Using Isoefficiency as a Metric to Assess Disaggregated Memory Systems for High Performance Computing

Anusha Devulapally
The Pennsylvania State University
USA
akd5994@psu.edu

Mahantesh Halappanavar
Pacific Northwest National Laboratory
USA
hala@pnnl.gov

Amit Puri
Indian Institute of Technology Guwahati
India
amitpuri@iitg.ac.in

Vijaykrishnan Narayanan
The Pennsylvania State University
USA
vijaykrishnan.narayanan@psu.edu

Andres Marquez
Pacific Northwest National Laboratory
USA
Andres.Marquez@pnnl.gov

ABSTRACT

Memory disaggregation is an approach to decouple compute and memory to minimize the total cost of ownership. However, analytical methods to study the impact of this approach are not readily available for high performance computing use cases. In this position paper, we propose *isoefficiency* as an approach to analytically demonstrate the classes of algorithms that would benefit from disaggregated memory technologies as we scale to a larger number of processors. Isoefficiency of an algorithm is given by a function $N(p)$ that measures the degree to which the problem size needs to increase with p (number of processors) to maintain a constant efficiency. We evaluate isoefficiency using sparse general matrix–matrix multiplication (SpGEMM) on a 2-socket shared-memory system and the simulation of a CXL-based disaggregated system. Our results support the suitability of isoefficiency for evaluating disaggregated memory systems with different design choices in conjunction with different application kernels.

KEYWORDS

CXL, disaggregated memory, isoefficiency, performance modeling

1 INTRODUCTION

Traditional cloud computing (data center) resources over-provision resources in order to enable efficient execution of applications, leading to underutilization of resources for many applications. *Memory disaggregation* is a paradigm where memory resources are decoupled from the compute nodes and are pooled together to be dynamically shared across multiple compute nodes [9] (detailed in §2). Disaggregated systems can potentially benefit resource sharing by heterogeneous workloads and memory-intensive applications such as database systems, graph analytics, artificial intelligence and machine learning applications [10, 23].

The emergence of compute-intensive generative artificial intelligence and rapid adoption of cloud computing has made memory disaggregation an important technology. However, memory disaggregation is a complex problem and a wide variety of design choices necessitate a coherent metric for evaluation. From the perspective of high performance computing with single application in mind, we propose *isoefficiency* as an important metric to compare different systems [12]. Isoefficiency is a critical metric that not only facilitates direct comparisons between different systems but also provides a comprehensive evaluation by considering algorithmic design and input characteristics. This metric is particularly significant for disaggregated memory systems, where the balance between resource provisioning and computational demand is crucial.

By focusing on the scalability of a system as more compute resources are added, isoefficiency captures the essential strong and weak scaling behaviors that these systems are expected to exhibit. As such, it serves as an essential tool in evaluating and optimizing disaggregated memory architectures, ensuring they meet the performance requirements of memory-intensive applications. Intuitively, isoefficiency measures the rate at which the input sizes need to grow relative to the growth in computing resources in order to keep parallel efficiency constant (detailed in §3).

Disaggregated memory systems are built by physically separate pools of memory that are accessible on-demand to the computing nodes over fabric attached memory (FAM). Compute Express Link (CXL version 3.0) is a PCIe-based protocol that supports switched cache-coherent memory pooling and memory sharing [11]. CXL as a FAM stand-in is interesting due to its wide industrial adoption, providing memory capacity expansion and system memory bandwidth increase with a reasonable latency penalty, making it important for high memory demand applications. In this position paper, we focus on the disaggregated memory systems using CXL technology in our simulation-based empirical evaluation (detailed in §4 and §5).

Contributions: Our contributions are as follows:

- Propose isoefficiency as a unified metric to evaluate disaggregated memory systems, for both individual and relative comparisons of systems.
- Propose combinatorial approach for sparse general matrix multiplication (SpGEMM) to demonstrate the efficacy of isoefficiency as a viable metric for quantitative evaluation.
- Perform empirical analysis using simulation of CXL-based system and representative and scalable synthetic inputs.

Since disaggregated-memory systems are nascent but rapidly evolving, we believe that development of efficient metrics for evaluation will enable consistent and comprehensive evaluation of different

systems and design choices, as well as in identifying specific applications that can benefit from disaggregated systems.

2 PRELIMINARIES AND RELATED WORK

The demand for computing has been growing at an exponential rate [19], especially due to the rapid adoption of artificial intelligence. Consequently, the need for memory capacity and bandwidth increase proportionately. However, traditional approaches of provisioning memory close to compute do not scale and lead to inefficient use of memory due to stranding and limitations in fine-grained data sharing in distributed systems. These limitations can be effectively addressed by *memory pooling*. However, providing coherent access to pooled memory is challenging. The Compute Express Link (CXL®) is an open industry-standard that builds on the Peripheral Component Interconnect-Express® (PCI-Express®or PCIe®) and provides protocols for coherent memory accesses between processors and computing devices such as CPUs, accelerators (GPUs, FPGAs, etc.), memory buffers, persistent memory, and solid-state drives. We refer you to authoritative documents on the exact details [6] provided by the three generations of CXL that specify the interconnect and multiple protocols, spanning from a single machine to 100s of machines [1, 7]. In this section, we only provide a brief survey of current literature since our goal is to propose a metric for evaluation and not a direct evaluation of CXL based systems.

Gouk *et al.* used CXL's memory protocol (`cxl.mem`) to directly connect host processor and memory resources without the overhead of data copying like RDMA [11]. They present DRAM-like performance when the workload resides in host-processor's cache. Wang *et al.* use a memory pooling system to combine RDMA and CXL to achieve memory disaggregation [21], whereas, Li *et al.* also use a memory pooling system that significantly reduces DRAM costs using CXL for cloud computing platforms [15]. Sun *et al.* provide performance evaluation and theoretical analysis of a CXL-ready system relative to NUMA-based CXL emulators demonstrating the advantages of such systems on various latency and throughput sensitive applications [20]. Lerner and Alonso provide an empirical study demonstrating the advantages of CXL on scale-up database architectures [14]. However, they do not provide details about performance gain, complexity or overhead of using CXL.

Performance analysis is an expansive field on its own [13]. In this position paper, we propose to use isoefficiency of Grama *et al.*, which provides a theoretical framework for evaluation of system scalability for specific algorithms and inputs by quantifying the relation between problem size and number of processors while keeping parallel efficiency a constant [12]. Isoefficiency metrics have been used in various applications, such as unbalanced workloads in parallel systems [2] and on big data frameworks like Hadoop and Spark [18]. On particular relevance in this context is the Roofline model of Williams *et al.* [22], which provides a visualization of performance by plotting arithmetic intensity (floating-point performance) as a function of theoretical peak performance and peak bandwidth of a specific architecture. Ding *et al.* provide a framework to evaluate disaggregated memory systems using Roofline model, emphasizing on the improved memory utilization and flexibility for various HPC workloads. Their analysis shows that most of the applications do not suffer from performance degradation when using disaggregated memory [8]. Unlike the Roofline model, our goal is to evaluate scalability of algorithm-architecture combinations for different problems without requiring to consider a vast number of possible combinations. This is where isoefficiency comes to fore.

3 ISOEFFICIENCY AS A METRIC

In this position paper, we posit that isoefficiency is a better metric to evaluate systems and system designs with disaggregated memory. Isoefficiency provides a consistent metric to measure the relative performances of different systems across three major components: (i) hardware parameters and system (or design) configurations, (ii) algorithmic design choices (computation, communication, etc.), and (iii) input characteristics. Relative performances can be measured with a single metric while varying across different components when the systems scale with number of processing units and other components. Since isoefficiency focuses on the scalability of a system when computing resources are added to the system, it provides a unified metric for relative comparisons.

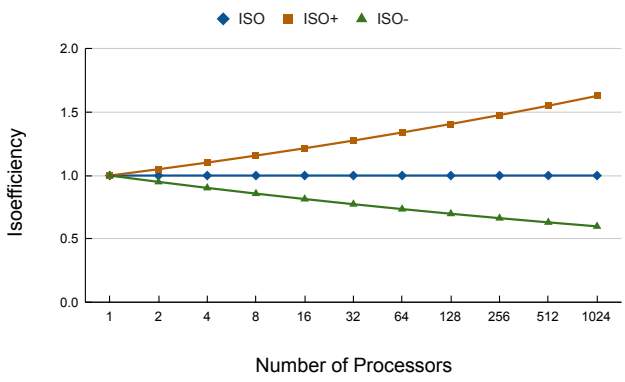

Figure 1: Illustration of isoefficiency – perfect (in blue), 10% super-efficient (orange) and 10% sub-efficient (green).

Introduced by Grama *et al.*, [12] isoefficiency evaluates the scalability of parallel systems and algorithms by analyzing the required growth in problem size N relative to the number of processors P to maintain a constant efficiency E. Intuitively, a proportional growth in work with growth in number of processors is considered as an isoefficient system. Efficiency E is given as the ratio of the speedup S to the number of processors P, where speedup S is the ratio of the execution time on a single processor T_s to the execution time on p processors T_p. The isoefficiency $E(p)$ determines the rate at which the problem size must increase to keep efficiency a constant, as described by the equation $N = K.E(p)$, where K is constant. This function provides insights on the scalability of a parallel system. We provide a hypothetical illustration for a problem that scales with number of processors in Figure 1. The three lines in the figure depict when efficiency scales linearly, super-linearly and sub-linearly.

4 EVALUATING ISOEFFICIENCY

In this section we detail our approach for computing isoefficiency for traditional shared-memory systems and systems with disaggregated memory. We perform the comparisons using sparse general matrix–matrix multiplication (SpGEMM) as the computational kernel. For enabling different memory footprints, we use R-MAT as the generative model for building the matrices. We perform experiments on a traditional 2-socket shared-memory system and a simulator for disaggregated memory system.

4.1 Algorithm: Sparse matrix multiplication

Our goal was to select an algorithm that would not only stress the memory system but also include sufficient floating-point operations. We therefore selected sparse general matrix–matrix multiplication (SpGEMM) as the first algorithm for demonstrating that isoefficiency is a viable metric for evaluating systems. Since our goal is not to implement the best algorithm for SpGEMM, we adopt a simplistic but work-efficient approach for SpGEMM using the combinatorial approach of Brualdi and Cvetkovic [3].

Given two sparse matrices A and B, we build a special data structure called a König digraph, $G = (W \cup R \cup K, E_1 \cup E_2)$. Each row in A is represented with a "white" vertex (W) in G. The "gray" vertices (R) represent both the columns of A and rows of B, and the "black" vertices (K) represent the columns in B. The nonzero elements in A and B are represented by directed and weighted edges in E_1 and E_2 respectively. A simple 3×3 matrix example is provided in Fig. 2. The nonzero values in $C = A \cdot B$ can be computed by performing two-hop walks starting from vertices in W and ending with vertices in K. For example, the nonzero value for $C(1, 2)$ in Fig. 2, will consist of a traversal from $w_1 \rightarrow r_1 \rightarrow k_2$ and $w_1 \rightarrow r_3 \rightarrow k_2$. The time complexity of this algorithm can be expressed as $O(n\Delta^2)$, where n is the number of rows in A and Δ is the maximum degree for vertices in W and R. While this is a loose bound, we note that the amount of work performed is proportional to the number of nonzeros in A and B.

Parallel Implementation: We implement SpGEMM in C++ using OpenMP with all data structures using 64-bit precision. We use a compressed sparse representation for Kŏnig digraph G, which provides efficient access to nonzeros for each row and column in A and B respectively. Unlike a serial implementation, parallel SpGEMM is nontrivial without first constructing the nonzeros in C, assuming that the structure of C is dense (either fully or in blocks). In our implementation, we first perform a symbolic walk to determine the exact location of each non-zero value in C, and then perform the actual computation. Due to the nature of our implementation, which is not highly optimized, the construction of C takes considerably more time than calculating the values of C, and does not scale well. We present the results in §5.

4.2 Input: Recursive-Matrix Traversal (R-MAT)

The input graphs are generated using the "recursive matrix" (R-MAT) graph generator of Chakrabarti *et al.* [5]. Consider a graph $G(V, E)$, where V are vertices and $|E|$ are number of edges. The adjacency matrix of G consists of V nodes ($V \times V$) matrix. This algorithm works by recursively sub-dividing the adjacency matrix of the generated into four quadrants and distributes the edges within these quadrants with specified probabilities. The distribution is dependent on the non-negative probabilities (a, b, c, d) whose sum equals to one. We consider three sets of probabilities ([4]) – RMAT-ER: $(0.25, 0.25, 0.25, 0.25)$, RMAT-G: $(0.45, 0.15, 0.15, 0.25)$ and RMAT-B: $(0.55, 0.15, 0.15, 0.15)$ in our experiments. We present results for RMAT-ER and RMAT-G. We only performed simulations with RMAT-ER.

The algorithm starts with an empty adjacent matrix, and an edge is placed in the matrix by choosing one of the four quadrants based on the (a, b, c, d) probabilities. The chosen quadrant is further divided into four smaller quadrants and this process repeats until the quadrant is of size $1x1$, where an edge is placed. This edge placement is repeated $|E|$ times to create the desired graph G. In our experiments, we set $|E| = 8 \times |V|$. The sparsity structure of RMAT-ER can be observed that each row has roughly the same number of nonzeros (normal distribution) with an equal probability to have any of the columns as its neighbor in G, and there, a highly random structure of neighborhoods. RMAT-G and RMAT-B display skewed normal distributions and unstructured neighborhoods. In other words, the R-MAT generator does not produce graphs with a good community structure [4].

4.3 Simulator: Disaggregated Memory System

We use `DRackSim` [16] simulator to experiment with a CXL-enabled disaggregated memory system. Although `DRackSim` can simulate a large-scale system, we configure it for one compute and remote memory node. The compute node models a multi-socket CPU system with an out-of-order pipeline at compute cores and a multi-level cache hierarchy for CPU-driven memory simulation. The memory manager is configured to allocate memory at a 50:50 ratio in local and remote memory, alternatively for each page. In the simulation, the remote memory node is connected to the compute node through a 10, 30 or 50 Gbps interconnect having 35ns of total latency for (de)packetization/processing. The interconnect uses a queue-based modeling that takes 64-byte memory request packets from the compute node and forwards them to the memory node. In a large-scale CXL3.0-enabled system, a compute node will only get a small share of the switch bandwidth. Therefore, we perform simulations at multiple interconnect bandwidths to validate the proposed metric for changing resource availability in a real system. However, the remote memory access at the memory node is always performed at cache line granularity (64B), which sends a 128-byte response packet, including cache line data. The DRAM simulation at local and remote memory is handled by integrating `DRAMSim2` [17], which instantiates multiple 2400MHz DDR4 (19.2Gbps) components each for local and remote memory. The thread context switching time is not considered in the simulator.

4.4 Traditional: 2-socket Shared Memory Node

The 2-socket shared-memory system that we used in this work is comprised of AMD EPYC 7282 16-core processor, utilizing the x86_64 architecture. It supports both 32-bit and 64-bit operation modes with a little-endian byte order. The system has 32 CPUs, organized as 16 cores per socket across 2 sockets, with each core operating on single thread, and two NUMA nodes(node 0: CPUs 0-15, node 1: CPUs 16-31). The processor runs at a frequency of

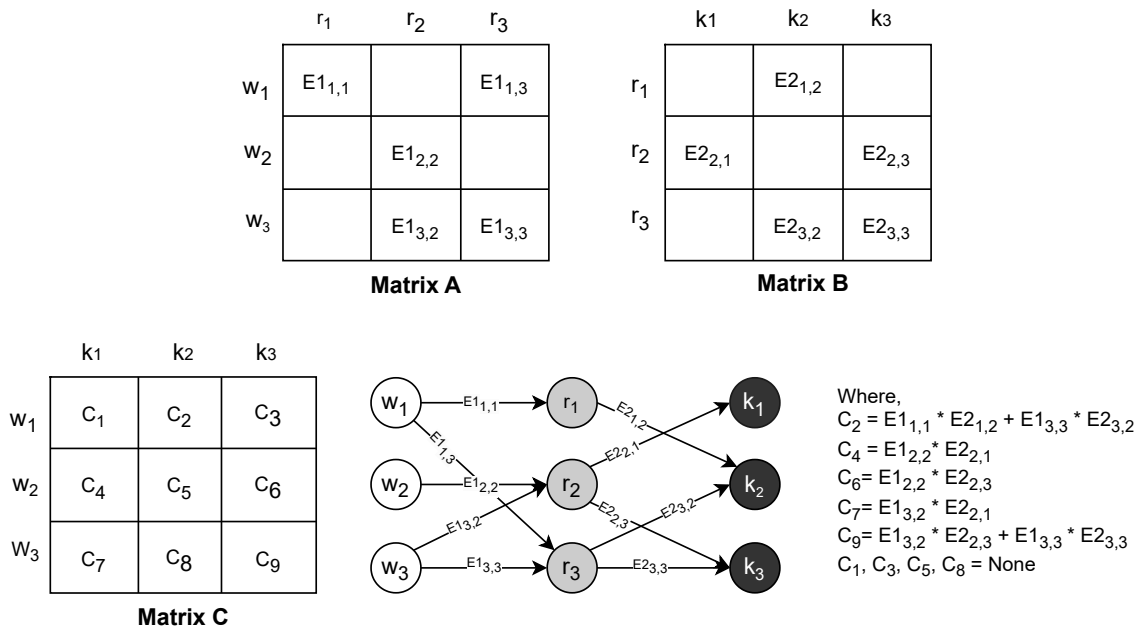

Figure 2: Illustration of sparse general matrix–matrix multiplication (SpGEMM) using the combinatorial approach.

approximately 2.8 GHz with a BogoMIPS value of 5589.58, and supports AMD-V virtualization technology. The memory hierarchy includes 32KB of L1 data- and L1 instruction-cache per core, 512Kb of L2 cache per core, and 16MB of shared L3 cache.

5 EXPERIMENTAL RESULTS

We perform sparse general matrix–matrix multiplication (SpGEMM) on a 2-socket system described in §4.4. We present the time for two specific steps: (*i*) Computing the structure of matrix *C*, and (*ii*) computing the values of matrix *C*. We describe the steps in §4.1. We note that further optimizations are required for computing the structure of *C* kernel. As we observe in Table 2, parallel efficiency number are very low indicating that this step does not scale well. We also note that we do not explicitly construct the König digraph since matrices *A* and *B* are stored in row-wise and column-wise formats.

Computing the nonzero values of matrix *C* after it's compressed sparse structure is built scales well proportional to the number of processors and doubling of the number of problem size. In order to double the amount of work, we double the number of nonzeros in matrices *A* and *B*. We note that the number of nonzeros is proportional to the matrix size (we use a fixed ration of 8× as noted in §4.2). By increasing the value of *SCALE* by one, we double the matrix size (2^{SCALE}), and consequently, the problem size. Similarly, by increasing the value of *SCALE* by two quadruples the work. We note that by using the probability values of $(0.25, 0.25, 0.25, 0.25)$ for R-MAT generator, we expect to have a normal distribution for the number of nonzeros for each row (and column).

5.1 Traditional 2-Socket Computing Results:

We present results from the 2-socket experiments for building the structure of *C* and computing the values of *C*. We present the results along two components: (*i*) Different input parameters – RMAT-ER and RMAT-G, and (*ii*) two growth rates for work – doubling and quadrupling the sizes. Table 2 summarizes the isoefficiency for computing the structure of *C*, and Table 1 summarizes the isoefficiency for computing the values of *C*.

	SCALE							
Threads	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
2	0.11	0.10	0.10	0.06	0.06	0.06	0.06	0.06
4	0.18	0.16	0.14	0.11	0.12	0.11	0.11	0.11
8	0.35	0.33	0.30	0.26	0.25	0.24	0.23	0.22
16	0.63	0.62	0.61	0.56	0.52	0.46	0.43	0.44
32	1.22	1.15	1.14	1.05	1.03	0.94	0.96	0.93

Table 1: Iso-efficiency of the traditional 2-socket shared system for the task of computing matrix C. Here, problem size of the input depends on Scale, i.e., 2^{SCALE}.

We observe in Figure 4 that building the structure of *C* is not an efficient kernel due to irregular memory accesses and the lack of arithmetic operations to amortize the access costs. We can also observe that quadrupling the problem sizes is more efficient than doubling the problem sizes. Further, RMAT-ER scales better than RMAT-G since load imbalances are expected to be smaller for RMAT-ER (due to differences in degree distributions).

In contrast, calculating the nonzero values after the structure has been built parallelizes efficiently. We observe better scaling for larger problems ($SCALE \geq 22$), as summarized in Figure 4. Further,

	SCALE							
Threads	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
2	0.07	0.07	0.06	0.06	0.06	0.05	0.05	0.051
4	0.08	0.07	0.07	0.06	0.06	0.06	0.06	0.057
8	0.08	0.08	0.08	0.07	0.07	0.07	0.06	0.06
16	0.09	0.09	0.08	0.08	0.07	0.07	0.07	0.07
32	0.09	0.09	0.08	0.08	0.08	0.07	0.07	0.07

Table 2: Iso-efficiency of the traditional 2-socket shared system for the task of building the structure of C. Here, problem size of the input depends on *SCALE*, i.e., 2^{SCALE}.

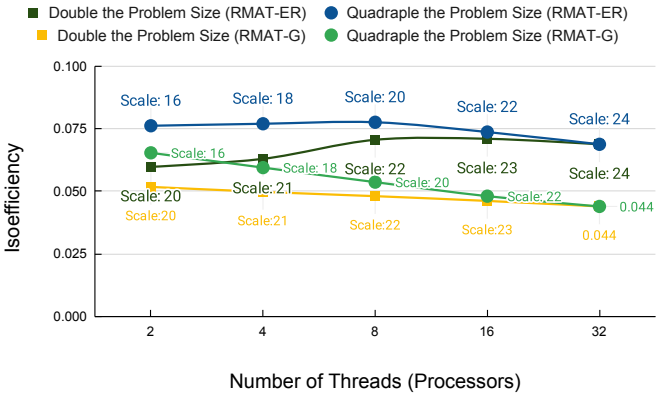

Figure 3: Isoefficiency for building the structure of *C* on traditional 2-scoket memory system.

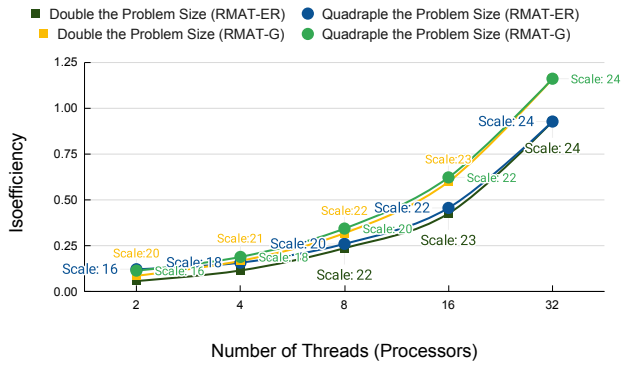

Figure 4: Isoefficiency for computing the values of *C* on traditional 2-scoket memory system.

we do not observe noticeable difference between RMAT-ER and RMAT-G. We can observe the stark differences in the isoefficiency values for the two kernels – building structure and computing values. We can therefore conclude that isoefficiency provides a unified metric for evaluating scalability of a system along different aspects — algorithm, input and hardware features.

5.2 Simulation Results

We now provide empirical evaluation for simulated results using `DRackSim` (detailed in §4.3). Since we had difficulties in simulating complete SpGEMM execution, we only present the results for computing the structure of *C* in this section. If accepted, we will present the complete results in the final version of this work. Results for two different settings of the network bandwidth are summarized in Table 3. The isoefficiency values for three different bandwidth settings are presented in Figure 5. Since the numbers for 50 Gbps and 30 Gbps are very similar, we omit them from Table 3. We can observe that the isoefficiency values successively decrease, and therefore, indicate that the growth in problem size needs to be more than linear in order to maintain a constant isoefficiency.

#Threads	*SCALE*	**M-Cycles**	**Isoefficiency**
	50 Gbps		
1	10	112	1.00
2	11	153	0.88
4	12	224	0.61
8	13	369	0.38
16	14	650	0.23
	10 Gbps		
1	10	121	1.00
2	11	166	0.36
4	12	247	0.12
8	13	402	0.04
16	14	720	0.01

Table 3: Results from the `DRackSim` simulator. Compute time is calculated in millions of cycles.

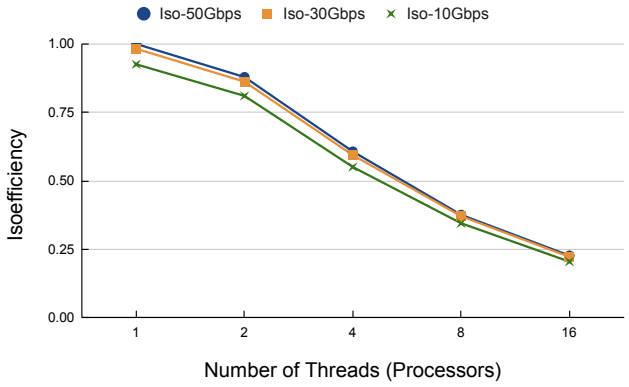

Figure 5: Isoefficiency calculated for computing the structure of *C* using the `DRackSim` simulator for different network bandwidths

6 SUMMARY AND FUTURE WORK

In this position paper, we proposed *isoefficiency* as a suitable metric to analyze the performance of disaggregated memory systems with respect to specific application workloads and input data characteristics. We used sparse general matrix–matrix multiplication (SpGEMM) as an illustrative example using synthetic matrices generated by the recursive-matrix (R-MAT) generator. We provide

results on a traditional 2-socket shared-memory system using a multithreaded implementation of a combinatorial approach for SpGEMM. We also provided preliminary results from using a simulator modeling a CXL-based disaggregated memory system.

In the future work, we plan to optimize SpGEMM algorithm and conduct extensive empirical evaluations on a 8-socket shared-memory system using inputs with varying degree distributions and application kernel. Further, we plan to perform runs on an experimental CXL-based system, which also be used to compute accurate parameters for the simulator, as well as validate the simulated performance values. We believe that this work will provide a consistent and rigorous approach to evaluate disaggregated memory systems for different applications using isoefficiency as the metric for evaluations.

7 ACKNOWLEDGEMENTS

This work was supported by the U.S. DOE Office of Science, Office of Advanced Scientific Computing Research, under award 76125: "AMAIS - Advanced Memory to support Artificial Intelligence for Science". The Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy under contract DE-AC05-76RL01830, and by PRISM, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

REFERENCES

[1] Marcos K. Aguilera, Emmanuel Amaro, Nadav Amit, Erika Hunhoff, Anil Yelam, and Gerd Zellweger. 2023. Memory disaggregation: why now and what are the challenges. *SIGOPS Oper. Syst. Rev.* 57, 1 (jun 2023), 38–46. https://doi.org/10.1145/3606557.3606563

[2] Jose L Bosque, Oscar D Robles, Pablo Toharia, and Luis Pastor. 2013. Analyzing scalability of parallel systems with unbalanced workload. *The Journal of Supercomputing* 64 (2013), 110–119.

[3] Richard A. Brualdi and Dragos M. Cvetkovic. 2008. A Combinatorial Approach to Matrix Theory and Its Applications. https://api.semanticscholar.org/CorpusID:119079649

[4] Ümit V Çatalyürek, John Feo, Assefaw H Gebremedhin, Mahantesh Halappanavar, and Alex Pothen. 2012. Graph coloring algorithms for multi-core and massively multithreaded architectures. *Parallel Comput.* 38, 10-11 (2012), 576–594.

[5] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. 2004. R-MAT: A recursive model for graph mining. In *Proceedings of the 2004 SIAM International Conference on Data Mining*. SIAM, 442–446.

[6] CXL Consortium. [n. d.]. CXL Specification.

[7] Debendra Das Sharma, Robert Blankenship, and Daniel Berger. 2024. An Introduction to the Compute Express Link (CXL) Interconnect. *ACM Comput. Surv.* (jun 2024). https://doi.org/10.1145/3669900 Just Accepted.

[8] Nan Ding, Samuel Williams, Hai Ah Nam, Taylor Groves, Muaaz Gul Awan, LeAnn Lindsey, Christopher Daley, Oguz Selvitopi, Leonid Oliker, and Nicholas Wright. 2022. Methodology for Evaluating the Potential of Disaggregated Memory Systems. In *2022 IEEE/ACM International Workshop on Resource Disaggregation in High-Performance Computing (REDIS)*. 1–11. https://doi.org/10.1109/REDIS56595.2022.00006

[9] Mohammad Ewais and Paul Chow. 2023. Disaggregated Memory in the Datacenter: A Survey. *IEEE Access* 11 (2023), 20688–20712. https://doi.org/10.1109/ACCESS.2023.3250407

[10] Sayan Ghosh, Nathan R. Tallent, Marco Minutoli, Mahantesh Halappanavar, Ramesh Peri, and Ananth Kalyanaraman. 2021. Single-Node Partitioned-Memory for Huge Graph Analytics: Cost and Performance Trade-Offs. In *SC21: International Conference for High Performance Computing, Networking, Storage and Analysis*. 01–16.

[11] Donghyun Gouk, Miryeong Kwon, Hanyeoreum Bae, Sangwon Lee, and Myoungsoo Jung. 2023. Memory Pooling With CXL. *IEEE Micro* 43, 2 (2023), 48–57. https://doi.org/10.1109/MM.2023.3237491

[12] Ananth Y Grama, Anshul Gupta, and Vipin Kumar. 1993. Isoefficiency: Measuring the scalability of parallel algorithms and architectures. *IEEE Parallel & Distributed Technology: Systems & Applications* 1, 3 (1993), 12–21.

[13] Raj Jain. 1991. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley. 720 pages.

[14] Alberto Lerner and Gustavo Alonso. 2024. CXL and the Return of Scale-Up Database Engines. *arXiv preprint arXiv:2401.01150* (2024).

[15] Huaicheng Li, Daniel S Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, et al. 2023. Pond: Cxl-based memory pooling systems for cloud platforms. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 574–587.

[16] Amit Puri, Kartheek Bellamkonda, Kailash Narreddy, John Jose, Venkatesh Tamarapalli, and Vijaykrishnan Narayanan. 2024. DRackSim: Simulating CXL-enabled Large-Scale Disaggregated Memory Systems. In *Proceedings of the 38th ACM SIGSIM Conference on Principles of Advanced Discrete Simulation* (, Atlanta, GA, USA,) *(SIGSIM-PADS '24)*. Association for Computing Machinery, New York, NY, USA, 3–14. https://doi.org/10.1145/3615979.3656059

[17] Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob. 2011. DRAMSim2: A Cycle Accurate Memory System Simulator. *IEEE Computer Architecture Letters* 10, 1 (2011), 16–19. https://doi.org/10.1109/L-CA.2011.4

[18] David Sanchez, Oswaldo Solarte, Victor Bucheli, and Hugo Ordonez. 2018. Evaluating the scalability of big data frameworks. *Scalable Computing: Practice and Experience* 19, 3 (2018), 301–307.

[19] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. 2020. Green AI. *Commun. ACM* 63, 12 (nov 2020), 54–63. https://doi.org/10.1145/3381831

[20] Yan Sun, Yifan Yuan, Zeduo Yu, Reese Kuper, Chihun Song, Jinghan Huang, Houxiang Ji, Siddharth Agarwal, Jiaqi Lou, Ipoom Jeong, et al. 2023. Demystifying cxl memory with genuine cxl-ready systems and devices. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 105–121.

[21] Zhonghua Wang, Yixing Guo, Kai Lu, Jiguang Wan, Daohui Wang, Ting Yao, and Huatao Wu. 2024. Rcmp: Reconstructing RDMA-Based Memory Disaggregation via CXL. *ACM Transactions on Architecture and Code Optimization* 21, 1 (2024), 1–26.

[22] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (apr 2009), 65–76. https://doi.org/10.1145/1498765.1498785

[23] Daniel Zahka and Ada Gavrilovska. 2022. FAM-Graph: Graph Analytics on Disaggregated Memory. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 81–92. https://doi.org/10.1109/IPDPS53621.2022.00017