasonable operating points for DIMM-NDP.

Index Terms—Processing In-Memory, near-memory accelerator, NUMA, software porting, performance engineering, case study.

CONTENTS

- *Matthias Gries is with Huawei Technologies Düsseldorf GmbH at the Munich Research Center (MRC), Riesstr. 25, 80992 München, Germany E-mail: gries@computer.org*
- *Pau Cabré and Julio Gago are with Metempsy, Pamplona 88, Barcelona 08018, Spain E-mail: {pau.cabre|julio.gago}@gmail.com*

Research Report released on April 15th, 2019

1 INTRODUCTION

The implications of the memory and power walls on memory subsystems are imminent [1], [2]. Operations have to move closer to data in the memory hierarchy in order to reduce redundant data movement as one of the major sources of wastage [3], [4], [5]. As a result of decades of scaling DRAM devices for memory capacity, the memory subsystem has been adapted to keep pace in terms of bandwidth, as shown in Fig. 1 for the *stream* [6] benchmark. "Brute force" effects, such as adding more memory channels, are nearly exploited and challenged by signal integrity, pin limitations and process technology scaling, as the design of analog frontends for IO interfaces becomes more complex [7]. Recent approaches to mitigate the memory wall

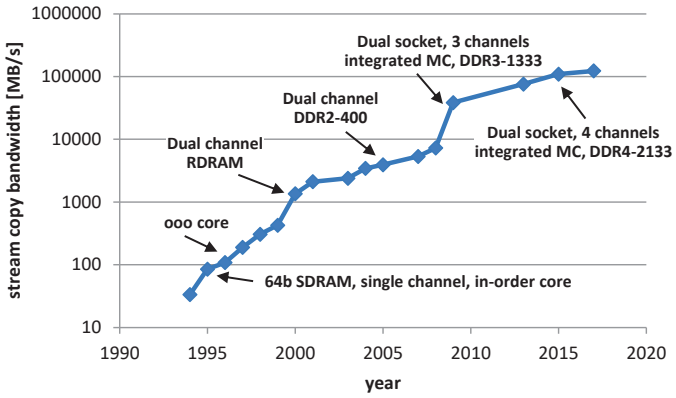


Fig. 1. Reported Stream [6] bandwidth on system-level, annotated with architecture modifications to sustain memory performance.

follow either *in-memory* (on the same technology process, such as [8] and references therein) or *near-memory* (e.g., 3D stacked, as in [9], [10], [11], [12]) ideas. These represent either longer-term research, since the economic manufacturing at high density still has to be shown, or relatively costly solutions with stacked DRAM, like High-Bandwidth Memory (HBM) and the Hybrid Memory Cube (HMC), that are limited by memory capacity and more costly to integrate than standard DRAM devices.

However, the effects of dark silicon [13] lead designers to more heterogeneous architectures. Programmers become more experienced with using accelerators and non-uniform memory in return. We can rely on this trend to introduce an acceptable solution by building on the concept of NUMA (non-uniform memory accesses) to associate memory locations with near-data accelerators.

1.1 Limited Adaptation of Near-Data Processing so far

The lack of adapting Near-Data Processing (NDP) in- or near-memory may be attributed to the following challenges:

- The integration of processing has been tried on the same technology process with the memory, either Non-Volatile Memory (NVM) or DRAM. In case of DRAM, this leads to either not scalable and slow, or not very complex designs, whereas the area of NVM integration and manufacturing is an open field of research.
- High-performance solutions (e.g., HBM, HMC) require stacking of logic and DRAM dies at higher costs and limited capacity, may depend on 2.5D integration with additional interposers and are subject to further thermal stress and testing challenges.
- Lacking standards in the past for interface protocols and programming, such that NDP has been perceived as an application-specific and proprietary solution.

1.2 Motivation for DIMM-NDP

The main features of DIMM-NDP are:

- Building on standard IP: NDP units enhance the Media Controller (MedC) on a memory module. The MedC is a discrete buffer chip positioned side-by-side the DRAM devices on the module and needed for forthcoming interface standards like JEDEC NVDIMM-P [14], [15], Gen-Z [16] and CCIX [17].
- DIMM-NDP employs unmodified DRAM chips and exploits unused rank-level bandwidth on DIMM, such that we follow the economy of scale of manufacturing standard DRAM.
- Memory capacity is not wasted. We focus on NDP in the shared memory subsystem of the host, where both sides (host and NDP) have concurrent access to memory. The memory module appears as normal Load-Reduced DIMM if NDP is switched off.
- Versatility: We use standard cores with special functional units as programmable NDP units that different application domains can take advantage of, as expected for a server setup.
- We take advantage of recent standard protocols to ease the implementation, namely forthcoming JEDEC NVDIMM-P [14], [15] that introduces asynchronous transfers as a subset, as well as NUMA and math libraries like BLAS [18] for programming.

By relying on standard DRAM, modular DIMMs and common programming abstractions, we see DIMM-NDP as an enabler for gradually introducing NDP into general-purpose servers for use with many applications that benefit from CPU-centric programming and high memory capacity. We lower the bar for adapting Near-Data Processing (NDP) in the main memory subsystem, since DIMM-NDP does not waste DRAM capacity. Application programmers can thus gradually migrate memory-bound code to DIMM-NDP by working on shared memory.

The contributions of this paper are a) the proposal of the DIMM-NDP architecture and host-CPU centric programming view amenable to scientific computing b) the performance evaluation of DIMM-NDP, c) a case study of the usability looking at the software porting effort, and d) revealing design tradeoffs in terms of form factor, performance, power and costs with feasibility checks.

After surveying related work in the next section, sections 3 and 4 introduce the architecture and software stack for DIMM-NDP. Section 5 shows feasibility checks for form factors, power and overhead costs. We describe the evaluation flow, performance results and the software porting efforts in sections 6 and 7, followed by concluding remarks.

2 RELATED WORK

Near-Data Processing (NDP) may happen *in-memory* on the same technology process for DRAM or Non-Volatile

Memory, or *near-memory*, where processing and memory are implemented on different process technologies, also see the surveys [1], [2]. We focus on the latter case, since in-memory variants require further research on manufacturing before widespread adaptation.

2.1 Techniques for Synchronizing Host and NDP

Solutions range from a) hardware-supported coherence [19], [20] enabling many styles of synchronization, b) using uncached memory regions [10], [11], [21] for exchanging information and separate address spaces for the actual computations (incurring overhead for copying data), and c) software-managed cache maintenance [9], [22] on shared memory regions for synchronization points defined by software. The hardware architecture may impose a handover mechanism, such that either the host or the NDP side work on the shared address range exclusively [21]. Contrarily to these approaches, in [23], a network stack is mapped onto the memory organization, such that the system appears as IP network-connected, distributed system for use with message passing (MPI).

Our solution allows concurrent accesses from host and NDP sides for shared address ranges. This enables many synchronization methods in software and is less limiting than, e.g., [21], [24], where only one side (host or NDP) can be in control of the memory at a time.

2.2 Options for Integration of NDP Technology

Recent focus has mainly been put on Through Silicon Via (TSV)-enabled heterogeneous die stacking [9], [10], [11], [12], [21], [25], where a logic die can be placed underneath a DRAM solution. Alternatively, logic can be put on memory modules next to DRAM devices, for instance, in distributed data buffer chips [24]. Finally, NDP may happen closer to the memory controllers of the main SoC (like in [26]), which is similar to a traditional on-die accelerator with a shortened path to main memory.

We favor lower costs, higher memory capacity and a modular solution over peak performance by relying on standard DRAM devices, modules and protocols, whereas solutions like [19], [24] modify DRAM interface logic and the protocol to the host respectively. High-bandwidth 3D-stacks are limited by memory capacity and thus not suitable for flat main memory of most server systems.

2.3 Variants of NDP Functionality

The functionality of NDP may vary from a set of fixed functions [22], [27] (including data reorganization [28], [29]), reconfigurable grid of functional units [11], [21], ASIPs (like LIW units [12]), to general-purpose, enhanced lightweight cores with a reduced memory hierarchy [9], [10], [25], [30].

Our solution stays with the access granularity of a rank (64B for DDR4), i.e., not with smaller device-level granularity as in [21], [24], [26]. As a result, we can share memory regions and do not have to copy and transcode data between host and NDP to a finer degree of parallelism. We can reach the same level of usable bandwidth, as long as 64B accesses are exploited, which we support by standard cores with L1 caches as NDP units that programmers are familiar with.

2.4 Different Form Factors for Memory Modules

NDP has been proposed for Buffers-on-Board [19] (BOB) where a proprietary protocol can be applied between the host and the BOBs, DIMMs complying to standard form factors [21], [24], DIMM interposers [22] and more accelerator-like deployments [26].

A standard DIMM carrying high-density DRAM stacks and a NVDIMM-P media controller (without NDP functionality) is described in [31]. Former DRAM buffer designs targeting standard modules include JEDEC DDR3 MB [32] and JEDEC AMB [33]).

2.5 Positioning DIMM-NDP

Our solution offers higher bandwidth than [22], [24], as we re-organize traditional ranks on DIMM into independent channels. This is at the expense of board-level routing (i.e., additional PCB layers), which we think is feasible for accelerator-like deployments (comparable with, e.g., GPUs and FPGA setups) since a) we share memory with the host and pay for memory only once, and b) avoid more costly silicon-interposer solutions required for high-bandwidth memory (see cost analysis in subsection 5.4).

We thus offer a gradual migration and upgrade path for server-class applications and their ecosystem that depend on high memory capacity, at a manageable degree of parallelism defined by the number of DIMMs and NDP-enabled memory ranks.

3 DIMM-NDP Architecture

DIMM-NDP increases available bandwidth for processing by implementing NDP units locally on memory modules (DIMMs). Media Controllers (MedC), as required for forthcoming standards like NVDIMM-P [14], [15] and Gen-Z [16], are complemented with NDP units, such that DIMMs can operate independently. In addition, memory ranks on DIMM may be reorganized as independent channels on DIMM (i.e., laid out as independent channels), as visualized in Fig. 2. The required routing can be solved on PCB-level with additional routing layers, as we still rely on the ball pitch of standard DRAM devices.

3.1 Architecture Overview

An overview of the building blocks involved is shown in Fig. 3. In the host System-on-Chip (SoC), the Memory Controller (MC) is enhanced with a memory-mapped control and status register interface to access DIMM-NDP instances. This register interface is a proxy to the more complex register interface implemented in the Media Controller (MedC).

The MedC placed on DIMM provides a fast path to DRAM if NDP units are switched off. The host's MC employs a plain DDR4/DDR5 protocol to access main memory in this case. The fast path is configured on a rank-by-rank basis (ranks seen by the host that may be laid out as independent channels on DIMM). That means, selected ranks on the host channel may use the NVDIMM-P protocol [14], [15], while all others employ plain DDR4/5.

If NDP units are enabled, the host's memory bus employs JEDEC NVDIMM-P asynchronous accesses to handle variable latency for requests issued by the host to the

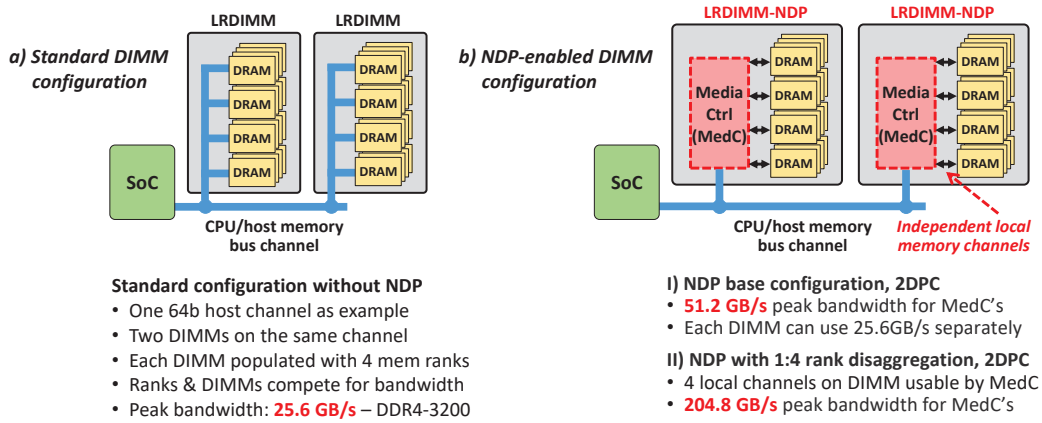


Fig. 2. Overview of DIMM-NDP memory organization and advantage with DIMM-local channels, 2 DIMMs per host channel.

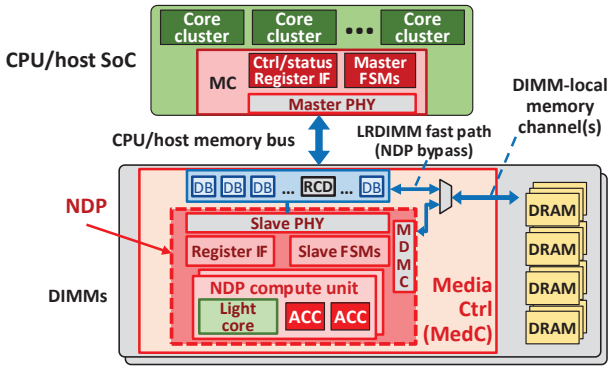


Fig. 3. Overview of DIMM-NDP architecture utilizing shared main memory with the host SoC.

corresponding ranks. Asynchronous accesses do not depend on DRAM timing nor DRAM organization (banks, pages and so on). Requests from the host are thus re-queued at the MedC. The MedC employs its own memory controller (MDMC) to schedule accesses from NDP units and accesses from the host according to DDR4/5 timing. Since memory is shared between the host and the NDP units, they compete for memory bandwidth whenever accesses occur concurrently. MDMC's scheduling policy resolves conflicts between the host and NDP units by priorities and DRAM timing parameters.

3.2 NDP Unit Setup

For versatility and energy efficiency, we consider in-order cores as programmable NDP units (similar to ARM's Cortex-A55 core) that can be enhanced with functional units for vector and matrix processing (accelerators [ACC]). We assume that the register file for the vector unit can be reused by the matrix unit. In our evaluation setup described in section 6, the ARMv8-A [34] core for NDP uses Scalable Vector Extensions (SVE, [35]) and the specific organization of the matrix unit from [36].

An NDP unit employs a reduced memory hierarchy of one level of caches plus an optional scratchpad memory in parallel to the L1 caches. We associate one NDP unit with each local memory channel. An NDP unit may access neighboring channels on the same DIMM at slightly higher access latency over the on-die hierarchical interconnect of the MedC chip.

3.3 Lateral Data Transfers between NDP Units

Lateral memory requests between the memory ranks and channels on the same DIMM are quick, since the MedC buffer includes the corresponding NDP units and local memory controllers (MDMCs in Fig. 3). Data requests issued by one NDP unit to other DIMMs and channels of the host can be routed through the memory controllers of the host SoC, but will be slower than local accesses. Forthcoming NVDIMM-P [14] will provide signaling channels to notify the host of pending events and actions at the DIMM side. NDP-data can be kept local on the same DIMM by, for instance, employing a NUMA-aware memory allocator to reduce the need for lateral transfers.

3.4 ECC Handling

If the NVDIMM-P [15] protocol is switched on, we assume that the Error-Correcting Code (ECC) processing is done at the media controller on DIMM, as many different types of DRAM and non-volatile memory may be used on DIMM with NVDIMM-P, requiring custom ECC methods. The host can have access to ECC event statistics that happen on the DIMM, such that this information can be used in a higher-level RAS policy (Reliability, Availability and Serviceability). If NDP is switched on, the MDMC instance (see Fig. 3) is responsible for ECC processing in our case.

4 Software View of DIMM-NDP

By building on the currently most pervasive JEDEC memory interface, hardware coherency techniques are not available. For using accelerators, this is manageable in software since programmers of scientific applications are familiar with synchronizing activity on CPUs and accelerators, like FPGAs and GPUs.

4.1 Workload Partitioning and Data Placement

By choosing coarser-grained programmable cores as NDP units, the degree of parallelism remains manageable. The number of cores in the host CPU and the number of NDP

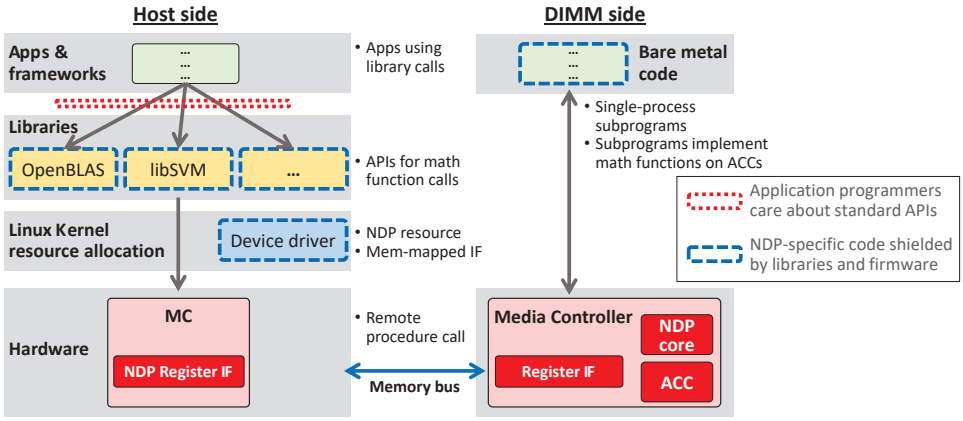


Fig. 4. DIMM-NDP programming abstraction on host and NDP sides and host-centric programming flow.

units are similar, such that, e.g., a pragmatic 1:1 association of host cores and NDP units for offloading may be applied. Data are partitioned by the number of NDP-enabled DIMMs. This is similar to using a number of PCIe-attached accelerators and thus a well-known workable constraint for programmers of scientific applications.

The MedC buffer is in control of several local ranks on DIMM, such that an NDP unit co-located at one rank can access data on the other ranks at the same MedC. Transfers between DIMMs must go over the host memory controller (MC) by signaling events over NVDIMM-P. Programmers can influence the frequency of remote transfers by exploiting a NUMA memory allocator on the host to keep the data local, and by means like data replication, exploiting the capacity of main memory.

4.2 Protection/Virtual Address Spaces

The infrastructure for address space protection and memory management of a standard core for use with NDP remains important for limiting the scope of the NDP unit. The assumption is that the process on the host that reserved the NDP unit is also responsible for configuring the unit, such that the unit runs in the same address space as the host process. The scope of the NDP unit can be reduced by only taking over a subset of the address translation tables of the host process to the NDP side. Alternatively, the NDP core may use a reduced translation table, e.g. by employing large pages or segments as done in, e.g., [37].

Translation entries can only be changed by a process on the host that reserved the NDP resource in the first place. The NUMA memory allocator for NDP is implemented on the host, such that data structures can be set up before handing over execution to the NDP side to avoid frequent updates of the translation tables at the NDP side.

4.3 Synchronization

With DIMM-NDP, programmers may use the host as the master that triggers activity on NDP units explicitly over their register interface. Alternatively, synchronization may happen over shared memory regions in a more distributed fashion. We thus expect synchronization points defined by software, such that the memory subsystem will be in a well-defined state by employing explicit cache maintenance operations on the host and NDP sides.

4.4 Software Implementation Flow

An NDP unit runs bare-metal in a first revision for scientific workloads and thus supports only one application context. We do not consider context switches on the NDP side, i.e., an NDP unit can only be reserved by one process on the host. We interpret "bare-metal" as bare machine computing as, e.g., applied in [38], where a self-contained Application Object includes a minimal single-execution operating environment. For instance, the NDP bare-metal code does not cover exception handling. Exceptions are handed over to the host process, either by signaling over the NVDIMM-P backchannel and/or shared memory. In our NDP context, the setup of the NDP execution environment is configured by the host process that allocated the NDP unit as a resource.

Overall, migrating code to DIMM-NDP becomes straightforward, particularly if the same instruction set is supported on the host and NDP.

A functionally correct implementation can be derived quickly with bare-metal compiled code on the NDP units. Effort can then be put on iterative refinements for performance tuning (see study in subsection 7.4), e.g., along vectorization and using scratchpad memory. Fig. 4 provides an overview of a first release of a software setup.

Programming complexity is shielded by established math libraries that the programmer is used to. In order to reserve and configure NDP resources, we can take advantage of the known NUMA abstraction to, for instance, co-locate data and NDP units. Each NDP-enabled DIMM appears as a separate NUMA domain. The NUMA allocator is implemented on the host, following a CPU-centric programming model as the host has access to the complete capacity of main memory.

5 Hardware Setups for DIMM-NDP

We distinguish module options for DIMM-NDP that support 2 and 4 local memory channels on DIMM respectively. Feasibility checks are considered for JEDEC DDR4, as comprehensive information is available for DDR4, and DDR5 is not standardized yet. We analyze two form factor options,

TABLE 1
Characteristics of Compute and Memory Building Blocks Reported in Related Work

Block	Clock	Technology	Area	Power	Description
ARM Cortex-A53 core	1.8GHz	TSMC FF+ 16nm	0.5mm^2	200mW	[39], running Dhrystone
64b mem IO frontend	DDR4-3200	10nm	6 - 9mm^2	400mW	[40], [41] for power estimation, [42], [43], [44] for area[a)]
Scratchpad		16nm	1.5mm^2		[45], for 2MB capacity
Vector unit	1GHz	40nm	4mm^2	<300mW	[46]: SP-FP data type, 1024b vectors
Vector unit	250MHz	LSI 28nm		30mW	[47]: 32b int, ≤1024b vector length
Vector unit	1GHz	28nm	3.6mm^2		[48]: DP-FP type, 4x128b lanes, 300KB caches
Matrix unit	700MHz	28nm	≤115mm^2	<40W	[49]: 8b int, 256x256 MAC array, 4MB SRAM

[a)] *Additional 10% to 20% area for memory controller*

power and chip area requirements, as well as overhead costs for the complete modules.

5.1 Power and Area Analysis of Building Blocks

Based on related work listed in Table 1, we project required area and power back-of-the-envelope. On a recent 10nm technology process, we estimate a feasible vector unit design for 512b vectors, including register file (16 to 32 vector registers), to take less than 1mm^2 and consume 100mW to 200mW at 1 to 1.5GHz clock range for use by the NDP unit. Scaling down the 256x256 matrix unit of the TPU [49] to the dimensions we consider for multiply-and-accumulate (MAC) operations as a special unit, still on 28nm, we need:

16 x 16 MAC: area 0.45mm^2, power: below 150mW
32 x 32 MAC: area 1.8mm^2, power: below 600mW

These are upper limits, since our design will use a more recent process technology, providing us headroom for a higher clock rate and/or support of different data types. We assume that the register file for the vector unit can be reused by the matrix unit.

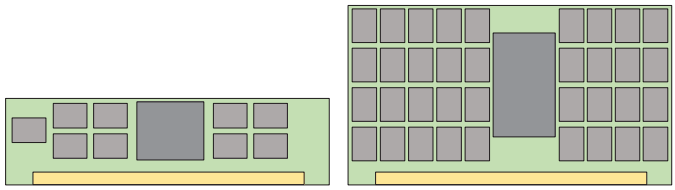
The area requirements for the MedC buffer chip on a 10nm-like process are summarized in Table 2 for the two DIMM scenarios of interest using two (standard DIMM) and 4 (custom DIMM) local memory channels for NDP respectively. The analog memory IO frontend blocks are the major contributing factor.

5.2 Chip Packages and Form Factors

The following analysis shows that the size of the MedC device is determined by the need for IO balls and not by the logic complexity of the buffer (see area evaluation in the preceding subsection for comparison).

5.2.1 DIMM Memory Module Form Factor and Power

The power envelope for a DDR4 DIMM is typically dimensioned for about 13W [50]. An LR-DIMM can hold up to 36 DRAM x4 devices, one RCD [51], and 9 DB [52] chips, employing both sides of the DIMM. An NDP-enabled MedC chip that supports two local DDR4 channels needs a 900-ball package estimated from prior art (Intel SMB [53], JEDEC DDR3 MB [32] and JEDEC AMB [33]). At a 0.8mm ball pitch as used by DRAM devices, such a package requires roughly 25mm x 25mm or a rectangular layout. This package fits on a standard DIMM with about 28mm usable height. A standard DIMM has enough room for 18 x8 DRAM devices, x8 DBs and the NDP-enhanced MedC, using both sides of the DIMM as sketched in Fig. 5 a).



Example a) horizontal placement, x8 devices, 32GB, DIMM form factor, 2 local channels, 2Tops, 13W

Example b) vertical placement, x4 devices, 128GB, custom DIMM height, 4 local channels, 16Tops, 40W

Fig. 5. DIMM-NDP modules: a) standard, b) custom DIMM form factors.

5.2.2 Accelerator Form Factor (Custom DIMM) and Power

A custom form factor allows higher design power, e.g., 40W, but needs additional power lines and cooling. This is the option for highest performance and capacity. Example: 4 channels with x4 DRAM need 72 DRAM devices, requiring roughly double the regular DIMM height using both PCB sides, see Fig. 5 b). Alternatively, 4 channels with x8 DRAM need 36 DRAM devices that can be arranged in 3 rows, such that only about 10mm extra height is needed. The MedC chip package roughly needs a 1500-ball package as estimated from prior art (e.g., IBM Power8 buffer [54]). The package takes 32mm x 32mm area at 0.8mm ball pitch and 26mm x 26mm area at 0.65mm pitch respectively.

5.3 Resulting NDP-enabled Memory Modules

The following NDP-enabled modules use the estimations from the previous subsections. DRAM device power is derived from Micron's DDR4 device power models [41] at 100% utilization with 2/3 reads, 1/3 writes (no power down cycles), open-page mode, 50% hit rate, 50% bit toggle rate.

5.3.1 NDP on Regular DIMM within 13W at 32GB

In this setup, the MedC embeds two in-order cores with their vector and matrix units to support two local DRAM channels at 51GB/s peak bandwidth for DDR4-3200. With 16Gb DRAM x8 devices, the total capacity of the DIMM is 32GB. The power envelope stays within 13W based on the conservative estimation in Table 3.

TABLE 2
Area Estimation of NDP Building Blocks on 10nm-like Technology Process for MedC Buffer Chip

Block	Area	Standard DIMM Quantity	Standard DIMM Total Area	Custom DIMM Quantity	Custom DIMM Total Area
In-order core with L1 caches	$1mm^2$	2	$2mm^2$	4	$4mm^2$
Vector unit, 512b	$1mm^2$	2	$2mm^2$	4	$4mm^2$
Matrix unit (16x16 | 32x32)	$0.5mm^2$ | $2mm^2$	2	$1mm^2$	4	$8mm^2$
Glue logic (MDMC, buses, etc.)	$4mm^2$	2	$8mm^2$	4	$16mm^2$
Memory IO frontend	$9mm^2$	3	$27mm^2$	5	$45mm^2$
Optional scratchpad	$1.5mm^2$	2	$3mm^2$	4	$6mm^2$
NVDIMM-P asynchronous accesses	$2mm^2$	1	$2mm^2$	1	$2mm^2$
Estimated MedC chip area (sum)			*<45mm²*		*<85mm²*

TABLE 3
Power Analysis of NDP on standard and custom DIMMs, logic at 2GHz on 10nm-like Technology Process

Block	Power	Standard DIMM Quantity	Standard DIMM Total	Custom DIMM Quantity	Custom DIMM Total	Note
In-order core	0.3W	2	0.6W	4	1.2W	
Vector unit, 512b	0.3W	2	0.6W	4	1.2W	
Matrix unit, 16 x 16	0.3W	2	0.6W			2Tops total (MAC counting as 2 ops)
Matrix unit, 32 x 32	1.2W			4	4.8W	16Tops total (MAC counting as 2 ops)
Glue logic	2W	1	2W	2	4W	MDMCs, on-die buses, FSMs, etc.
Mem IO frontend	0.4W	3	1.2W	5	2W	local channels and interface to host
x8 DRAM device	0.4W	18	7.2W			including ECC, DDR4-3200
x4 DRAM device	0.32W			72	23W	including ECC, DDR4-3200
Discrete DBs (if needed)	0.8W	1	0.8W	1	0.8W	mainly 2x memory IO frontend
Estimated DIMM total power (sum)			*13W*		*37W*	

5.3.2 NDP Accelerator Form Factor $\leq$ 40W, 128GB

The MedC embeds 4 cores, each with one 32x32 MAC array and one vector unit, to support 4 local DRAM channels at 102GB/s bandwidth for DDR4-3200. With 16Gb x4 devices, we reach 128GB capacity, staying below 40W (Table 3).

5.4 Cost Analysis for Memory Module

Cost factors vary depending on specific supply chains, i.e., the following should be seen as indicative only. Assuming a volume of 1 million DIMMs, non-recurring engineering (NRE) costs can be neglected for the most part. This corresponds to about 60k fully populated sockets, which we consider as low volume and realistic to achieve. We recognize the following factors that limit additional costs to less than 20% by switching from LRDIMM to DIMM-NDP:

- PCB layout with additional routing layers: Overhead is less than $10 per board (see [55] for an overview of PCB cost factors).
- MedC buffer chip: Additional costs are below $30 since the buffer has a similar complexity as a smartphone chipset [56].
- DRAM costs: Stay the same. At $7/GB[1], costs for 128GB plus ECC are about $1000.
- Cooling and power distribution: the 128GB DIMM-NDP can still be cooled by air flow (16 boards, 40W each, equal two "heavy" PCIe boards); overhead of a few dollars per board.

1. DRAM spot price, as of Dec. 2018 tracked at DRAMeXchange: https://dramexchange.com/

DRAM costs remain the dominating factor of the DIMM. In comparison, solutions based on HBM and HMC are up to one order of magnitude more expensive than DDR4 at the same capacity due to TSV processing steps and TSV area for heterogeneous stacking, testing and package costs [57]. [58] suggests a price premium of about 3x for HBM2 over GDDR5. In addition, we have to factor in the costs for custom logic for each stack to support NDP units in a heterogeneous 3D stack or have to consider a wide memory interface in the media controller. Silicon interposer costs for integrating HBM are at the same level as the costs for the buffer chip stated above due to the required area.

6 Evaluation Setup and Flow

6.1 Selected Workloads

In order to characterize DIMM-NDP, we complement low-level benchmarks for memory bandwidth and latency with full applications from the domain of data analytics, scientific computing and machine learning as representatives of a wider range of compute methods.

6.1.1 Elementary Tests

- *stream* [6] stresses memory bandwidth for 4 different matrix compute kernels.
- Pointer chasing, as in *lmbench* [60], reveals round-trip memory latency under low load.

TABLE 4
Characteristics of Selected Data Sets for liblinear and libSVM Applications

Name	Type	Classes	Samples	Features	Features/ sample [avg.]	Use case
News20.binary	sparse	2	20K	1.4M	462	text classification in newsgroups
Avazu	sparse	2	40M	1M	15	click-through prediction for ads
Protein	sparse	3	18K	357	102	protein structure prediction
e2006-tfidf	sparse	8	16K	150K	1218	financial risk assessment
YearPredictMSD	dense	86	463K	90	90	year of song prediction by audio features

TABLE 5
Host SoC Configuration as Reference

Parameter	Description
Core count	up to 48 ARMv8-A [34] out-of-order
Core type	ARM Cortex A76-like [59] (decode width 4, 16 SP-FLOPS/cycle), AArch64 state
Core frequency	3GHz
HW prefetcher	stride prefetcher on I$ (degree 1), D$ (degree 4) and L2 (degree 8)
L1 cache	64KB I$ and 64KB D$ per core
L2 cache	512KB per core, private
Last Level Cache	up to 48MB exclusive with L2$'s
Main memory	8 channels at DDR4-3200
MC memory control	FR-FCFS (first row hit – first come first served); open_adaptive open-page mode
Interconnect	2 bi-directional rings, 16B/clock per direction
PCIe IO	32 available PCIe gen4 lanes for accelerators

6.1.2 Applications

Kaggle's survey on the state of data science [61] reveals methods like logistic regression, neural networks and decision trees in the top-5 of the most popular techniques. We choose representative workloads from this set that span memory and compute-bound programs.

- *liblinear* [62] for solving linear classification and regression. The solvers rely on Level-1 vector Basic Linear Algebra Subprograms (L1 BLAS) and are memory bandwidth-bound.
- *libSVM* [63] for solving support vector classification, regression, and multi-class classification. The solvers rely on L1 BLAS vector operations and are memory bandwidth-bound.
- *SSD* [64] (Single Shot MultiBox Detector) applies convolutional networks for the detection of multiple objects in images. SSD relies on Level-3 BLAS matrix operations and is more compute than memory-bound.
- *xgboost* [65] addresses classification and regression problems with gradient boosting by employing decision trees, relying on tree traversals, and is memory latency-bound.

6.1.3 Data Sets

The data sets in Table 4 are selected for *liblinear* and *libSVM* and listed at the libSVM page ([63], from various sources). These represent different set sizes and feature sets. The number of samples are shown for training. Additional 10% to 20% are usually available for testing.

For *SSD*, we apply the SSD300 model with the Pascal VOC07+12 dataset referenced at the SSD webpage [64] for training the object class recognition in images.

For *xgboost*, we use the kaggle-higgs competition for identifying Higgs bosons (250k events with 30 features) and the YearPredictMSD data sets. Both are referenced in the xgboost distribution [65].

6.2 System Configuration

6.2.1 Host SoC

Inspired by recent high core-count CPUs like Qualcomm Centriq (48 cores), Marvell ThunderX2 (32 cores), AMD EPYC (32 cores) and Intel Xeon Scalable Processor (28 cores) and their memory subsystems, we consider the reference system in Table 5 as host SoC.

6.2.2 NDP Unit

The parameters for one NDP unit are summarized in Table 6, including vector, matrix and scratchpad support. Considering up to 2 DIMMs per host memory channel and four ranks per DIMM, we can have up to 64 NDP units per host SoC socket.

6.2.3 Mapping from System-physical to DIMM/DRAM-physical Addresses by the Host

As in related work [19], [67], address interleave between DRAM banks, ranks and channels on cacheline granularity on the host memory bus has to be adapted to keep data local to NDP units. In our case, fine-grain address interleave between DIMMs and host channels should not be applied for NDP-enabled memory ranks. The MedC buffer on DIMM may apply any address interleave scheme for the ranks and banks under its control to reach higher utilization.

TABLE 6
Configuration for one NDP Unit

Parameter	Description
Core type	ARMv8-A [34] in-order, ARM Cortex A55-like pipeline, AArch64 state
Core frequency	2GHz
Vector support	SVE512-like [35]
Matrix support	16 x 16 single-precision Floating Point MAC array [36]
Memory	single channel at DDR4-3200
MDMC mem control	FR-FCFS (first row hit – first come first served); open_adaptive open-page mode
L1 cache	32KB I$ and 64KB D$ per core
Scratchpad	optional 2MB

TABLE 7
Energy per Elementary Operation on 10nm-like Process Extrapolated from [4], [5], [40], [66]

Operation	Host: High-freq process [pJ]	NDP: Low voltage process [pJ]	Description
Exec, 1 instruction	110 (OoO)	13.1 (InOrder)	complexity of control logic, I$
Idle	40 (OoO)	10 (InOrder)	factoring in IPC, CLK, leakage
Integer simple / complex	0.1 / 3	0.07 / 2.2	Int Add / Int Mul
FP simple / complex	0.9 / 3.7	0.7 / 2.8	FP Add / FP Mul
SIMD Int simple / complex	0.2 / 6	0.15 / 4.5	128b width
SIMD FP simple / complex	1.8 / 7.4	1.3 / 5.5	128b width
SVE vector instruction		(4x)	512b: 4x the value of 128b SIMD instr
D$ read	20	17	64b value
L2$ read	170		512b, includes wiring
LLC read	400		512b, includes wiring, on-die buses
DRAM access (DIMM)	3 to 10 pJ/bit typ.		depending on operating point[a)]
DRAM access (IO)	1.1 to 1.7 pJ/bit		for address, cmd, data lines[a)]
DRAM background power	93 mW for each x8 device		refresh overhead & active standby [41]

[a)] *DRAM read & write percentage, DRAM page hit & further statistics derived from simulation in Gem5 and used with the DDR4 Micron power model [41] for x8 devices at DDR4-3200*

6.3 Evaluation Flow

6.3.1 Performance Evaluation

The Gem5 simulator [68] is used as framework in system emulation mode. We added the following enhancements to model the features described in subsection 6.2:

- The out-of-order core model is calibrated to match the performance level of a modern core in terms of issue width, internal bandwidth, queue depths and functional units
- We derived the NDP unit model from a Gem5 branch that models vector instructions (SVE [35]) and added SVE gather/scatter support
- NVDIMM-P-like asynchronous communication between the host and NDP core systems is modeled by a bridge block between the two subsystems
- We implement the matrix unit as additional functional unit that reassigns selected SVE command encodings for use with the unit (a real implementation requires new instructions)
- We model the impact of cache maintenance by ARM cache-clean & invalidate operations on virtual addresses (thus covering all cache levels on the host) on the host and the NDP sides

We use ARM's compiler for HPC, release 18, as starting point for SVE-enabled assembly, as well as the ARM Instruction Emulator (ArmIE) for functional verification of the SVE implementation.

For partitioning the execution of the workload onto the host and DIMM-NDP, we split the workload into execution phases where either the host or the NDP side is active on the working set. In this way, we can manage synchronization with spinlocks on shared memory in software and rely on explicit cache maintenance for consistent data. In order to determine these phases, we profile the workloads on real x86 and ARM servers and not in simulation to:

a) Derive the number and length of the phases for the complete workloads
b) Assess and characterize the loss of scalability due to parallel overhead (such as communication and synchronization) at high core and NDP unit counts, where appropriate depending on the programming approach

With performance counters, we determine hot code sections. We further classify these into CPU and memory-bound sections by using LLC statistics, such as MPKI and LLC usage. We prefer offloading procedures to NDP, either at low level (e.g., for batches of BLAS1 dot product calls in libSVM) or at higher level (e.g., for calls of the optimization solver in liblinear).

6.3.2 Energy Evaluation

We use activity counters to project energy efficiency with annotations for logical and memory operations for recent 10nm-like process technology by extrapolation from [4], [5], [40], [66], see Table 7. We use these for ranking alternatives for design-space exploration, but not for absolute results. We do not consider energy consumption related to waiting for acquiring a spinlock. Spinlocks are the mode of operation for this case study, but can be improved by, e.g., explicit signaling and power down phases triggered over the DIMM-NDP register interface.

7 Performance Evaluation of DIMM-NDP

This section describes results for performance evaluation of DIMM-NDP based on simulation of the microarchitecture in a system setup with selected workloads, as well as an assessment of the software porting effort to reach the reported performance.

7.1 Performance and Efficiency Results

7.1.1 Model Calibration

We use the elementary stream and lmbench tests (subsection 6.1) to align achievable latency and bandwidth by our core models with existing designs.

7.1.1.1 Host "Big" Core: The round-trip latency for read accesses under light load as determined by pointer chasing on main memory reaches about 270 core clock cycles (90ns), which is about 10% to 20% worse than for an high-end server, representing a mid-range setup. Similarly, latencies for the cache levels are set based on measurements on real systems (several Intel Xeon v3 and v4 processor systems, ARM Juno R2 board with ARM Cortex-A72, Huawei TaiShan 2280 ARM server with HiSilicon Kunpeng 916, and APM X-C1 evaluation boards).

7.1.1.2 NDP "Little" Core: The unloaded latency to main memory derived from pointer chasing reaches about 115 core clock cycles (58ns) mainly due to a reduced memory hierarchy compared to the big core setup. As expected, the usable bandwidth from main memory for the little core much more depends on efficient prefetching, either in hardware or software, but can reach the same level of utilization as the big core of about 85% for stream and BLAS1 kernels with the use of the vector unit. The effort for this kernel-specific tuning will be discussed in subsection 7.4.

7.1.2 Kernel-level Performance

We apply isolated single unit models for the host SoC and the NDP unit to evaluate compute kernels. Assuming a balanced symmetric use of the units, we then extrapolate full system performance by the number of units that can work independently (up to 48 cores on the host, up to 64 NDP units in the memory subsystem with 8 channels if 4 ranks per DIMM and two DIMMs per channel are used). For the host model, we restrict the available memory bandwidth and LLC capacity for one core to $1/n^{th}$ of the host's memory bandwidth and LLC capacity, with n representing the total core count we want to extrapolate performance for. For one NDP unit, on the other hand, we assume that the unit can exploit the full bandwidth of one memory channel locally on the DIMM. We exclude initialization time of data structures from the execution time of kernels. We consider a dataset-dependent warmup phase in simulation such that we reach a steady state, both for the use of data (e.g., weight vectors) and the architecture model (e.g., branch predictor state).

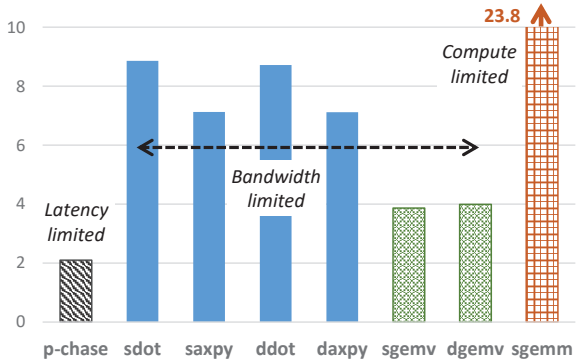


Fig. 6. DIMM-NDP speedup over host CPU for compute kernels, extrapolated from single-unit models.

Fig. 6 shows the speedup for compute kernels on DIMM-NDP compared to the host for 8 NDP units per memory channel. Pointer chasing (*p-chase*) is latency-limited and benefits from the smaller memory hierarchy of NDP. *dot* and *axpy* are BLAS level 1 kernels (vector - vector operations), *gemv* BLAS level 2 (matrix - vector ops), and *gemm* BLAS level 3 (matrix - matrix ops) kernels respectively for either double (d) or single (s) precision floating-point operations. *sgemm* takes advantage of the matrix unit (dimension 16x16 considered for the evaluation).

BLAS-L1 kernels are limited by memory bandwidth and thus take full advantage of the additional bandwidth offered by DIMM-NDP. *sgemm* benefits from matrix operations and is rather limited by computations. sgemm could scale further for larger matrix units.

7.1.3 Performance of Applications

We employ a full-system simulation model, coupling the model of the host with the model of NDP computing by a custom bridge block that allows both sides to operate on shared memory, while the host model has to adhere to NVDIMM-P-like timing. Also, as for the kernel results, the host model is subject to scaled-down bandwidth depending on the number of active cores at the host side. Fig. 7 shows the relative performance for *libSVM* and *liblinear* for different numbers of NDP units per host memory channel. The selection of the right version of a compute kernel (e.g., degree of unrolling and prefetching) is dataset-dependent. The software optimizations are described in more detail in subsection 7.4.

For *liblinear*, we offload the full solver to NDP. Dataset YearPredMSD can take advantage of kernels that are tuned for dense data structures and scales best, coming close to the results of the kernels shown in Fig. 6. The speedup for datasets avazu and news20.bin flattens out for liblinear, since the degree of parallelism is limited by the number of 10 concurrent cross-validation runs and number of classes chosen for this setup. Results are shown for solver 1 (support vector classification). We also evaluated solver 0 (logistic regression) that led to similar results (see section 7.4).

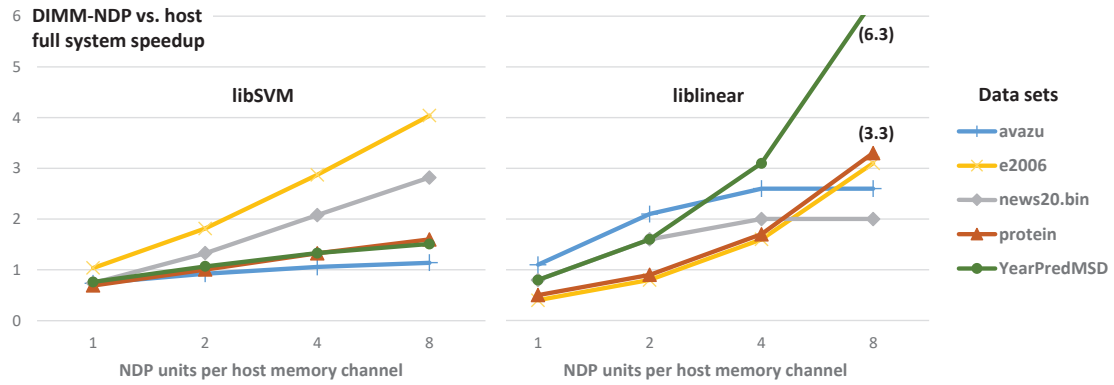


Fig. 7. DIMM-NDP speedup over host CPU for libSVM and liblinear applications, full system.

For *libSVM*, we exploit concurrency by distributing sets of samples to different NDP units on a more fine-grain level than for liblinear, based on the code structure. Since each sample may contain a different number of features, balancing the load of the units becomes more challenging, requiring dynamic workload distribution for best results. Datasets with a limited number of features per sample (YearPredMSD, avazu, protein) show reduced scalability since the time spent in the BLAS L1 *dot* kernel is limited, thus increasing the effort for managing the samples compared to the host.

For *SSD*, about 95% of the time on the host is spent in the convolutional layers that depend by 90% on the execution of the *sgemm* BLAS L3 kernel. By employing the matrix unit and optimizing the data reorganization stage for performing convolutions with matrix multiplications, we are able to achieve a speedup of 5.1. The assumption is that NDP units run independent SSD instances.

TABLE 8
NDP Scratchpad Speedup for liblinear

Dataset	News20	Avazu	Protein	e2006	YearPredictMSD
Speedup	1.94	1.96	1.08	1.68	1.10

7.1.4 Impact of Using the Scratchpad

Table 8 summarizes the speedup for liblinear for using a scratchpad vs. relying on the L1 cache only. For this sensitivity analysis, we approximate the behavior of a scratchpad with a fully associative cache in parallel to the L1 cache that is responsible for a separate memory region (i.e., data placement in memory determines the right cache). For the scratchpad version, we can remove much of the L1-specific prefetch code in favor of bulk transfers to the scratchpad. The scratchpad appears favorable for datasets where a decent number of features can take advantage of the locality offered by the scratchpad.

7.1.5 Energy Efficiency

Employing activity counters (see subsection 6.3), the relative energy required by BLAS L1 and L2 kernels for different vector and matrix sizes is shown in Fig. 8. The baseline is the corresponding execution on the host processor. The reduction of required energy is mainly due to fewer accesses in the memory hierarchy triggered by cache maintenance operations between cache levels. On kernel level, we recognize a 3x to 4x potential for reducing energy.

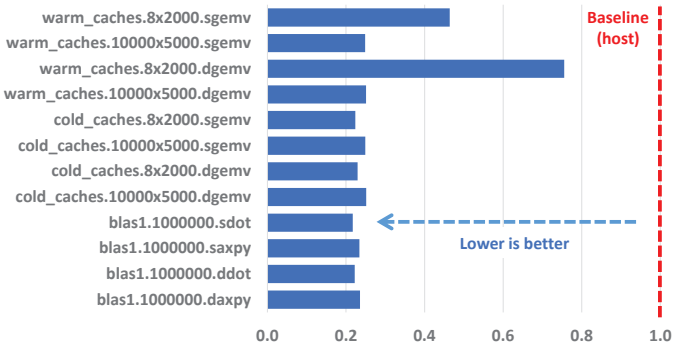


Fig. 8. NDP kernel-level reduction of energy for BLAS L1 and L2 kernels and different vector and matrix sizes.

On application-level, Fig. 9 shows the efficiency results for DIMM-NDP relative to complete execution on the host for *liblinear* and the selected datasets for our full system setup. We attribute 4x the energy for IO signaling from the host compared to direct access by an NDP unit due to our setup with 2 DIMMs per channel and buffers on DIMM. If we reduce this factor to 2x, the results are impacted by less than 3%. The reduction potential by 2x to 3x can thus mainly be attributed to the efficiency of the BLAS Level 1 kernels.

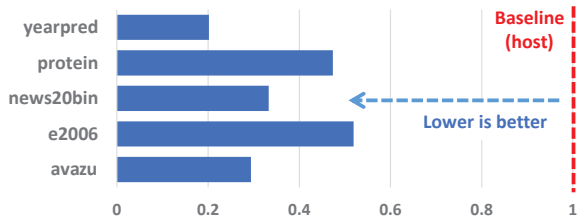


Fig. 9. NDP application-level reduction of energy for liblinear and different data sets.

7.2 Comparison of Near Main Memory Processing vs. Loosely Coupled Accelerator

We perform a sensitivity analysis of the number of synchronizations between the host and NDP units on the

TABLE 9
Coupling Characteristics between Host SoC and NDP Units with Different Interface Options for NDP.

Configuration	Peak bandwidth to/from host	Host latency to NDP memory	Capacity	Peak NDP-local bandwidth	Note
Standard DIMM, 2DPC[a)]	205GB/s	<200ns	512GB	0.8TB/s	host shared memory
Custom DIMM, 2DPC[a)]	205GB/s	<200ns	2TB	1.6TB/s	host shared memory
NDP over PCIe	63GB/s	μs range	≤512GB	≤1.6TB/s	extra DRAM costs
NDP over reduced PCIe[b)]	63GB/s	0.5μs range	≤512GB	≤1.6TB/s	extra DRAM costs

[a)] *2 DIMMs per channel* [b)] *Such as latency-optimized Gen-Z [16] or CCIX [17]*

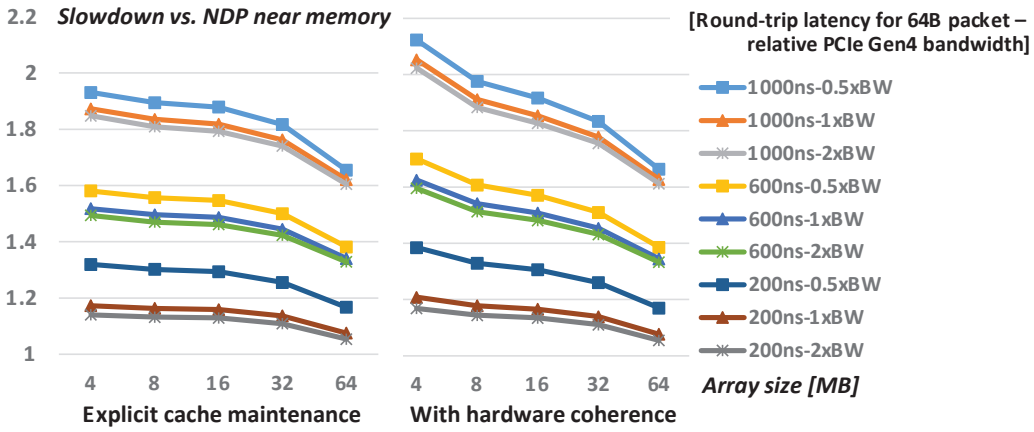


Fig. 10. Slowdown of PCIe setup vs. DIMM-NDP near main memory for pointer chasing workload and varying array sizes.

performance of DIMM-NDP and a loosely PCIe-attached accelerator with the same NDP units and memory capacity. In this way, we characterize the impact of tight vs. loose coupling on overall performance to position DIMM-NDP relative to more established solutions. The comparison is optimistic for the PCIe accelerator since memory must be allocated on the card, usually at lower capacity than the main memory of the host. Additional data partitioning and bulk copying between the host and the accelerator required in this case are not considered here.

7.2.1 PCIe NDP Setup

The DIMM-NDP advantage in terms of bandwidth and latency compared to PCIe for working with the host is summarized in Table 9, considering parameters from Tables 5, 6 and subsection 5.3. For phases where the host and NDP units have to synchronize, our DIMM-NDP solution in main memory should have an up to 3x advantage in terms of bandwidth (if data are copied; by applying cache maintenance, the advantage is higher) and up to 10x in terms of latency. We use pointer chasing as a stress test to highlight the impact of synchronization/communication between the two sides. Arrays are initialized with random access patterns by the host for use with pointer chasing by the NDP units. For the NDP PCIe setup, the arrays have to be copied to the NDP unit after initialization, whereas for NDP in main memory the host performs cache-clean operations. The results of the tests are communicated back to the host accordingly.

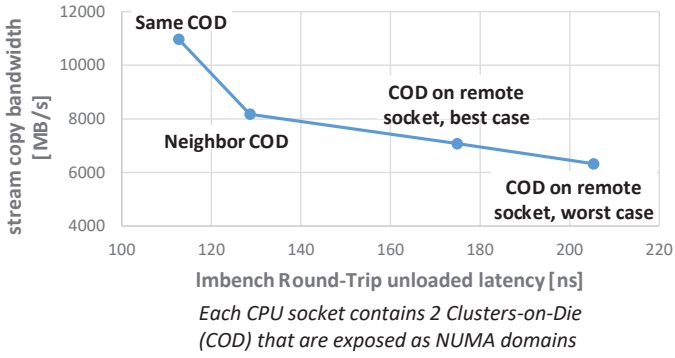


Fig. 11. Degradation of memory access latency and bandwidth with distance between allocated NUMA nodes for compute and memory, on Intel Xeon E5-2660v3 processor based dual socket server.

7.2.2 PCIe NDP Results

Fig. 10 presents the results for the pointer chasing test. We distinguish two coherence methods. The results on the left depend on explicit cache maintenance operations to clean caches on either the host or the NDP side. This represents the mode of operation that we use with DIMM-NDP as well, and models the use of classic PCIe. The results on the right rely on hardware coherence to keep data consistent and thus represent future interfaces like CCIX [17]. We change the round-trip latency and peak bandwidth to provide a sensitivity analysis of these parameters, thus covering the potential of protocol stacks for Gen-Z [16] and CCIX [17].

By varying the array size for pointer chasing, we change the ratio of time spent on the host for initializing the

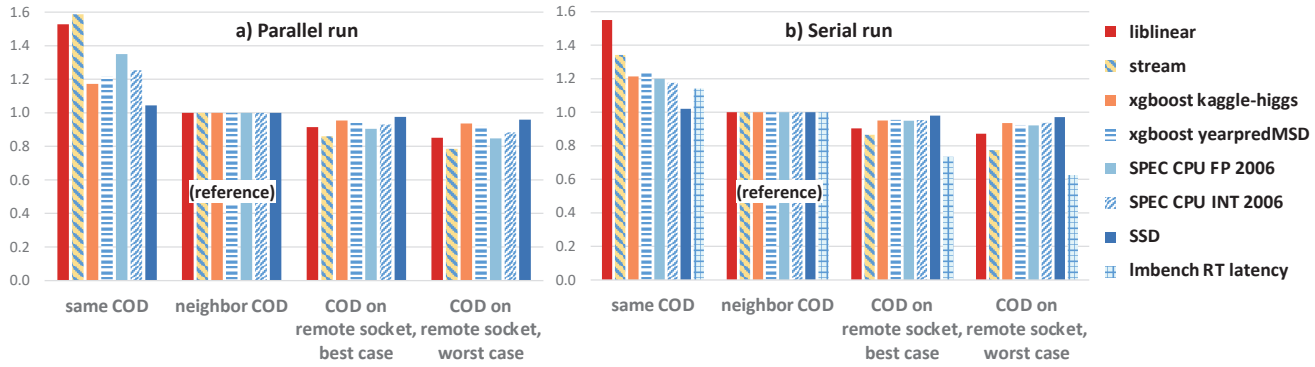


Fig. 12. Degradation of workload performance on Intel Xeon E5-2660v3 processor based dual socket server depending on the allocation of NUMA nodes for compute and memory.

array vs. time spent on NDP for traversing the array by pointer chasing. The baseline (DIMM-NDP) is at least 60% faster than the implementation on the host, i.e., DIMM-NDP performs better, even though the host and NDP units have to synchronize data on a fine time scale. Fig. 10 reveals a slowdown of up to 2x compared to DIMM-NDP for the PCIe cases, effectively slowing down the benchmark compared to the host for selected configurations. We can also see a slight slowdown for the configurations with hardware coherence compared to explicit cache maintenance, whereas explicit cache maintenance requires more programming effort.

7.2.3 Discussion of Near-Data Interface

The DIMM-NDP implementation for this setup is limited by serializing synchronization and computing. The synchronization can further be reduced by switching from using spinlocks on shared memory to a more direct signaling path controlled by the host over the memory-mapped register interface of the NDP units.

Cache maintenance over levels of the memory hierarchy on the host becomes a limiting factor for fine-grain preservation of consistent data, since the processing cores can either compute or do cache maintenance. This is seen in Fig. 10 for small array sizes, as more frequent interactions between the host and the NDP units are needed. The characteristics in Table 9 suggest at least an advantage of 4x for DIMM-NDP over the PCIe configuration for exchanges with the host, whereas the results rather converge to a factor of two. Overall, tighter coupling with the host by employing DIMM-NDP enables more host-centric usages, since smaller kernel calls can be offloaded to NDP units to achieve better performance. This setup also enables a more gradual transition to using accelerators by offloading more and more functions to DIMM-NDP on shared memory.

7.3 Slowdown of Workloads Running Mainly on the Host due to NVDIMM-P Asynchronous Accesses

In order to characterize the impact of additional latency introduced by NVDIMM-P asynchronous accesses on performance for workloads running mainly on the host, we perform a sensitivity analysis of latency on two 2P servers (Intel Xeon E5-2660v3 processor and Huawei TaiShan 2280) by placing compute in one NUMA domain and all memory in a different NUMA domain, e.g., on a remote socket or different cluster-on-die (COD). Fig. 11 shows the characterization by our elementary bandwidth and latency tests using one running instance on the x86 server. For reproducible measurements, turbo mode, hyperthreading, active and idle power savings states are switched off. As placing the NUMA nodes for compute and memory on the same COD has a more than linear advantage (bypassing any interconnect bottleneck between CODs) over configurations with higher distance, we select the "neighbor COD" configuration as reference in the following comparison.

Fig. 12 shows the speedup of workloads depending on the distance between the allocated NUMA memory and compute nodes. A parallel run is confined by the number of cores in one NUMA domain, 5 cores in our case. Beside the elementary tests, liblinear reveals the largest slowdown of the workloads, corresponding to about 3% for every 10ns of latency added, whereas all others degrade by less than 1.1% for every 10ns of latency added. This is in-line with the study by Clapp et al. [69] who estimate up to 3.5% of CPI increase for every 10ns added. Trends on the Huawei TaiShan server are less pronounced, since the reference configuration already incurs higher latency.

We expect the asynchronous access mode of NVDIMM-P to add 10ns to 20ns of latency, since the host's memory controller has to pull data from the DIMM's buffer after the availability is signaled by the buffer. Also, our MedC architecture introduces arbitration in its memory controller MDMC before accesses from the host reach main memory, adding up to about 50ns latency overall. As mitigation for workloads that do not share data with NDP, hybrid channel configurations for the host memory bus are possible: Defined ranks and DIMMs may only be used by the host directly with the DDR4/5 protocol for best performance using the abundant memory capacity of main memory. NVDIMM-P can be enabled rank-by-rank, co-existing with native DDR4/5 accesses on the same host channel.

In our gem5 simulation model, as memory-bound phases are offloaded to NDP and do not depend on the host bus, the overall slowdown due to NVDIMM-P latency on the host bus (by sensitivity analysis of our bridge block in gem5) becomes negligible at less than 1% for liblinear.

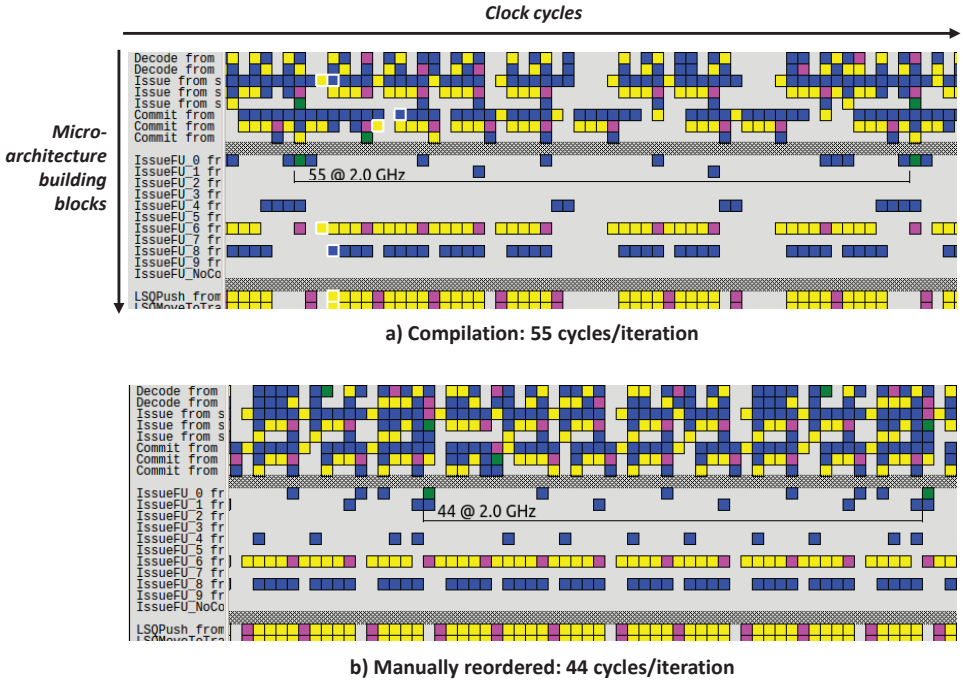


Fig. 13. Dreams [70] screenshot showing microarchitecture events over time before and after manual instruction reordering for a compute loop. The number of cycles per iteration can be reduced by 20% in this example.

7.4 Software Porting Effort to Use DIMM-NDP

In order to reach the reported performance of DIMM-NDP, we iteratively refined the software implementation on the NDP side to take full advantage of data parallelism. The case study underpins the potential of several tuning steps to reach better performance on NDP's in-order core and its vector unit. This can be used as a proxy for estimating the effort for providing customized application-specific libraries for use with DIMM-NDP.

7.4.1 Optimization Steps

We distinguish the following optimization steps to quantify the development effort for particularly tuning the hot BLAS L1 kernels:

- *Host-only reference:* Optimization of data layout to struct-of-arrays and loop fusion to simplify vectorization, using baseline version on the host
- *NDP baseline:* Splitting workload into execution phases based on profiling, implementing spinlocks to establish handover between phases
- *SWPF:* Analyzing performance, adding software prefetches, tuning memory access patterns
- *Unroll:* Experimenting with several unrolling sizes & analyzing instruction scheduling
- *Reorder:* Manual instruction reordering
- *Outer unroll:* Outer loop unrolling with adaption of prefetching and order of instructions
- *Reduction:* Optimization of vector reductions, using predication and register blocking

Note that the effort for parallelization on several NDP units is part of the first two steps and rather straightforward, as the degree of parallelism is similar to the parallelism of the number of cores on the host side. That means, we can take advantage of profiling results and best-known methods on a high-core-count CPU to check the partitioning for NDP.

As the optimization phases may reveal subtle side-effects on a small scale between the microarchitecture blocks of our NDP architecture, we use a custom version of the Dreams tool to trace and track performance effects on cycle granularity together with our gem5 simulation models. Fig. 13 shows an example, where Dreams enables cycle-by-cycle optimization by displaying the impact of manual instruction reordering, as we trace the stages of the processing pipeline.

7.4.2 Porting Results

The following study summarizes the effort for the dataset YearPredMSD. Fig. 14 shows the impact of the optimization steps on the overall speedup that we are able to achieve for the logistic regression solver. Most of the effort of about one man-month has to be invested only once for establishing a domain-specific library with common compute kernels, like BLAS for linear algebra. Then, the application programmer can mainly focus on the data layout and partitioning for NDP. In this way, the programmer can start with a working, host-centric implementation and gradually move functionality to NDP to tune performance.

7.5 Discussion of Further NDP Variants

7.5.1 PCIe Accelerator vs. near Main Memory

We think that near-memory processing at shared memory of the main host SoC allows accelerating CPU-centric processing where "big" cores can continue to shine, e.g., for using interpreted and dynamic languages (python, scripting, etc.) and rich runtime environments (Java, perl, etc.) with frequent calls of kernels, like the compute "dwarfs" [71]. We can take advantage of established programming abstractions for heterogeneity, such as NUMA and domain-specific

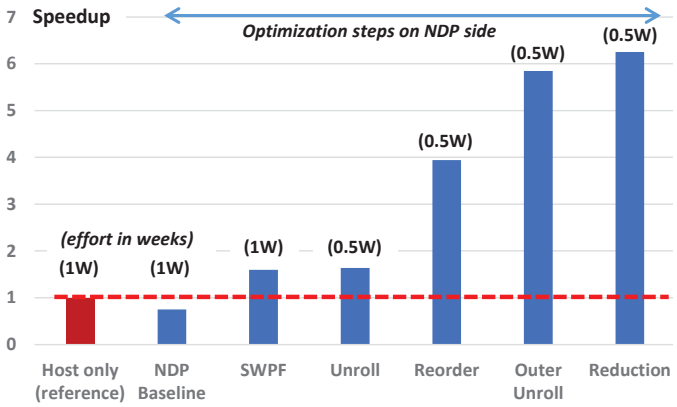


Fig. 14. Impact of iterative optimizations of the NDP-sided software for YearPredMSD dataset and logistic regression solver; relative full-system performance with 8 NDP units per host memory channel are shown with effort estimations in weeks (W) for one experienced programmer.

libraries, to ease the transition to programming NDP, while the burden of data partitioning and managing data is reduced compared to memory capacity-limited accelerators.

7.5.2 Minimal DIMM-NDP Version

As long as the host's memory controllers implement the asynchronous read/write access subset of NVDIMM-P, the register interface for using NDP can be implemented and memory-mapped only at the DIMM side, at the expense of communication overhead on the host memory bus. Likewise, the synchronization between host and NDP may happen completely in software over shared memory without depending on the register interface. In this way, first generation DIMM-NDP can be introduced without having to modify the IP of the host SoC.

7.5.3 Higher DRAM Speed Locally

The use of a MedC buffer on DIMM also enables the use of faster memory devices locally on DIMM, such as GDDR and HBM. Then, MedC always has to translate the protocols on the host bus to local timing and protocols, like GDDR. The host must thus use NVDIMM-P asynchronous requests for all accesses to the DIMM, even if NDP is switched off.

7.5.4 Augmenting the Host with Special Units

One may argue about adding vector and matrix units to the host SoC to achieve better performance. In our study, for BLAS1/2 algebra, already the FPUs of the host saturate the memory bandwidth seen by the host. Vector units do not help the host in this situation, whereas NDP takes advantage of the extra bandwidth available on DIMM. The matrix unit may help the host for codes with high arithmetic intensity. However, the unit is rather hard to integrate into the general-purpose memory hierarchy of the host (including its register context).

8 Concluding Remarks

We have evaluated the hardware feasibility, programming effort and performance of Near-Data Processing (NDP) on memory modules for server applications. Our proposal DIMM-NDP for the hardware architecture and software view targeting scientific applications looks promising for simplifying the transition to NDP by utilizing commodity DRAM devices on memory modules at high memory capacity and bounded overhead costs:

- We can exploit the modularity of DIMMs for variable deployments of NDP and for strengthening the role of the host CPU by broader use with accelerated applications from different application domains.
- As the costs for DRAM become a dominant factor in server deployments, DIMM-NDP makes sure that memory capacity can always be used by the host, even if NDP is switched off.
- Application code can gradually be moved to NDP since the memory is shared and available at full capacity, avoiding data partitioning and copying hassles compared with more capacity-limited accelerators.

As a result, we expect many different application domains to take advantage of programmable NDP units and DIMM-NDP, as needed for a general-purpose server architecture. We position DIMM-NDP complementary to application-specific accelerators, while building on the software infrastructure that has been established for accelerators and heterogeneous computing to ease the transition. Later generations of the technology may step away from a bare-metal implementation and exploit support by an operating system [72], also extending higher-level programming abstractions to NDP units, such as OpenMP and openCL.

Acknowledgments

The authors would like to thank Andrea Nobile, SUN Yunliang and Heiko Schick from Huawei MRC, as well as Oscar Rosell, Isaac Hernández, Javier Bueno and Toni Juan from Metempsy for their support of the project and invaluable feedback.

References

[1] P. Siegl, R. Buchty, and M. Berekovic, "Data-centric computing frontiers: A survey on processing-in-memory," in *Int'l Symposium on Memory Systems (MEMSYS)*. New York, NY, USA: ACM, Oct. 2016, pp. 295–308.

[2] R. Balasubramonian, J. Chang, T. Manning, J. H. Moreno, R. Murphy, R. Nair, and S. Swanson, "Near-data processing: Insights from a MICRO-46 workshop," *IEEE Micro*, vol. 34, no. 4, pp. 36–42, Aug. 2014.

[3] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *Int'l Solid-State Circuits Conference (ISSCC)*. New York, NY, USA: IEEE, Feb. 2014, pp. 10–14.

[4] S. Borkar, "Exascale computing - a fact or a fiction? (keynote)," Int'l Parallel & Distributed Processing Symposium (IPDPS), May 2013.

[5] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the future of parallel computing," *IEEE Micro*, vol. 31, no. 5, pp. 7–17, Oct. 2011.

[6] J. D. McCalpin. (2016, Jul.) STREAM: Sustainable memory bandwidth in high performance computers. [Online]. Available: https://www.cs.virginia.edu/stream/

[7] P. R. Kinget, "Scaling analog circuits into deep nanoscale CMOS: Obstacles and ways to overcome them," in *IEEE Custom Integrated Circuits Conference (CICC)*. New York, NY, USA: IEEE, Sep. 2015.

[8] B. Feinberg, U. K. R. Venalam, N. Whitehair, S. Wang, and E. Ipek, "Enabling scientific computing on memristive accelerators," in *Int'l Symposium on Computer Architecture (ISCA)*. Los Alamitos, CA, USA: IEEE Computer Society, Jun. 2018, pp. 367–382.

[9] E. Azarkhish, D. Rossi, I. Loi, and L. Benini, "Design and evaluation of a processing-in-memory architecture for the smart memory cube," in *Int'l Conference on Architecture of Computing Systems (ARCS)*. Basel, Switzerland: Springer Int'l Publishing, Apr. 2016, pp. 19–31.

[10] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," in *Int'l Symposium on Computer Architecture (ISCA)*. New York, NY, USA: ACM, Jun. 2015, pp. 105–117.

[11] M. Gao and C. Kozyrakis, "HRL: Efficient and flexible reconfigurable logic for near-data processing," in *Int'l Symposium on High Performance Computer Architecture (HPCA)*. Los Alamitos, CA, USA: IEEE Computer Society, Mar. 2016, pp. 126–137.

[12] R. Nair, S. F. Antao, C. Bertolli, P. Bose, J. R. Brunheroto, T. Chen, C.-Y. Cher, C. H. A. Costa, J. Doi, C. Evangelinos, B. M. Fleischer, T. W. Fox, D. S. Gallo, L. Grinberg, J. A. Gunnels, A. C. Jacob, P. Jacob, H. M. Jacobson, T. Karkhanis, C. Kim, J. H. Moreno, J. K. O'Brien, M. Ohmacht, Y. Park, D. A. Prener, B. S. Rosenburg, K. D. Ryu, O. Sallenave, M. J. Serrano, P. D. M. Siegl, K. Sugavanam, and Z. Sura, "Active memory cube: A processing-in-memory architecture for exascale systems," *IBM Journal of Research and Development*, vol. 59, no. 2/3, 2015.

[13] M. B. Taylor, "Is dark silicon useful?" in *Design Automation Conference (DAC)*. New York, NY, USA: ACM, Jun. 2012, pp. 1131–1136.

[14] B. Gervasi and J. Hinkle, "Overcoming system memory challenges with persistent memory and NVDIMM-P," in *JEDEC Server Forum*. Arlington, VA, USA: JEDEC, Jun. 2017.

[15] *Proposed DDR5 NVDIMM-P Bus Protocol*, 2261.13C ed., JEDEC Solid State Technology Association, committee JC-45.6, Nov. 2017.

[16] *Gen-Z Core Specification*, 1st ed., Gen-Z Consortium, https://genzconsortium.org, Feb. 2018.

[17] *An Introduction to CCIX*, CCIX Consortium, Inc., https://www.ccixconsortium.com, 2018.

[18] University of Tennessee Knoxville and Oak Ridge National Lab. (2017, Nov.) BLAS (basic linear algebra subprograms). [Online]. Available: http://www.netlib.org/blas/

[19] E. Vermij, C. Hagleitner, L. Fiorin, R. Jongerius, J. van Lunteren, and K. Bertels, "An architecture for near-data processing systems," in *ACM Int'l Conf. on Computing Frontiers (CF)*. New York, NY, USA: ACM, May 2016, pp. 357–360.

[20] A. Boroumand, S. Ghose, M. Patel, H. Hassan, B. Lucia, K. Hsieh, K. T. Malladi, H. Zheng, and O. Mutlu, "LazyPIM: An efficient cache coherence mechanism for processing-in-memory," *IEEE Computer Architecture Letters*, vol. 16, no. 1, pp. 46–50, 2017.

[21] A. Farmahini-Farahani, J. H. Ahn, K. Morrow, and N. S. Kim, "NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules," in *Int'l Symposium on High Performance Computer Architecture (HPCA)*. Los Alamitos, CA, USA: IEEE Computer Society, Feb. 2015, pp. 283–295.

[22] J. Cong, Z. Fang, F. Javadi, and G. Reinman, "AIM: Accelerating computational genomics through scalable and noninvasive accelerator-interposed memory," in *Int'l Symposium on Memory Systems (MEMSYS)*. New York, NY, USA: ACM, Oct. 2017, pp. 3–14.

[23] M. Alian, S. W. Min, H. Asgharimoghaddam, A. Dhar, D. K. Wang, T. Roewer, A. McPadden, O. OHalloran, D. Chen, J. Xiong, D. Kim, W. mei Hwu, and N. S. Kim, "Application-transparent near-memory processing architecture with memory channel network," in *IEEE/ACM Int'l Symposium on Microarchitecture (MICRO)*. Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2018, pp. 803–815.

[24] H. Asghari-Moghaddam, Y. H. Son, J. H. Ahn, and N. S. Kim, "Chameleon: Versatile and practical near-DRAM acceleration architecture for large memory systems," in *IEEE/ACM Int'l Symposium on Microarchitecture (MICRO)*. Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2016.

[25] M. Drumond, A. Daglis, N. Mirzadeh, and D. Ustiugov, "The Mondrian data engine," in *Int'l Symposium on Computer Architecture (ISCA)*. New York, NY, USA: ACM, Jun. 2017, pp. 639–651.

[26] T. Dysart, P. Kogge, M. Deneroff, E. Bovell, P. Briggs, J. Brockman, K. Jacobsen, Y. Juan, S. Kuntz, R. Lethin, J. McMahon, C. Pawar, M. Perrigo, S. Rucker, J. Ruttenberg, M. Ruttenberg, and S. Stein, "Highly scalable near memory processing with migrating threads on the Emu system architecture," in *Workshop on Irregular Applications: Architecture and Algorithms (IA3)*. Los Alamitos, CA, USA: IEEE Computer Society, Nov. 2016, pp. 2–9.

[27] S. L. Xi, O. Babarinsa, M. Athanassoulis, and S. Idreos, "Beyond the wall: Near-data processing for databases," in *Int'l Workshop on Data Management on New Hardware (DaMoN)*. New York, NY, USA: ACM, Jun. 2015.

[28] M. Gokhale, S. Lloyd, and C. Hajas, "Near memory data structure rearrangement," in *Int'l Symposium on Memory Systems (MEMSYS)*. New York, NY, USA: ACM, Oct. 2015, pp. 283–290.

[29] B. Akin, F. Franchetti, and J. C. Hoe, "Data reorganization in memory using 3D-stacked DRAM," in *Int'l Symposium on Computer Architecture (ISCA)*. New York, NY, USA: ACM, Jun. 2015, pp. 131–143.

[30] M. Scrbak, M. Islam, K. M. Kavi, M. Ignatowski, and N. Jayasena, "Exploring the processing-in-memory design space," *Elsevier Journal of Systems Architecture*, vol. 75, pp. 59–67, Apr. 2017.

[31] S. Lee, B. Jeon, K. Kang, D. Ka, N. Kim, Y. Kim, Y. Hong, M. Kang, J. Min, M. Lee, C. Jeong, K. Kim, D. Lee, J. Shin, Y. Han, Y. Shim, Y. Kim, Y. Kim, H. Kim, J. Yun, B. Kim, S. Han, C. Lee, J. Song, H. Song, I. Park, Y. Kim, J. Chun, and J. Oh, "A 512GB 1.1v managed DRAM solution with 16GB ODP and media controller," in *Int'l Solid-State Circuits Conference (ISSCC)*. New York, NY, USA: IEEE, Feb. 2019, pp. 384–385.

[32] *LRDIMM DDR3 Memory Buffer (MB)*, JESD82-30 ed., JEDEC Solid State Technology Association, Oct. 2014.

[33] *FBDIMM Advanced Memory Buffer (AMB)*, JESD82-20A ed., JEDEC Solid State Technology Association, Mar. 2009.

[34] *ARM Architecture Reference Manual – ARMv8, for ARMv8-A architecture profile*, ARM DDI 0487d.a (ID103018) ed., ARM Ltd., Oct. 2018.

[35] *ARM Architecture Reference Manual Supplement, The Scalable Vector Extension (SVE), for ARMv8-A*, ARM DDI 0584a.d (ID122117) ed., ARM Ltd., Dec. 2017.

[36] A. Nobile and G. von Boehn, "Multiply accumulator array and processor device," World Intellectual Property Organization (WIPO), Int'l Publication Number WO 2018/228703 Al, Dec. 2018.

[37] M. Gao, G. Ayers, and C. Kozyrakis, "Practical near-data processing for in-memory analytics frameworks," in *Int'l Conference on Parallel Architecture and Compilation (PACT)*. Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2015, pp. 113–124.

[38] G. H. Khaksari, R. K. Karne, and A. L. Wijesinha, "A bare machine application development methodology," *FCS Int'l Journal of Computers and Their Applications (IJCA)*, vol. 19, no. 1, pp. 10–25, Mar. 2012.

[39] L. Gwennap, "Kirin 950 takes performance lead," in *Mobile Chip Report*. Mountain View, CA, USA: The Linley Group, Nov. 2015.

[40] T. Vogelsang, "Understanding the energy consumption of dynamic random access memories," in *IEEE/ACM Int'l Symposium on Microarchitecture (MICRO)*. Los Alamitos, CA, USA: IEEE Computer Society, 2010, pp. 363–374.

[41] *Calculating Memory Power for DDR4 SDRAM*, TN-40-07 ed., Micron Technology, Inc., 2017.

[42] C. Gonzalez, E. Fluhr, D. Dreps, D. Hogenmiller, R. Rao, J. Paredes, M. Floyd, M. Sperling, R. Kruse, V. Ramadurai, R. Nett, S. Islam, J. Pille, and D. Plass, "POWER9: A processor family optimized for cognitive computing with 25Gb/s accelerator links and 16Gb/s PCIe Gen4," in *Int'l Solid-State Circuits Conference (ISSCC)*. New York, NY, USA: IEEE, Feb. 2017, pp. 50–51.

[43] B. Bowhill, B. Stackhouse, N. Nassif, Z. Yang, A. Raghavan, C. Morganti, C. Houghton, D. Krueger, O. Franza, J. Desai, J. Crop, D. Bradley, C. Bostak, S. Bhimji, and M. Becker, "The Xeon processor E5-2600 v3: A 22nm 18-core product family," in *Int'l Solid-State Circuits Conference (ISSCC)*. New York, NY, USA: IEEE, Feb. 2015, pp. 1–3.

[44] I. Cutress. (2017, Jun.) Analyzing the silicon: Die size estimates and arrangements: The Intel Skylake-X review. [Online]. Available: https://www.anandtech.com/show/11550/

[45] S.-Y. Wu, C. Y. Lin, M. C. Chiang, J. J. Liaw, J. Y. Cheng, S. H. Yang, M. Liang, T. Miyashita, C. H. Tsai, B. C. Hsu, H. Y. Chen, T. Yamamoto, S. Y. Chang, V. S. Chang, C. H. Chang, J. H. Chen, H. F. Chen, K. C. Ting, Y. K. Wu, K. H. Pan, R. F. Tsui, C. H. Yao, P. R. Chang, H. M. Lien, T. L. Lee, H. M. Lee, W. Chang, T. Chang, R. Chen, and M. Yeh, "A 16nm FinFET CMOS technology for mobile SoC and computing applications," in *Int'l Electron Devices Meeting (IEDM)*. New York, NY, USA: IEEE, Dec. 2013.

[46] M. Stantic, O. Palomar, T. Hayes, I. Ratkovic, A. Cristal, O. Unsal, and M. Valero, "An integrated vector-scalar design on an in-order ARM core," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, Jul. 2017.

[47] Y. Ge, M. Tomono, M. Ito, and Y. Hirose, "High-performance and low-power consumption vector processor for LTE baseband LSI," *Fujitsu Scientific and Technical Journal (FSTJ)*, vol. 50, no. 1, pp. 132–137, Jan. 2014.

[48] Y. Lee, C. Schmidt, S. Karandikar, D. Dabbelt, A. Ou, and K. Asanovic, "Hwacha preliminary evaluation results, v3.8.1," University of California at Berkeley, Electrical Engineering and Computer Sciences, Tech. Rep. UCB/EECS-2015-264, Dec. 2015.

[49] N. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. luc Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," in *Int'l Symposium on Computer Architecture (ISCA)*. New York, NY, USA: ACM, Jun. 2017, pp. 1–12.

[50] A. N. S. Sarma and V. D. Ambali, "Cooling solution for computing and storage server," in *IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm)*. New York, NY, USA: IEEE, Jun. 2017, pp. 840–849.

[51] *DDR4 Registering Clock Driver (DDR4RCD01)*, JEDEC JESD82-31 ed., JEDEC Solid State Technology Association, Aug. 2016.

[52] *DDR4 Data Buffer Definition (DDR4DB01)*, JEDEC JESD82-32 ed., JEDEC Solid State Technology Association, Nov. 2016.

[53] *Intel C112/C114 Scalable Memory Buffer (SMB) data sheet*, 332444-001 ed., Intel Corp, May 2015.

[54] *POWER8 Memory Buffer Datasheet for DDR3 Applications*, 1st ed., IBM Corp., Jan. 2016.

[55] NCAB Group, "Cost drivers in PCB production," NCAB Group Seminars, 2015. [Online]. Available: https://www.ncabgroup.com/downloads/

[56] TechInsights Inc. (2017, Nov.) Cost comparison Huawei Mate 10, iPhone 8, Samsung Galaxy S8. [Online]. Available: https://www.techinsights.com/blog/cost-comparison-huawei-mate-10-iphone-8-samsung-galaxy-s8

[57] M. Alfano, B. Black, J. Rearick, J. Siegel, M. Su, and J. Din, "Unleashing fury: A new paradigm for 3-D design and test," *IEEE Design & Test*, vol. 34, no. 1, pp. 8–15, Feb. 2017.

[58] S. Burke. (2017, Aug.) The cost of HBM2 vs. GDDR5 & why AMD had to use it. [Online]. Available: https://www.gamersnexus.net/guides/3032-vega-56-cost-of-hbm2-and-necessity-to-use-it

[59] L. Gwennap, "Cortex-A76 rev amps core design," in *Microprocessor Report*. Mountain View, CA, USA: The Linley Group, Jun. 2018.

[60] C. Staelin and L. McVoy. (2007, Nov.) Lmbench - system benchmarks. [Online]. Available: http://lmbench.sourceforge.net/man/lmbench.8.html

[61] Kaggle Inc. (2017) The state of data science & machine learning. [Online]. Available: https://www.kaggle.com/surveys/2017

[62] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "LIBLINEAR: A library for large linear classification," *Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, Aug. 2008. [Online]. Available: https://github.com/cjlin1/liblinear

[63] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, pp. 27:1–27:27, 2011. [Online]. Available: https://www.csie.ntu.edu.tw/~cjlin/libsvm/

[64] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *European Conference on Computer Vision (ECCV)*. Cham, Switzerland: Springer LNCS 9905, Oct. 2016, pp. 21–37. [Online]. Available: https://github.com/weiliu89/caffe/tree/ssd

[65] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *22nd ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, Aug. 2016, pp. 785–794. [Online]. Available: https://xgboost.ai/

[66] K. Czechowski, V. W. Lee, E. Grochowski, R. Ronen, R. Singhal, R. Vuduc, and P. Dubey, "Improving the energy efficiency of big cores," in *Int'l Symposium on Computer Architecture (ISCA)*. Los Alamitos, CA, USA: IEEE, Jun. 2014, pp. 493–504.

[67] S. M. Hassan, S. Yalamanchili, and S. Mukhopadhyay, "Near data processing: Impact and optimization of 3D memory system architecture on the uncore," in *Int'l Symposium on Memory Systems (MEMSYS)*. New York, NY, USA: ACM, Oct. 2015, pp. 11–21.

[68] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, May 2011.

[69] R. Clapp, M. Dimitrov, K. Kumar, V. Viswanathan, and T. Willhalm, "Quantifying the performance impact of memory latency and bandwidth for big data workloads," in *Int'l Symposium on Workload Characterization (IISWC)*. Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2015, pp. 213–224.

[70] Intel Labs Barcelona, "Overview - Dreams - AWB/Leap projects," 2013. [Online]. Available: http://asim.csail.mit.edu/redmine/projects/dreams

[71] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick, "A view of the parallel computing landscape," *Communications of the ACM*, vol. 52, no. 10, pp. 56–67, Oct. 2009.

[72] A. Barbalace, A. Iliopoulos, H. Rauchfuss, and G. Brasche, "It's time to think about an operating system for near data processing architectures," in *16th Workshop on Hot Topics in Operating Systems (HotOS)*. New York, NY, USA: ACM, May 2017, pp. 56–61.