Executing Data Integration Effectively and Efficiently Near the Memory

Chenfeng Zhao, Xuan Zhang, and Roger D. Chamberlain
McKelvey School of Engineering
Washington Univ. in St. Louis

***Abstract*—The ability to perform data analysis on large data sets initially requires the data to be in a prescribed format. Transforming data from its initial form into the desired form is becoming more and more of an overall performance bottleneck. We explore the use of near-memory processing to both accelerate the execution of data transformation workloads and reduce their energy needs. Relative to a traditional architecture, average speedups are $3.5\times$ with 76% energy savings across a benchmark of 12 data integration applications.**

I. Introduction

Not only is big data voluminous, it is varied. Individual data analysis tasks must first collect data sets that are often in disparate locations and a wide variety of formats. These data sets almost always need some form of transformation applied to them prior to the analysis task itself, such as format normalization, data cleansing, type checking, and outlier detection. These data integration activities typically consume an inordinate amount of time and effort both on the part of the data analyst and the computing systems that execute them.

Data Integration is a term frequently used for the general problem of taking data in some initial form and transforming it into a desired form. While the individual transforms are each (mostly) quite straightforward, the task is quickly complicated by the fact that individual data streams can be quite large and there are frequently many streams, each requiring a distinct transformation specification. Tens to hundreds of multi-gigabyte data streams must be concurrently integrated, and this must be done prior to the real data analysis, the ultimate goal. The issue of how to effectively achieve data integration is a pain point for enterprise data, sensor data, scientific data, financial data, to name a few.

Here, we investigate the use of near-memory processing to execute data integration tasks. In particular, we seek to exploit two properties that are common to many data integration workloads and are well-suited to execution on near-memory processing architectures.

1) **Abundant parallelism** – data integration workloads are, for the most part, embarrassingly parallel, so their performance scales well with large numbers of processor cores. Relative to traditional computing systems, the majority of near-memory processing architectures employ a larger number of smaller cores.
2) **Substantial data movement** – they are also characterized by having a large fraction of their operations data movement instructions, implying a strong sensitivity to the architecture of the memory subsystem. A central feature of near-memory processing systems is the proximity of the processor cores to the memory.

We explore the impact of each of these properties, quantifying both the performance implications and energy savings achievable though the use of near-memory processing architectures for executing data integration workloads.

We do not propose any new near-memory processing architecture, but rather utilize approaches that have been previously described. We make the following contributions in this work:

- We characterize data integration workloads to gain insights on the suitability and potential performance benefits of executing data integration near the memory.
- We quantitatively evaluate near-memory processing for the execution of data integration workloads.
- We assess both the performance improvement and energy savings that are achievable, and separately examine the distinct implications of wider parallelism (i.e., a larger number of simpler cores) and lower memory access overheads (i.e., physical location of the cores near the memory).

Overall, for the particular near-memory processing system modeled, the performance improved an average $3.5\times$ and the energy reduced an average $4.2\times$ (76%) relative to a traditional baseline system.

II. Background and Related work

A. Data Integration

As an illustration of the importance of data integration, consider the following two examples from graph analysis and cloud micro-services. In graph analysis, Malicevic et al. [6] describe an improvement to a breadth first search algorithm that results in a $3\times$ improvement in execution time for the breadth first search in isolation. However, it requires the graph data to be in a different form, and when one includes the necessary pre-processing in the performance measurement, the overall execution increases by $1.5\times$. Second, in cloud micro-services, Pourhabibi et al. [8] report that up to 30% of execution time is currently spent in the data format transforming process, and as protocol processing is improved by the use of smart NICs that fraction will only increase. They propose using a custom accelerator to execute these workloads.

B. Near-Memory Processing

The emergence of 3-D stacked memory technology has opened the door for practical deployment of processor cores near the physical DRAM. The structure of these memories has multiple DRAM chips stacked on top of a single logic chip. These chip layers are connected by vertical high-bandwidth and low-power through-silicon vias (TSVs). The logic layer at the bottom consists of both interconnections and controller

logic. In current commercial implementations, the logic layer is not fully utilized (i.e., there is a portion of unused area on the chip). Therefore, the research community has considered integrating general-purpose processor cores or custom accelerators into the logic layer as an approach to implementing a near-memory processing strategy.

There have been a number of proposed near-memory processing (NMP) architectures. Drumond et al. [3] proposed an architecture that utilizes general-purpose Arm Cortex-A35 CPUs as near-memory processing cores. In addition, they altered the execution of common data analytics operators to be more NMP-friendly by optimizing for sequential memory accesses over random memory accesses. Boroumand et al. [1] proposed a near-memory processing architecture with either general-purpose cores or programmable accelerators for Google workloads. Peng et al. [7] explored the suitability of HPC scientific applications on an NMP architecture with general-purpose cores as the execution unit. Our work also utilizes general-purpose programmable cores with small caches in the NMP architecture. However, compared to these prior works, we focus on a different application domain. There has also been recent work utilizing custom logic as execution units. For example, Jang et al. [4] present a accelerator-based NMP architecture for a set of primitives in garbage collection workloads. Singh et al. [10] recently published a survey of the field.

The present work seeks to exploit these ideas for acceleration of data integration workloads.

III. Workload Characterization

For the empirical work in this paper, we use the the Data Integration Benchmark Suite (DIBS) [2] as representative workloads. We characterize them using four metrics: temporal locality, spatial locality, memory access rate, and arithmetic instruction rate. These workloads have a high degree of data movement, motivating the emphasis on memory in the application characterization.

Temporal and spatial locality are quantified using the techniques proposed by Weinberg et al. [12]. Each locality score is on a normalized range [0,1], with higher scores indicating a greater degree of locality.

Figure 1(a) shows the temporal and spatial locality of each of the DIBS applications. We classify the applications into 3 classes: (1) low spatial and low temporal locality, (2) low spatial and high temporal locality, and (3) high spatial locality.

The *fix_float* application, with relatively low spatial locality and lowest temporal locality belongs to class 1. Others have shown that this type of workload can benefit from near-memory processing techniques, e.g., see [10].

The *edgelist_csr* application, with low spatial locality and high temporal locality, belongs to class 2. In this case, the high temporal locality implies that a deep cache hierarchy can benefit performance, so it might not do as well on a near-memory architecture.

The remaining 10 out of the 12 applications, belonging to class 3 with high spatial locality, lie at the right of Figure 1(a).

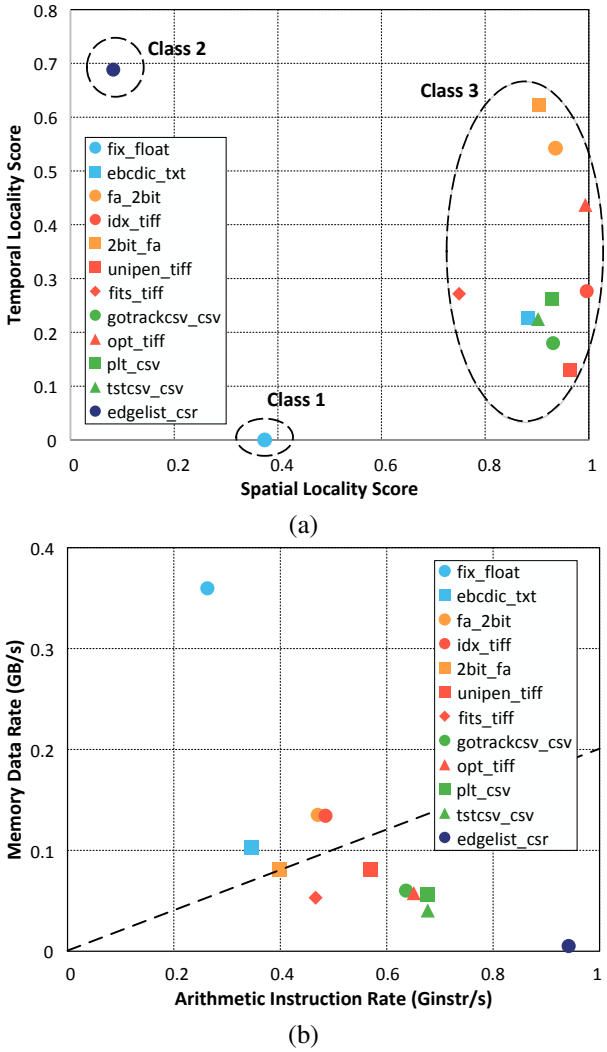

Fig. 1. Workload characterization of data integration workloads. (a) Temporal and spatial locality scores. (b) Memory data rate vs. arithmetic instruction rate.

These locality scores give mixed signals as to the suitability of these workloads for near-memory processing techniques.

We measure the memory accesses and arithmetic instructions to further characterize our data integration workloads. In addition, we will discuss this characterization in terms of their ratio, which we call the *memory/ops* ratio.

Figure 1(b) shows the memory data rate and arithmetic instruction rate in the two-dimensional space. From the figure, it can be observed that *fix_float* has the highest memory/ops ratio, indicating that this workload is the most promising one that could benefit from a near-memory processing architecture.

Consistent with the locality characterization of Figure 1(a), 10 workloads with relatively high spatial locality scores also show medium memory/ops ratios. Among these 10 workloads, *ebcdic_txt*, *fa_2bit*, and *idx_tiff* have higher memory/ops ratios.

The *edgelist_csr* application is the most computationally intensive workload, making it an outlier in both characterizations (a fact it has in common with *fix_float*).

To distinguish the workloads by the potential benefits provided by near-memory processing, we draw a dashed line in Figure 1(b). The workloads above the line are more memory intensive and have the greater chance for performance improvement. While the particular slope of the line is arbitrary, we return to this point in Section VI.

IV. Proposed Near-Memory System

The high-level architecture of our near-memory processing system is illustrated in Figure 2. As previously proposed by Pugsley et al. [9], a number of memory channels (four in Figure 2(a)) are used to connect the host multicore chip to a set of 3-D memory stacks (eight in the figure), using the "far memory" topology described by Micron for HMC. In the performance analysis that follows, we assume that the host multicore chip contains 16 traditional, out-of-order processor cores with private L1 I&D caches, private L2 caches, and a shared L3 last-level cache (for the time being, ignore the dashed wide-parallel box in the figure).

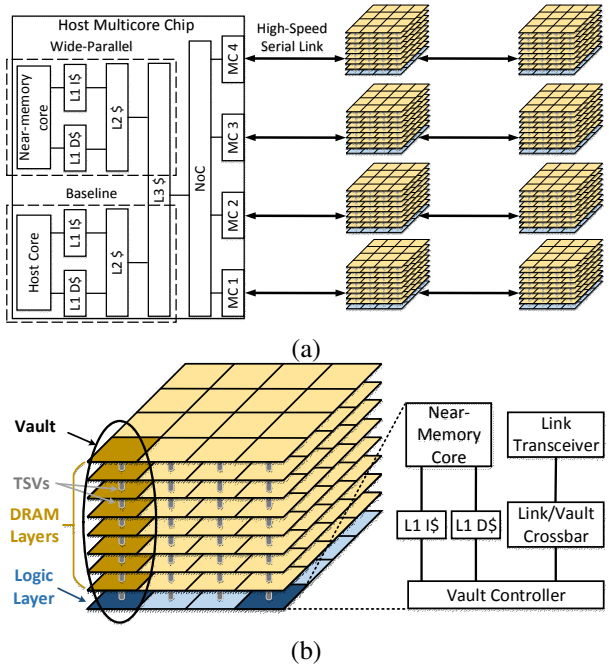

Fig. 2. Architecture of near-memory processing system. (a) Stacked memory connection diagram (adapted from [9]). (b) Near-memory processor diagram in each vault.

Figure 2(b) shows the design of the near-memory processing subsystem. Associated with each vault is a low-power, in-order core that has immediate access to the memory within the vault, talking directly to the vault controller. This in-order core is fabricated on the logic layer chip, using previously empty chip area. It has L1 I&D caches, but no L2 or L3 cache. This arrangement yields 16 near-memory cores for each memory stack, resulting in 128 near-memory cores for a system with 8 memory stacks. The link/vault crossbar and vault controller are unchanged from a typical HMC-like memory stack.

If we are to utilize otherwise empty chip area on the logic layer, we must ensure that not only area, but power constraints are not exceeded. Jeddeloh and Keeth [5] report 68 mm^2 of area on the logic layer, or 4.3 mm^2 per vault. Using the Arm Cortex-A7 as a candidate near-memory processor core, the core plus 32 KB L1 I&D caches have an area of 0.45 mm^2 and a total power consumption below 100 mW.[1] This represents a usage of only about 10% of the available area.

A traditional HMC stack has 4 memory channels that consume a total of 5.8 W [9]. However, the far memory topology illustrated in Figure 2(a) only utilizes 2 channels per memory stack. If the remaining 2 channels are powered off (or not fabricated at all), this makes 2.9 W available for the near-memory cores. 16 cores at 100 mW each only requires 1.6 W, which is well within the power budget.

The advantages of this architecture for executing data integration workloads are two-fold. First, because the workloads are straightforward to parallelize, a larger number of smaller cores are well matched to the computational requirements. Using the same reasoning as the designers of the IBM BlueGene family of supercomputers, a larger number of smaller cores can yield both performance benefits and energy savings if the problem has sufficient parallelism. This is the same notion that motivated the big.LITTLE systems from Arm. Second, because the memory access patterns of the workloads are primarily local, associating a near-memory core with each memory vault results in the bulk of memory accesses being local. Not only do the accesses not have to traverse the memory channel(s) across the circuit board, but most don't even have to traverse the internal crossbar of the memory stack.

To execute the data integration workloads, the data are divided into smaller partitions and mapped to each computation unit by the host processors. As each partition is independent, they can be processed by the near-memory cores in parallel. After the data integration computations are complete, the transformed data will be aggregated and processed by the host processors. This allows the downstream application processing to take advantage of the complex cache system and high computation ability available on the host side.

V. Methodology

We evaluate the performance and energy consumption of data integration workloads via architectural simulation (using the `gem5` simulator). First, we describe three distinct architectures that we use to perform the evaluations: (1) the near-memory processing system that is the primary target of the investigation, (2) a traditional, out-of-order core system used as the baseline/host for comparison purposes, and (3) a wide-parallel system that utilizes low-power, in-order cores which serves as an intermediate system between the baseline/host system and the near-memory processing system. It is used to distinguish between various factors in the performance assessment. After this, we describe additional details of the simulation, including the methods for performance calibration and energy estimation.

[1] https://developer.arm.com/ip-products/processors/cortex-a/cortex-a7

A. Near-Memory Processing System

The structure of the architectural simulation model for the near-memory processing system essentially follows Figure 2. The particulars of the host processor cores and memory stacks are described below in Section V-B. Here, we provide the particulars of the near-memory processors.

As alluded to in the previous section, we use an Arm Cortex-A7 in-order, pipelined core as the processor model for the near-memory computation. Each core is clocked at 1.2 GHz and has 32 KB L1 I&D caches, and each vault of the memory stack is allocated one near-memory core. With 8 memory stacks and 16 vaults per stack, there are 128 near-memory cores.

The close association of each near-memory core with a particular vault of the memory stack implies a non-uniform memory access (NUMA) latency. Accesses to remote vaults on the same stack must traverse the on-logic-chip crossbar, and accesses to vaults on remote stacks must access the topological path(s) shown in Figure 2(a). On the other hand, accesses to the local vault need only talk to the local vault controller. Each vault has a 32-bit vertical interface with 2 Gb/s TSV signaling rate [5]. Thus, an internal bandwidth of 8 GB/s can be achieved in each vault, and overall memory bandwidth is 1 TB/s.

We do not assume coherent caches across the near-memory cores, but rather insert explicit cache flush instructions to enforce memory consistency.

B. Baseline/Host System

The baseline/host processors have a common design, but serve two distinct purposes in our evaluations. First, they serve as the baseline for both performance and energy comparison purposes. As such, the quantitative evaluation is normalized to the baseline system's performance and energy usage. Figure 2(a) illustrates this baseline system if one assumes there are no near-memory cores in the logic layer of the memories. Second, they serve as the host processor(s) for the near-memory processing system. In this circumstance, they are available to execute other tasks concurrently with the near-memory subsystem.

To maintain a common ISA across the entire system, we employ 16 Cortex-A15 out-of-order cores as the processor model for the baseline/host computation. Each core is clocked at 2 GHz and has its own 32 KB L1 I&D caches, 256 KB L2 cache, and shared L3 8 MB last-level cache. Cache coherence is maintained using the traditional MESI-style protocol.

The baseline/host multicore chip has 4 memory channels, each modeled after HMC stacked memory, giving a total memory bandwidth of 160 GB/s. With 8 memory stacks of 4 GB each, the memory capacity is 32 GB. This is straightforward to alter given the flexibility of the "far memory" topology. The bandwidth to/from the baseline/host cores, however, is limited by the number of memory channels available on the baseline/host.

C. Wide-Parallel System

When comparing the proposed near-memory processing system with the traditional baseline system, there are two substantial differences that can (and should) be evaluated separately. One, a larger number of simple cores are utilized in place of a smaller number of complex cores, and two, each simple core has lower-latency, higher-bandwidth access to (a subset of) the memory. To give us the ability to query the performance and energy usage implications of each of these two features separately, we consider an intermediate, wide-parallel system that only includes the first of the above two substantial differences from the baseline system.

The wide-parallel system has the same cache and memory configurations as the aforementioned host/baseline system. However, this system replaces the 16 Cortex-A15 cores with 128 Cortex-A7 cores, the same type and number of cores used in the near-memory system. Figure 2(a) illustrates this wide-parallel system via the dashed box showing a near-memory core.

Fortunately, the two substantial differences between the near-memory processing architecture and the baseline traditional architecture also correspond closely to the two primary characteristics that are common across the target data integration workloads. The abundant parallelism in the workloads can benefit from the larger number of simple cores, and the substantial (local) data movement can benefit from the positioning of those cores close to memory.

By comparing the baseline system to the wide-parallel system, we can discern the impact and importance of the parallelism in both the workloads and the execution architecture. By comparing the wide-parallel system to the near-memory system, we can discern the impact and importance of the memory bandwidth and latency. Finally, we can see the overall impact of the near-memory system by comparing it to the baseline system.

The parameters of all three of these systems are shown in Table I.

D. Simulation

All of our simulation models are built in `gem5`. The standard distribution contains a stacked memory model based upon HMC and processor core models for both the Cortex-A15 cores (`ex5_big.py`) and Cortext-A7 cores (`ex5_LITTLE.py`).

The data integration workloads come from DIBS [2], which provides both source code and input data sets. All are compiled with `gcc` version 5.4.0 utilizing `-O3` optimizations. To enforce cache coherence between the host caches and the near-memory caches, we extended the core model to support cache flushing. In the cache flushing API, the corresponding cache block is determined based on the physical address which is re-translated from the virtual address. If dirty, the cache block is flushed to the memory stack.

The energy model computes the energy due to dynamic power consumption by summing the contributions from the following elements: cores, caches, NoCs, memory channel

TABLE I
EVALUATED SYSTEM CONFIGURATIONS.

Baseline/Host System	
Core configuration	Arm Cortex-A15, out-of-order, 2 GHz
Number of cores	16
L1 cache	32 KB I&D private
L2 cache	256 KB private
L3 cache	8 MB shared
Memory	32 GB
Wide-Parallel System	
Core configuration	Arm Cortex-A7, in-order, 1.2 GHz
Number of cores	128
Caches & memory	same as baseline
Near-Memory System	
Host cores & caches	same as baseline
Near-memory cores	Arm Cortex-A7, in-order, 1.2 GHz
Number of cores	128
L1 cache	32 KB I&D private
Memory	32 GB

transceivers, logic layer of the memory stack (without the near-memory processing elements), and the memory stack DRAM layers. Processor core energy (both for Cortex-A7 and Cortex-A15) is computed using energy per instruction measurements provided by [11] and instruction counts from `gem5`. Cache and NoC energy is modeled using McPAT assuming a 28 nm process node.

The memory subsystem is based upon an HMC model, using energy data provided by [5]. The total memory stack energy is 10.38 pJ/bit accessed. Of this, the DRAM layers consume 3.7 pJ/bit and the logic layer consumes 6.78 pJ/bit (of which, 43% is consumed by the transceivers) [9].

VI. EVALUATION

In this section, we examine how data integration workloads benefit from the near-memory processing system described above. We compare both performance and energy consumption for the baseline, wide-parallel, and near-memory target systems.

A. Performance Improvement

We first examine performance improvement by showing speedup relative to the baseline in Figure 3. The applications to the left of *2bit_fa* on the graph are above the dashed line in Figure 1(b) and those to the right of *2bit_fa* are below the line.

Examining the middle bar for each application (the speedup of the wide-parallel system), we observe that all of applications improve. While the geometric mean speedup (for all applications) is $3.04\times$ and the minimum speedup is $2.48\times$, 5 of the applications exceed $3.4\times$ speedup. We can conclude that the abundant parallelism in the data integration workloads can effectively be exploited by a larger number of individually less-powerful cores, and that the effectiveness of this approach is strong for all of the applications.

Note that while this wide-parallel system is used to assess the degree to which parallel execution can benefit data integration applications, it does not represent a realistically viable system in any practical sense for general workloads. At this scale, cache coherence overheads are often dominant, a fact that isn't an issue here simply because the data integration applications are, in effect, embarrassingly parallel, so they generate minimal coherence traffic.

The right-most bar for each application indicates the speedup for the target near-data processing architecture relative to the baseline architecture. During this execution, the host cores are essentially idle (only responsible for startup and termination) and therefore available to execute other applications such as the data analysis task that is downstream of data integration in the workflow of interest.

Again, the overall results reflect significant performance improvement. The geometric mean speedup is $3.46\times$, and the individual application speedup ranges from $2.57\times$ up to $5.82\times$. The 4 applications with the largest memory/ops ratio exhibit the greatest performance improvement relative to the wide-parallel system, each improving an additional $1.21\times$ or more relative to the wide-parallel system. In other words, they benefit from the cores' close proximity to memory and are not hurt by the lack of L2 and L3 caches associated with each near-memory core.

The outlier here is *edgelist_csr*. Its performance is slightly worse ($0.99\times$) when transitioning from the wide-parallel system to the near-memory system, mostly due to the lack of L2 and L3 caches for the near-memory cores. This however is not surprising and predicted by its outlier position in Figure 1(a).

Across the board, we see fairly good performance gains for the near-memory processing architecture by leveraging highly parallel near-memory computing units and accounting for the impact of small memory-side caches. It is also worth noting that a large fraction of this performance gain is attributable to the benefits of parallelism and a smaller fraction due to the benefits of the processor cores' physical proximity to the memory.

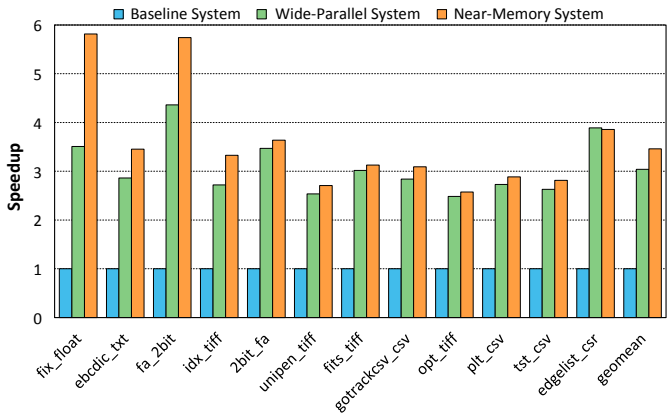

Fig. 3. Speedup of data integration workloads for wide-parallel and near-memory system relative to baseline.

We next return to the point that, in the near-memory

system, the host cores are essentially idle while the data integration application is executing, freeing them up for other tasks. For example, when executing the initial data analysis algorithm described by Malicevic et al. [6], three-quarters of the execution time is consumed by pre-processing (i.e,. data integration) and one-quarter of the execution time is consumed by the algorithm (i.e., data analysis). If the data integration is accelerated by a factor of 3 (less than the geometric mean of our benchmark applications), the overall memory bandwidth is underutilized (TSV utilization is no more than 5% across the board), and the data analysis is executed concurrently with data integration (in a pipelined manner), then the overall performance gain is a factor of 4. In other words, the entire execution time of the data integration is overlapped by the data analysis time. Given that the near-data cores were integrated onto otherwise unused chip area on the logic layer of the stacked memory, this is almost "data integration for free."

B. Energy Consumption

We next quantitatively examine the energy consumption benefits of the near-memory architecture. Figure 4 shows the energy savings for the wide-parallel and the near-memory architectures. The relative energy improvement (number above each bar) is computed by dividing the baseline system energy consumption by the wide-parallel system energy or the near-memory processing system energy, respectively.

Focusing first on the comparison between the baseline system and the near-memory target system, we see that the overall energy reduction is quite significant, with a geometric mean of $4.23\times$. Even the application with the least benefit, *2bit_fa*, requires just over one-third of the energy of the baseline system, for an energy savings of almost $3\times$ when executed on the near-memory system.

Turning our attention to how that energy savings is attributable to the wide-parallel aspects of the system versus the processing proximity to the memory, we once again observe an important relationship between applications with a high memory/ops ratio and energy reduction attributable to the physical proximity of the memory. The four applications that sit above the dashed line in Figure 1(b) all have less energy savings for the intermediate wide-parallel system with improved energy savings when transitioning to the near-memory target system. The remaining applications show substantial energy savings, with the majority of that savings being attributable to the wide-parallel nature of the applications' execution.

We can discern why this is the case by examining the energy breakdown, shown in Table II. The table decomposes the energy consumption into 6 categories: cores plus L1 caches, L2 caches, L3 caches, NoC, transceivers for the memory channels, and memory stack (including both vault controllers and the DRAM chips). They are indicated as a percentage relative to the total energy.

The energy that is saved by moving from the wide-parallel design to the near-memory design is primarily energy attributed to L2 and L3 caches and memory channel transceivers. This is largest in the applications with the highest memory/ops ratio, and is much smaller in those applications with a lower memory/ops ratio.

A final observation is that, across the board, all of the applications' energy consumption is dominated by core+L1 energy, a fact that is true for all three architectures we consider. This points to a potential approach (left for future work) for executing data integration workloads that exploits alternative computational platforms, such as reconfigurable logic.

VII. Conclusion

Data integration is an important yet not well-explored bottleneck for data analysis flows. In this paper, we characterize data integration workloads based on localities and memory/arithmetic operation intensity. Our characterization reveals that most of data integration workloads have regular memory access patterns and varying computation intensity.

We find that a near-memory processing architecture can benefit data integration workloads both in terms of performance and energy consumption. Our proposed near-memory system outperforms the baseline/host system with 16 Arm Cortex-A15 cores, exhibiting an average $3.5\times$ speedup and $4.2\times$ energy efficiency improvement, by utilizing its highly parallel 128 Arm Cortex-A7 cores inside the stacked memory logic layer. In addition, by comparing the baseline system and near-memory system with an intermediate wide-parallel system, we are able to attribute benefits separately to the availability of abundant parallelism and memory proximity. While all of the applications benefit from wide-parallel execution, the benefits of memory proximity are more concentrated on applications that have a high memory/ops ratio.

We conclude that near-memory processing is a promising strategy to improve the performance and reduce energy consumption for data integration workloads. In the future, we expect that a custom near-memory accelerator tailored to these workloads will have even higher performance and lower energy requirements.

VIII. Acknowledgments

This work supported by NSF under grants CNS-1739643 and CNS-1763503.

References

[1] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan *et al.*, "Google workloads for consumer devices: Mitigating data movement bottlenecks," in *Proc. of 23rd Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, 2018, pp. 316–331.

[2] A. M. Cabrera, C. J. Faber, K. Cepeda, R. Derber, C. Epstein, J. Zheng, R. K. Cytron, and R. D. Chamberlain, "DIBS: A data integration benchmark suite," in *Proc. of ACM/SPIE Int'l Conf. on Performance Engineering Companion*, Apr. 2018, pp. 25–28.

[3] M. Drumond, A. Daglis, N. Mirzadeh, D. Ustiugov, J. Picorel, B. Falsafi, B. Grot, and D. Pnevmatikatos, "The Mondrian Data Engine," in *Proc. of 44th ACM International Symposium on Computer Architecture*, Jun. 2017, p. 639–651.

[4] J. Jang, J. Heo, Y. Lee, J. Won, S. Kim, S. J. Jung, H. Jang, T. J. Ham, and J. W. Lee, "Charon: Specialized near-memory processing architecture for clearing dead objects in memory," in *Proc. of 52nd IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 726–739.

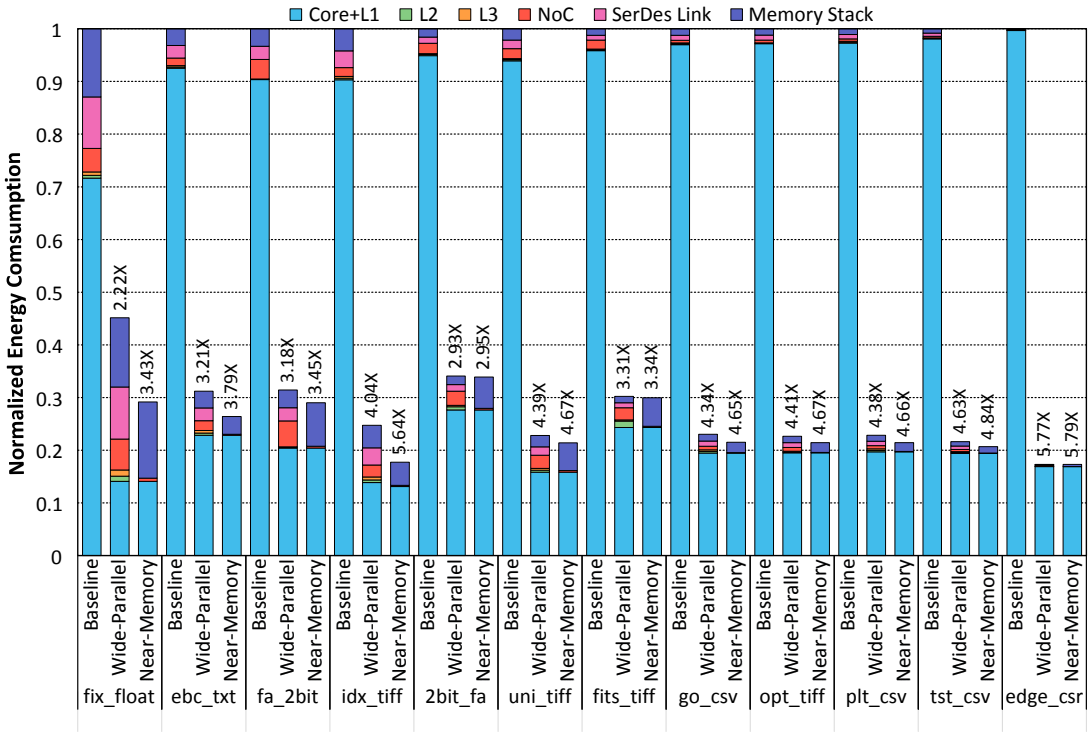

Fig. 4. Energy consumption of data integration workloads obtained from the baseline, wide-parallel and near-memory system , normalized to the baseline.

[5] J. Jeddeloh and B. Keeth, "Hybrid memory cube new DRAM architecture increases density and performance," in *Proc. of Symposium on VLSI Technology*, 2012, pp. 87–88.

[6] J. Malicevic, B. Lepers, and W. Zwaenepoel, "Everything you always wanted to know about multicore graph processing but were afraid to ask," in *Proc. of USENIX Annual Technical Conference*, Jul. 2017, pp. 631–643.

[7] I. B. Peng, J. S. Vetter, S. Moore, R. Joydeep, and S. Markidis, "Analyzing the suitability of contemporary 3D-stacked PIM architectures for HPC scientific applications," in *Proc. of 16th ACM International Conference on Computing Frontiers*, 2019, pp. 256–262.

[8] A. Pourhabibi, S. Gupta, H. Kassir, M. Sutherland, Z. Tian, M. P. Drumond, B. Falsafi, and C. Koch, "Optimus Prime: Accelerating data transformation in servers," in *Proc. of 25th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, 2020, p. 1203–1216.

[9] S. H. Pugsley, J. Jestes, H. Zhang, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, A. Davis, and F. Li, "NDC: Analyzing the impact of 3D-stacked memory+logic devices on MapReduce workloads," in *Proc. of IEEE International Symposium on Performance Analysis of Systems and Software*, 2014, pp. 190–200.

[10] G. Singh, L. Chelini, S. Corda, A. J. Awan, S. Stuijk, R. Jordans, H. Corporaal, and A.-J. Boonstra, "Near-memory computing: Past, present, and future," *Microprocessors and Microsystems*, vol. 71, Nov. 2019.

[11] E. Vasilakis, "An instruction level energy characterization of Arm processors," *Foundation of Research and Technology Hellas, Inst. of Computer Science, Tech. Rep. FORTH-ICS/TR-450*, 2015.

[12] J. Weinberg, M. O. McCracken, E. Strohmaier, and A. Snavely, "Quantifying locality in the memory access patterns of HPC applications," in *Proc. of the ACM/IEEE Conference on Supercomputing*, 2005.

TABLE II
ENERGY CONSUMPTION OF DATA INTEGRATION WORKLOADS, EXPRESSED AS A FRACTION.

		Core+L1	L2	L3	NoC	SerDes Link	Memory Stack
fix_float	Baseline	71.6%	0.6%	0.6%	4.4%	9.8%	13.0%
	Wide-Parallel	31.2%	2.2%	2.5%	13.0%	22.0%	29.1%
	Near-Memory	48.3%	0.0%	0.0%	2.0%	0.0%	49.7%
ebc_txt	Baseline	92.5%	0.2%	0.3%	1.4%	2.4%	3.2%
	Wide-Parallel	73.2%	1.3%	1.5%	6.0%	7.7%	10.2%
	Near-Memory	86.5%	0.0%	0.0%	0.7%	0.0%	12.7%
fa_2bit	Baseline	90.3%	0.1%	0.0%	3.7%	2.5%	3.3%
	Wide-Parallel	65.0%	0.5%	0.2%	15.7%	8.0%	10.6%
	Near-Memory	70.4%	0.0%	0.0%	1.2%	0.0%	28.4%
idx_tiff	Baseline	90.3%	0.3%	0.4%	1.7%	3.2%	4.2%
	Wide-Parallel	56.0%	2.0%	2.4%	9.1%	13.1%	17.4%
	Near-Memory	73.9%	0.0%	0.0%	1.4%	0.0%	24.7%
2bit_fa	Baseline	94.9%	0.2%	0.1%	2.0%	1.2%	1.6%
	Wide-Parallel	81.1%	1.8%	0.8%	7.9%	3.6%	4.8%
	Near-Memory	81.5%	0.0%	0.0%	0.9%	0.0%	17.6%
uni_tiff	Baseline	93.9%	0.2%	0.2%	1.9%	1.6%	2.1%
	Wide-Parallel	69.5%	1.5%	1.6%	10.9%	7.1%	9.4%
	Near-Memory	74.0%	0.0%	0.0%	1.3%	0.0%	24.7%
fits_tiff	Baseline	95.8%	0.2%	0.1%	1.7%	0.9%	1.2%
	Wide-Parallel	80.4%	4.1%	0.7%	7.6%	3.1%	4.1%
	Near-Memory	81.1%	0.0%	0.0%	0.8%	0.0%	18.1%
go_csv	Baseline	97.0%	0.2%	0.2%	0.5%	0.9%	1.2%
	Wide-Parallel	84.6%	1.2%	1.4%	3.0%	4.2%	5.6%
	Near-Memory	90.5%	0.0%	0.0%	0.4%	0.0%	9.1%
opt_tiff	Baseline	97.2%	0.1%	0.1%	0.5%	0.9%	1.2%
	Wide-Parallel	86.1%	0.6%	0.7%	3.1%	4.1%	5.4%
	Near-Memory	91.1%	0.0%	0.0%	0.3%	0.0%	8.6%
plt_csv	Baseline	97.3%	0.1%	0.2%	0.5%	0.8%	1.1%
	Wide-Parallel	86.2%	1.1%	1.3%	2.7%	3.7%	5.0%
	Near-Memory	91.8%	0.0%	0.0%	0.4%	0.0%	7.8%
tst_csv	Baseline	98.1%	0.1%	0.1%	0.3%	0.6%	0.8%
	Wide-Parallel	89.7%	0.7%	0.8%	2.0%	2.9%	3.9%
	Near-Memory	93.9%	0.0%	0.0%	0.3%	0.0%	5.9%
edge_csr	Baseline	99.7%	0.0%	0.0%	0.2%	0.1%	0.1%
	Wide-Parallel	97.4%	0.2%	0.1%	1.4%	0.3%	0.5%
	Near-Memory	97.6%	0.0%	0.0%	0.1%	0.0%	2.3%