Performance Modeling and Evaluation of a Production Disaggregated Memory System

Ning Zhang
Computer Science Department, Illinois Institute of Technology, Chicago, USA
nzhang23@hawk.iit.edu

Brian Toonen
Argonne Leadership Computing Facility, Argonne National Laboratory, Chicago, USA
toonen@alcf.anl.gov

Xian-He Sun
Computer Science Department, Illinois Institute of Technology, Chicago, USA
sun@iit.edu

William Allcock
Argonne Leadership Computing Facility, Argonne National Laboratory, Chicago, USA
toonen@alcf.anl.gov

ABSTRACT

High performance computers rely on large memories to cache data and improve performance. However, managing the ever-increasing number of levels in the memory hierarchy becomes increasingly difficult. The Disaggregated Memory System (DMS) architecture was introduced in recent years for better memory utilization. DMS is a global memory pool between the local memories and storage. To leverage DMS, we need a better understanding of its performance and how to exploit its full potential. In this study, we first present a DMS performance model for performance evaluation and analysis. We next conduct a thorough performance evaluation to identify application-DMS characteristics under different system configurations. Experimental tests are conducted on the RAM Area Network (RAN), a DMS implementation available at the Argonne National Laboratory, for performance evaluation. Then, the results of performance experiments are presented along with an analysis of the pros and cons of the RAN-DMS design and implementation. The counterintuitive performance results for the K-means application are analyzed at code-level to illustrate DMS performance. Finally, based on our findings, we present some discussions on future DMS design and its potential on AI applications.

CCS CONCEPTS

• **Computer systems organization**; • **Architectures**; • **Parallel architectures**; • **Multicore architectures**;

KEYWORDS

Performance Modeling, Disaggregated Memory, C-AMAT, Performance Evaluation, Utilization, RAN

ACM Reference Format:
Ning Zhang, Brian Toonen, Xian-He Sun, and William Allcock. 2020. Performance Modeling and Evaluation of a Production Disaggregated Memory System. In *The International Symposium on Memory Systems (MEMSYS 2020), September 28–October 01, 2020, Washington, DC, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3422575.3422795

MEMSYS 2020, September 28–October 01, 2020, Washington, DC, USA

ACM ISBN 978-1-4503-8899-3/20/09…$15.00
https://doi.org/10.1145/3422575.3422795

1 INTRODUCTION

Larger memory is under demand in High Performance Computing (HPC) due to the sustained development of exascale computing, big data applications and container techniques [1]. In the meantime, the influence of memory on the overall performance and power consumption of HPC systems has also increased significantly [2–4]. Memory performance is application dependent and varies with workload inputs and data characteristics [5]. On HPC systems, this variability in applications and workloads can lead to low memory performance as well as poor memory utilization [6]. Fig. 1 presents the distributions of the average and maximum memory usage across all the nodes of the Argonne's Cooley computer cluster [7, 8] (384GB memory for each node) in 2018. Here the memory usage of each running job is monitored every minute. Fig. 1 shows, while the overall memory utilization is very low (around 25GB on average, shown by the blue violin), sometimes the memory of several nodes near full utilization (nearly 384GB, shown by the red violin). There is a dilemma in memory system design of computer clusters. On one hand, sometime some nodes do need a large memory capacity. On the other hand, most of the time most of the nodes do not need large memory capacity. We need to reconsider memory system design for better memory utilization, costs, power consumption, as well as performance [5, 6]. This dilemma motivates the idea of disaggregated memory [1].

Disaggregated memory system (DMS) decouples memory and computing to improve memory utilization [6]. DMS adds a (remote) global memory layer to expand the memory capacity of compute nodes. The global memory usually consists of a large pool of memory, a memory controller, and a network interface to communicate with compute nodes. Transparent memory access can be handled by operating system or hypervisor extensions to maintain the local abstraction of the remote memory [1, 7, 9, 10]. Network technology advances faster than memory technology. Modern network technologies such as Fiber-optic, Myrinet and InfiniBand enable low latency (e.g., a few microseconds) and high bandwidth (e.g., 800 Gbps) communication, which brings us new opportunities to design and implement efficient DMS.

DMS separates the upgrade and scaling of processors and memories and has a potential for better memory utilization. However,

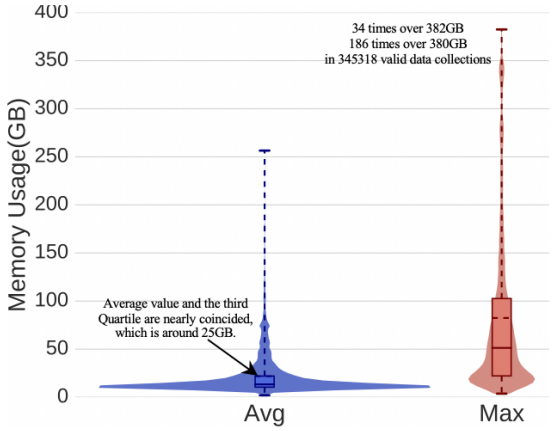

Figure 1: Memory usage distribution of Cooley in 2018

DMS does not remove the "Memory Wall" [11] problem automatically. DMS needs to be modeled and utilized to mitigate the memory-wall impact. There are two issues. First, DMS is an additional layer in the memory hierarchy. What its influence on the overall memory system is the major concern, rather than its individual performance. So, we need a hierarchical recursive memory system model such as the AMAT (Average Memory Access Time) [12] mode. Second, DMS is designed for sharing. Concurrent data access is a vital concern of DMS. The traditional data access model, the AMAT [12] model, is designed for sequential data access [13]. So, AMAT needs to be extended to consider concurrent data access. Fortunately, a recent work, the C-AMAT (Concurrent-AMAT) [14, 15] model, has extended the AMAT model to consider the integrated impact of data locality, data concurrency and data access overlapping. C-AMAT is a good choice for performance modeling of DMS.

While C-AMAT is designed to measure concurrent data accesses, it is a relatively new model. How to measure C-AMAT in an actual multicore environment, where each core has its own individual private L1 and L2 catch, has never explored. In this study, we first extend C-AMAT to multicore architectures and introduce mechanisms to measure the DMS multicore C-AMAT on the Cooley computer cluster [7, 8]. Next, we conduct a thorough performance evaluation on Cooley using different benchmarks with different memory access patterns. The counterintuitive performance of the K-means benchmark (see Section 4) is analyzed in-depth for a better understanding of DMS. This study makes the following contributions:

1. The C-AMAT model is extended to multicore architectures and is applied to measure DMS performance on an Argonne DMS machine.
2. A thorough performance testing is conducted. The Argonne RAN-DMS system is tested and studied with diverse applications.
3. An in-depth, code-level performance analysis and evaluation are carried out to understand and utilize the performance of the K-means application.
4. The potential and limitations of RAN are carefully studied. Results show that DMS, as a shared, global layer of a memory hierarchy, has many unique properties which need more investigation.

The rest of the paper is organized as follows. Section 2 reviews the background. Section 3 describes the performance modeling of DMS and its measurement methodology on real hardware. Section 4 provides the conducted performance evaluations and analyses of the findings. Section 5 shows the related work. Finally, Section 6 concludes the work.

2 BACKGROUND

In this section, we motivate DMS and present the recently proposed C-AMAT memory model.

2.1 Disaggregated Memory

As mentioned previously, DMS introduces a new memory layer to the memory hierarchy by adding external memory appliances connected over network. In [1] and [9], these appliances are known as memory blades. For the RAN implementation associated with Cooley [8], the memory appliances are Kove XPDs [7]. Each memory appliance is connected to multiple computing servers, and dynamically allocates data across all computing servers. A computing server can utilize all remote memory appliances dynamically to meet its needs, without being swapped to persistent storage, which can be orders of magnitude slower in data access. In DMS, a high bandwidth Host Channel Adapter (HCA) links the computing servers and memory appliances. Remote memory access can be implemented by swapping local pages to the memory appliance through explicit RDMA transfers. If the local memory's capacity is exhausted, a local page is evicted to remote memory. Users can utilize local memory locality to improve the overall performance of a DMS. Most prior evaluations of DMS [1, 3, 9, 16–18] are based on simulations and emulations, which cannot precisely present the performance under software and hardware implementation and interference. It is valuable exploring the performance evaluations and optimizations of an actual disaggregated memory machines.

2.2 The C-AMAT Model

C-AMAT (Concurrent-AMAT) [14] is an extension of the conventional AMAT [12] model to consider both data locality and concurrent data access. C-AMAT is defined as the ratio of the total (active) memory access cycles over the total number of memory accesses. Let $T_{MemCycle}$ represent the total number of cycles executed in which there is at least one outstanding memory reference; let C_{MemAcc} represent the overall number of data accesses to memory:

$$C - AMAT = \frac{T_{MemCycle}}{C_{MemAcc}} \quad (1)$$

C-AMAT is simple. The beauty of C-AMAT is: 1) Its cycle is the memory active cycle for a given memory layer. So, it can be used at each layer of a memory hierarchy for that layer's performance measurement. By default, unless stated differently, C-AMAT is the L1 cache's C-AMAT, which is the performance of the memory system, since L1 is the top level catch of the memory hierarchy (please notice that for a multicore system, we need to pay special attention on how to measure the L1 C-AMAT, since each core has its individual L1 cache.). Active cycle means only the clock cycles

where data access exist are counted. Active cycle's performance is the true performance of a memory system. 2). An overlapping mode is adopted to count memory access cycles. Multiple data accesses count multiple times. That is, $T_{MemCycle}$ increases by one when there are two memory accesses in the same cycle. Like AMAT [12], C-AMAT has a parameterized formulation as shown in Eq. (2) [9].

$$C - AMAT = \frac{H}{C_H} + pMR \times \frac{pAMP}{C_M} \tag{2}$$

H is the hit time, the same as defined in AMAT. *The Hit Concurrency* (C_H) in Eq. (2) could be contributed by multi-port cache, multi-banked cache or pipelined cache structures. The *Pure Miss Concurrency* (C_M) could be contributed by non-blocking cache structure. The *Pure Miss Ratio* (*pMR*) is the number of pure misses over the total number of accesses. The concept of pure miss is new. Pure miss is a miss containing at least one miss cycle which does not overlap with any hit [14, 15, 19]. *Pure Average Miss Penalty* (*pAMP*) is the average number of pure miss cycles per pure miss access. Like AMAT, C-AMAT is recursive and can be recursively applied to next level of the memory hierarchy [20, 21].

The C-AMAT model given by Eq. (1) and Eq. (2) needs to be refined to support multicore and DMS.

3 PERFORMANCE MODELING AND MEASUREMENT METHODOLOGY

3.1 C-AMAT Model for Multicore

The C-AMAT model can be applied to multicore processors in two ways, single-core measurement, and multicore (single processor) measurement. For the former, we are interested in a single core's, say core A's, performance. For the latter, we are interested in the overall multicore performance as a single multicore processor. By the definition of C-AMAT, core A's C-AMAT can be measured by core A's number of memory accesses and core A's memory active cycles. Likewise, the multicore C-AMAT can be measured by the multicore processor's overall number of accesses and the multicore's memory active cycles. Here the multicore memory active cycle is defined as, *a multicore memory is active if and only if any of its cores actively accessing memory.* To fully understand the measurement of a multicore C-AMAT, the multicore C-AMAT equation is given in Eq. (3) to illustrate the case where private L1 exists.

$$C{-}AMAT_{ncore} = \frac{Any\ (\{TMemCyle_1,\ TMemCyle_2, \ldots,\ TMemCylen\})}{\sum_{j=1}^{n} CMemAcc_j} \tag{3}$$

where $C\text{-}AMAT_{n\text{-}core}$ is the multicore C-AMAT of n cores and n private caches. $CMemAcc_j$ is the number of memory accesses of each of the j individual L1 caches. $TMemCyle_j$ is the active cycle of the L1 cache of core j. For example, if there are 6 cores in a multicore processor, then n is 6. For measuring the multicore L1 C-AMAT, all L1 from the multicore processor are considered as one integrated cache. Therefore, the number of memory accesses to this integrated cache is the summation of the number of memory accesses from all L1 caches in this multicore processor. Its active cycles are the cycles where any of the 6 L1cache is active.

3.2 Recursive C-AMAT Model for Deep Memory Hierarchies

The C-AMAT model is recursive and can accurately analyze the performance of a hierarchical memory system. The recursive C-AMAT is shown in the following Eq. (4) [21].

$$C - AMAT_i = \frac{H_i}{C_{H_i}} + MR_i \times \kappa_i \times\ C - AMAT_{i+1} \tag{4}$$

where

$$\kappa_i = \frac{pMC_i}{MC_i} \tag{5}$$

$C\text{-}AMAT_i$ is the C-AMAT of its level i cache; For example, when i is 1, $C\text{-}AMAT_1$ is the C-AMAT of L1; $C\text{-}AMAT_2$ is the C-AMAT of L2 cache. Eq. (4) contains three characteristics of caches such as locality, concurrency and overlapping. The parameter MR_i (*Miss Ratio of level i cache*) represents locality impact; the parameter C_{H_i} (*Hit Concurrency of level i cache*) represents concurrency impact; the parameter κ_i represents the impact of the overlapping between *Pure Miss Cycles* (*pMC*) and *Miss Cycles* (*MC*), as given by Eq. (5), which also means the overlapping between hit and miss accesses in this level i cache. A smaller κ_i value means a better overlapping between hit and miss access. By Eq. (5), κ_i is a positive value less than or equal to 1 [20, 21]. Based on the recursive C-AMAT model, C-AMAT is a memory performance metric integrating the memory characteristics of the entire memory hierarchy.

3.3 Recursive C-AMAT Model for Deep Memory Hierarchies

The execution time of a program consists of two parts: the processor computing time and memory stall time [12]. The processor computing time is the time executing the user program. The memory stall time is the time where the processor is stalled waiting for accessing data. This stall time consists of the access delay of memory media, contention delay of queueing, and, in multi-thread cases, the latency due to cache coherency and consistency, etc. Eq. (6) is the classic formulation of processor execution time (CPU-time) in terms of memory stall time [12].

$$CPU - time = IC \times \left(CPI_{exe} + \frac{MSC}{IC}\right) \times Cycletime \tag{6}$$

Here, IC is the number of instructions, cycle-time is the time duration of a clock cycle, and CPI_{exe} is the ideal processor computation cycle per instruction without any data access delay. *MSC* is the overall memory stall cycles. Eq. (7) presents the relation between the memory stall cycles and C-AMAT [20, 21].

$$MSC = IC_{mem} \times C - AMAT \times (1 - Overlap_{cm}) \tag{7}$$

Here, IC_{mem} is the number of the memory access instructions (Load and Store instructions); $Overlap_{c\text{-}m}$ is the ratio of the overlapping time between computing time and memory access time over the total memory access time (If the CPU is in-order, the $Overlap_{c\text{-}m}$ should be 0. But if the CPU is out-of-order, the $Overlap_{c\text{-}m}$ should be more than 0.). From Eq. (6) and Eq. (7), we get Eq. (8) which gives the relation between C-AMAT and the execution time.

$$CPU{-}time = IC \times Cycle{-}time \times (CPI_{exe} + f_{mem} \times C{-}AMAT \times (1 - Overlap_{cm})) \tag{8}$$

Here, f_{mem} is the ratio of the memory access instructions IC_{mem} (Load and Store instructions) over the number of instructions IC. From Eq. (8), we can see that C-AMAT is the main factor of memory stall. By Eq. (8), the task of reducing memory stall time becomes the task of reducing the value of C-AMAT.

3.4 Impact of Pure Miss Cycles on C-AMAT

Pure Miss Cycles (*pMC*) is a measurable performance parameter and the L1's pure miss cycles are the memory stall cycles of the underlying CPU [20, 21]. We need the following two relations to measure and understand the C-AMAT value on the Argonne DMS machine, Cooley.

Theorem 1. The relation between C-AMAT and *pMC* is given by Eq. (9).

$$C-AMAT = \frac{H}{C_H} + \frac{pMC}{C_{MemAcc}} \tag{9}$$

Proof. Based on the Pure Miss Concurrency's definition [9], we have Eq. (10) for the relation between *pMC* and Accumulated Pure Miss Cycles (pMC_{accum}).

$$pMC = \frac{pMC_{accum}}{C_M} \tag{10}$$

where C_M is the *Pure Miss Concurrency*. The $pMC_{accum}$is shown as Eq. (11).

$$pMC_{accum} = C_{pMA} \times pAMP \tag{11}$$

where C_{pMA} is the number of pure miss accesses and *pAMP* is the *Average Pure Miss Penalty*. Substituting Eq. (11) into Eq. (10), we have Eq. (12):

$$pMC = \frac{C_{pMA} \times pAMP}{C_M} \tag{12}$$

By definition, *pMR* can be shown as Eq. (13).

$$pMR = \frac{C_{pMA}}{C_{MemAcc}} \tag{13}$$

Combining Eq. (12) and Eq. (13), we get the Eq. (14).

$$\frac{pMC}{C_{MemAcc}} = pMR \times \frac{pAMP}{C_M} \tag{14}$$

Finally, submitting Eq. (14) into Eq. (2), we get C-AMAT versus *pMC* equation Eq. (9).

Theorem 1 gives the relation between *pMC* and C-AMAT. Theorem 2 presents the relation between *pMC* and the next level C-AMAT.

Theorem 2. The relation between *pMC, Miss Ratio,* κ and the next level cache's C-AMAT is indicated by Eq. (15).

$$pMC_1 = C_{Mem\mathrm{Acc}_1} \times MR_1 \times \kappa_1 \times^{\circ} C-AMAT_2 \tag{15}$$

Proof. The recursive C-AMAT formulation is given by Eq. (4). From Eq. (4) and Eq. (9), following some mathematic deduction, we can get Eq. (15), with the fact that L1's *pMC* is equal to the multiplication of the number of data accesses in L1 ($C_{Mem\mathrm{Acc}_1}$), the *Miss Ratio* of L1 (MR_1), the overlapping metric (κ_1) and the C-AMAT of L2 cache.

By Eq. (6), the impacts of a memory system on computing is its memory stall time. Memory stall time is the focus of our DMS study. In *summary*, results given in this section show, to reduce memory stall time in a concurrent data access environment we need to reduce the C-AMAT (Eq. (8)) value, and reducing the C-AMAT value is equivalent to reduce the pure miss cycles (Eq. (15)). That paves the way for our measurement and optimization on the Argonne Cooley machine.

3.5 Measurement Methodology in Real Hardware

To measure multicore C-AMAT, we need to measure the number of active cycles and the number of data accesses in each core's cache (see Eq. (3)). Intel provides several performance counters which can be used to calculate L1 and L2 cache's C-AMAT, and other parameters in the recursive C-AMAT model such as κ_1. These performance counters [22] are provided by Intel Performance Monitoring Units (PMUs). The PMU is a component built inside a processor to monitor its performance events. These performance events provide facilities to characterize the interaction between programmed sequences of instructions and micro-architectural sub-systems. Table 1 lists the metrics we used in this work and their corresponding events from PMUs.

To collect these events, the perf profiler [23] is used to monitor these events for each core. Perf profiler tracks the performance of each core including the cache systems. By using it, the events needed for calculating multicore C-AMAT can be measured and the multicore C-AMAT of a specific caches level can be obtained.

4 PERFORMANCE EVALUATION

In this section, we evaluate a special case of DMS, the RAN system available at the Argonne National Laboratory. We firstly conduct a performance comparison between RAN and HDD disk to evaluate the benefit of RAN in terms of adding an additional memory layer. Then, we conducted a detailed performance evaluation by running different applications with different access patterns on RAN and local memory. Finally, we analyzed some interesting performance phenomenon identified by the models introduced in Section 3.

4.1 Setup

All the evaluations are conducted on Cooley [8], a computer cluster at the Argonne National Laboratory which is equipped with the RAN architecture. RAN is a practical implementation of DMS and is deployed on Cooley. As shown in Fig. 2, each node in Cooley has two Intel Haswell E5-2620 v3 processors. The hyperthreading of these processors is disabled in these processors in our performance

Table 1: List of Metrics And Their Relevant Events

Metric Name	Event Name
L1 Active Cycles	CYCLE_ACTIVITY.CYCLES_LDM_PENDING
L1 Data Accesses	MEM_UOPS_RETIRED.ALL_LOADS + MEM_UOPS_RETIRED.ALL_STORES
L1 Pure Miss Cycles	CYCLE_ACTIVITY.STALLS_L1D_PENDING
L1 Miss Cycles	CYCLE_ACTIVITY.CYCLES_L1D_PENDING
L2 Active Cycles	CYCLE_ACTIVITY.CYCLES_L1D_PENDING
L2 Data Accesses	L2_RQSTS.REFERENCES

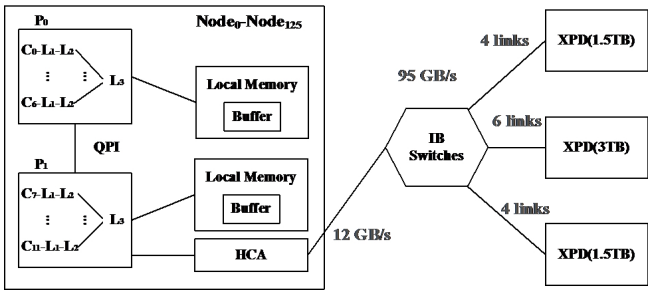

Figure 2: The architecture of RAN in Cooley

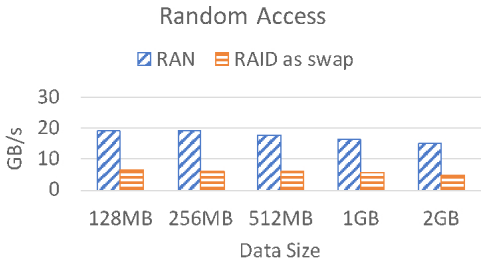

Figure 3: Performance of Random Access using RAN and HDD

evaluation. Each processor has six cores with six hardware thread. In each processor, there are three levels of cache. L1 and L2 caches are private to each core. The L3 cache is shared by all the cores in the same processor. All the processors in one node share the local 384 GB DDR4 memory. RAN includes three Kove XPD memory targets connected by InfiniBand switches with different number of links [7]. Their overall bandwidth is about 95GB/s [7]. RDMA over InfiniBand is used to access the XPD box, which can be simply regarded as a memory pool with InfiniBand interconnects. The total capacity of this Cooley RAN is a 6TB DRAM (one XPD with 3TB and two with 1.5TB capacity). When one node in Cooley needs to access remote memory, a buffer in the local memory of the node will be allocated to exchange the data with the XPDs. RAN allows applications to transparently access the remote memory pool without code modifications. A C API developed by Kove can also be used to explicitly move data to and from remote memory [7]. All our experiments are conducted with the transparent mode. The performance results presented are averaged over 10 runs.

4.2 RAN vs HDD

To compare the performance of RAN with that of HDD disk, we conduct a performance evaluation to access RAN and HDD disk with random access pattern. In this evaluation, we use one node from Cooley with 12 threads to issue random accesses. Bandwidths from local memory to RAN and from local memory to the HDD disk (HP Smart Array P420i Controller, RAID5, qty-7 300GB 15K RPM SAS drives) are measured with different data sizes. Here we access the HDD disk as the virtual memory and limit the local memory used in this evaluation to 75% of the data set by using cgroups. The buffer size in local memory when accessing RAN is also limited to 75% of the data set. The local memory can be considered as the L4 cache of RAN or the HDD disk. In both cases, when the assigned local memory is exhausted, local pages will be evicted and swapped with the next tier.

The access latency of RAN (around 3us) is much smaller than that of HDD (about 12ms). One of the motivations of DMS is it will overperforms HDD. Our experiments confirm this assumption. As shown in Fig. 3, RAN is much faster than the HDD array for all data sizes, even though RAN is in the remote side connected with InfiniBand network, while the HDD disks are local with the nodes. From this result, we can see that if an application data size is more than the local memory size, it is a better choice to run the application on RAN than on HDD disks even they are with the nodes locally.

Table 2: List of Applications with Different Access Patterns

Application	Access Pattern	Footprint
MiniFE	Sequential	32GB
DGEMM	Sequential	23GB
Graph500	Random	17GB
GUPS	Random	16GB

4.3 Impact of Memory Access Patterns on Application Performance

For understanding the performance of different data access patterns on RAN, the performance evaluations of four applications with different memory access patterns are conducted on a Cooley node. As shown in Table 2, the four applications are MiniFE and DGEMM with sequential memory access pattern, Graph500 and GUPS with random memory access pattern, respectively. Fig. 4 shows the execution time and performance slowdown of these four applications [24–27] with different memory configurations and thread scaling. Lmem means that only local memory is used; Xmem(X%) means that RAN is used and the buffer size in the local memory is X% of the total memory used by the application. Xmem(X%) SLD is the performance slowdown for Xmem(X%) over that of Lmem with the same number of threads. RAN provides the ability to limit the size of its local memory buffer, allowing us to measure the influence of local memory buffer size on DMS performance.

Fig. 4(a) and Fig. 4(b) present the performance of two applications with sequential memory access patterns (MiniFE and DGEMM). Binding to local memory (Lmem) provides the best performance for both. These two applications slowdown 3.3 to 4.9 and 2.6 to 9.1 times with Xmem(25%) on RAN, respectively. Fig. 4(c) and Fig. 4(d) present the performance of two applications with random memory access patterns (Graph500 and GUPS). These two applications also achieve the highest performance by only allocating data to the local memory (blue bars). But these two applications slowdown 10.7 to 17.5 and 29.5 to 86.8 times with Xmem(25%) on RAN, respectively. With larger local buffer size Xmem(75%), their execution time and slowdown are much lower than those with Xmem(25%) because larger local buffer size can improve locality (reducing its miss ratio) in local memory, especially for locality-friendly applications such as MiniFE.

By comparing Fig. 4(a-b) and Fig. 4(c-d), it is evident that the application memory access pattern is an important factor to achieve

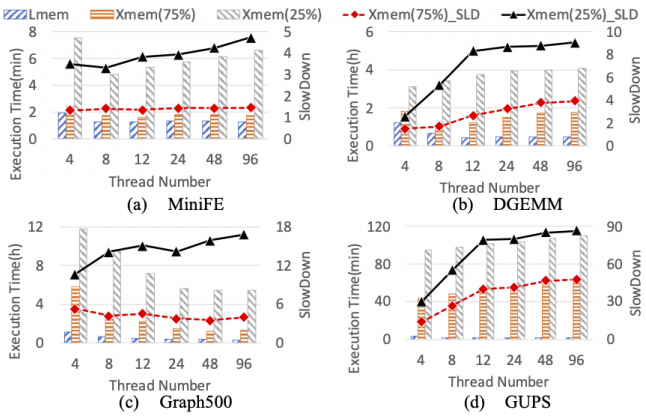

Figure 4: Time & slowdown of different applications

higher RAN performance. If an application has sequential access pattern, both prefetcher and out-of-order core can perform well to increase the number of memory requests. In that case, the bottleneck becomes the maximum number of concurrent requests that can be supported by the underlying hardware bandwidth. Therefore, sequential applications are sensitive to memory bandwidth. However, applications exhibiting random access patterns are more sensitive to memory latency. In the case of random-access patterns, the memory addresses cannot be predetermined, and the number of requests remains small. Thus, they are penalized by the higher latency of RAN. Since RAN's bandwidth is limited by that of InfiniBand HCA (about 12GB/s) (see Fig.2), which is lower than that of local memory (about 68GB/s) but has a smaller gap than that of their latencies (about 11ns vs 3us), applications with sequential access have lower performance loss when using RAN.

As shown in Fig. 4, the Xmem(75%) RAN performance is better than the Xmem(25%) performance on these four applications, which is as expected. Eq. (4) and Eq. (8), explain this since by reducing the miss ratio of local memory one can increase the performance of local memory and, therefore, improve the overall performance. Larger buffer size will reduce conflict misses and reduce remote DMS data access.

4.4 Case Study on K-means

K-means is a popular benchmark for machine-learning applications. The OpenMP-based K-means from Rodinia benchmark set [28] with 23GB memory footprint is tested on one node with different memory configurations and a variety of thread counts. These performance tests produced some interesting and unexpected results.

4.4.1 Counterintuitive Results. Usually, local memory performs better than RAN because of the longer latency and lower bandwidth of RAN. However, Fig. 5 shows that the execution time of Xmem(75%) is better than that of Lmem when the thread count is between 4 and 96. Similarly, the execution time of Xmem(25%) is better than that of Lmem when the thread count is between 4 and 48. The superior performance of K-means while using RAN is counterintuitive and unexpected.

Since K-means is a data-intensive benchmark, based on Eq. (8), its execution time performance is highly related to L1's performance

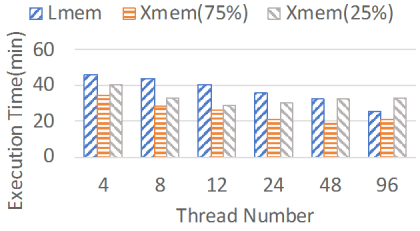

Figure 5: Execution time of K-means with different configurations

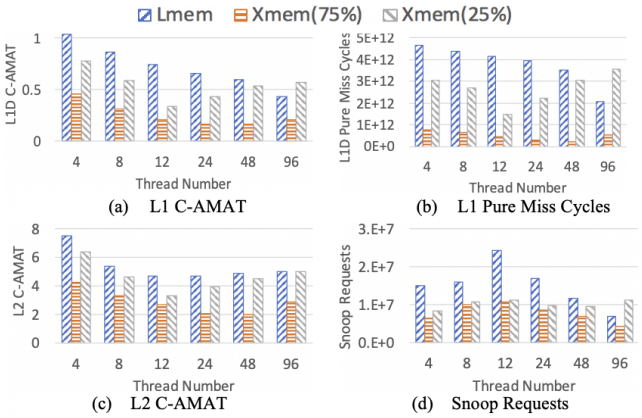

Figure 6: K-means performance with different metrics

(its C-AMAT). Based on Eq. (4) and the structure of the memory hierarchy, the L1 C-AMAT reflects the overall performance of the memory hierarchy under L1, including the impact of local memory and/or RAN. That is the reason, in a single node or single processor measurement, we use the term C-AMAT and the term L1's C-AMAT interchangeably for the performance measurement. If RAN's performance is changed, the (L1's) C-AMAT value also will be changed accordingly. Here the network impact on RAN is also implicitly included in C-AMAT. Therefore, we can analyze the variation of the memory hierarchy and RAN's performance through the study of C-AMAT.

Using Eq. (3) and its measurement methodology supported by Intel PMUs, the obtained C-AMAT value is shown in Fig. 6(a). The lower C-AMAT means the better L1 performance. In Fig. 6(a), the L1's C-AMAT of Xmem(75%) is less than that of Lmem with thread number from 4 to 96. Similar trend can be observed for Xmem(25%) case with 4 to 48 threads. From the analysis of Eq. (8), the main reason of these extraordinary results is that the RAN memory system has a better L1 performance than that of local memory. The L1 C-AMAT's trend in Fig. 6(a) is very similar to that in Fig. 5 except with a smaller performance gap. To calculate the execution time through Eq. (8), we need to multiply the f_{mem} and $\text{Overlap}_{\text{c-m}}$ factors to C-AMAT and then add in CPI_{exe}. Both f_{mem} and $\text{Overlap}_{\text{c-m}}$ are less than 1. These multiplication and addition make C-AMAT having larger performance gaps among these three memory configurations, compared Fig. 5 with those in Fig. 6(a).

To explain Fig. 6(a), we need to utilize Eq. (9) to see how other metrics affect C-AMAT. *H (Hit Time)* and C_{MemAcc} (*the overall*

number of data accesses to cache) for L1 are fixed in our testing configurations with the same application and input. However, C_H (*the Hit Concurrency of the cache*) may increase with the thread number of applications until it reaches the hardware concurrency limitation. *pMC* also changes with different factors, such as the overlapping *k*, memory hierarchy configuration, locality and miss concurrency as shown by Eq. (10) and Eq. (15).

Since the H and C_{MemAcc} are the same in this case study of K-means due to being tested on the same hardware, the L1 value is only related to *pMC* and C_H. From Fig. 6(b), the trend of L1's *pMC* is like that in Fig. 6(a). Using more threads can increase C_H until reaching the hardware concurrency limitation, which is a reason why the L1 C-AMAT keeps decreasing with more threads. This *pMC* trend and the increase of C_H of L1 are the main reasons for the trend of L1's C-AMAT in Fig. 6(a), based on Eq. (9). The performance gaps among these three memory configurations in Fig. 6(a) is much smaller than those in Fig. 6(b). That means pure miss reduced more than miss on RAN. By Eq. (5), only pure misses reduce C-AMAT, not misses. Fig. 6(b) confirms our assumption that the RAN K-means overperformance is *due to latency hiding overlapping*.

From Fig. 6(c), the trend of L2 cache's C-AMAT is like the trend of Fig. 6(a) except having smaller performance gaps among the three memory configurations. Here the miss ratio of L1 and κ_1 made the difference gaps in Fig. 6(c) smaller based on the recursive function Eq. (4). As shown in Fig. 6(d), the number of snoop requests from L3 cache using RAN is also reduced compared with that using local memory in each case with counterintuitive performance. These show *overlapping is a reducer of miss penalty.*

4.4.2 Assumption. From the C-AMAT performance analysis, we can see that the reason of RAN outperforming local memory in these K-means cases is the performance improvement of L1 and L2 cache and the reduction of the snoop requests from L3 cache, as shown by Fig. 6. The reduction of snoop requests from L3 cache gives a clue that using RAN makes less thread communications. Threads communicate to each other to share data and data areas. There are two different types of data sharing: true sharing and false sharing [29]. True sharing communications are impossible to be removed since they are due to the inherent data sharing of the codes and the algorithms. False sharing can be removed with appropriate code/data arrangement. In addition, the L1 cache and L2 cache's performances are improved when using RAN. So, we conclude, RAN has led to a more appropriate data arrangement which mitigates the false sharing impact via concurrent data access overlapping. In other words, the K-means code as provided exhibits false sharing and using RAN can reduce the false sharing impact, resulting in improved performance.

4.4.3 Code-level Analysis of K-means with the C-AMAT Model. Having identified the reason for performance anomaly, we examined the K-means algorithms and identified the source of the false sharing. Fig. 7 shows the main codes of K-means, which consists of five parts: *find_nearst_point* (line 11), *update_new_centers* (from line 13 to 14), *update_centers_len* (line 15), *update_membership* (line 16 and 17) and others (all lines except its main codes from Fig. 7). The code-level analysis of these codes is conducted by Vtune [30] from Intel to get each code area's performance data, including the

```
1  #pragma omp parallel \
2  shared(feature,clusters,membership,partial_new_centers,partial_new_centers_len) {
3      int tid = omp_get_thread_num();
4      #pragma omp for \
5              private(i,j,index) \
6              firstprivate(npoints,nclusters,nfeatures) \
7              schedule(static) \
8              reduction(+:delta)
9        for (i=0; i<npoints; i++) {
10         /*find_nearest_point only contains line 11*/
11         index = find_nearest_point(feature[i],nfeatures,clusters,nclusters);
12         /*update_new_centers contains from line 13 to line 17*/
13         if (membership[i] != index) delta += 1.0;//update_membership(line 13 to 14 )
14         membership[i] = index;
15         partial_new_centers_len[tid][index]++;//update_centers_len(line 15)
16         for (j=0; j<nfeatures; j++)//update_centers(line 16 to 17)
17            partial_new_centers[tid][index][j] += feature[i][j];
18       }
19 }
```

Figure 7: The main codes of K-means

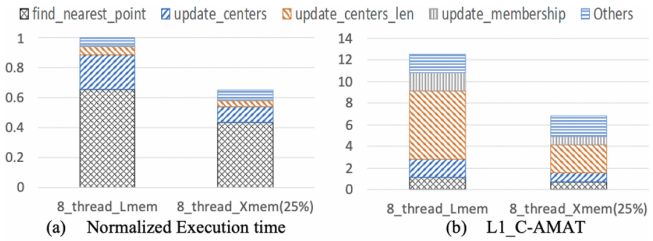

Figure 8: Normalized execution time and L1_C-AMAT of different code areas of K-means

C-AMAT values. By using Vtune, we get the performance results for each of these five code areas in Fig. 8 which shows the normalized execution time and L1 cache's C-AMAT values of these code areas on local memory and RAN.

In Fig. 8, we took one K-means' performance case from Fig. 5, where only Lmem and Xmem (25%) are measured and only for the 8 threads performance, to conduct a code-level analysis. From Fig. 8(a), the major improvement of Xmem (25%) are from *find_nearest_point* and *update_centers* code sections. By analyzing the code-level L1 C-AMAT values in Fig. 8(b), we get a different observation with that of Fig. 8(a). We find *update_center_len* has the largest performance improvement in C-AMAT. As shown in Fig. 8(b), *update centers_len* performs poorly on local memory but its performance is dramatically improved on RAN (Recall, the lower the C-AMAT value is, the better the memory performance is). So, *update_centers _len* is likely the code area causes a major false sharing. On the other hand, the *find_nearest_point* and *update_centers* code sections have much more instructions than *update_centers_len*. Also, the feature array in *find_nearest_point* and *update_centers* have much more contents than the *partial_new centers_len* array has. That means they likely have more data instructions issued than that of *partial_new_centers_len*. By Eq. (8), *find_nearest_point* and *update_centers* can get more performance improvement in terms of execution time because they have more data instructions and bigger data sizes than that of *update_centers_len* (Fig. 8(a)), even they have a lower L1_C-AMAT improvement than that of *update_centers_len* (Fig. 8(b)). Eq. (8) shows that data access improvement is more important for certain code sections than others in terms of the overall execution time.

Now, let us zoom in on the *update_centers_len* code section. Observing the parameters in the integer *partial_new_centers_len* array, we find that *tid* and *index* with the values (*tid*=8 and *index*=5)

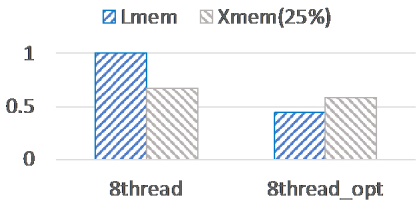

Figure 9: Normalized execution time of K-means with and without false sharing

in K-means make this array only stay in three cache lines and 8 threads share them with write operations. These write operations causes false sharing and snoop requests are sent to private L1 and L2 cache. These snoop requests can dramatically affect cache performance, and, in turn, decrease the overall performance. To prove our analysis, we optimized the K-means codes to avoid false sharing by putting each item of *partial_new_centers_len* array on different cache lines. Fig. 9 show the performance difference, where 8thread is the original K-means implementation with false sharing and 8thread_opt is the optimized K-means implementation without false sharing. From Fig. 9 local memory performance improves highly after removing the false sharing. RAN does not have any performance advantage after the false sharing is removed.

4.4.4 Why using RAN can reduce the penalty of false sharing in K-means?. Fig. 10 is from the *hpctoolkit* summary view [31]. The top one is the map of execution time for different functions of the original K-means implementation with local memory having 8 thread, the bottom one is that with RAN having 8 threads. The false sharing is happened in the purple area of the top figure and in the black area of the bottom figure. From Fig. 10, at first the *memcpy* in RAN is slower than in local memory since RAN has longer latency than local memory. But when running the next code areas, using RAN is faster than in local memory, where we have more black bar areas than purple ones. It means that using RAN has less impact of false sharing than using local memory. The deeper memory hierarchy and longer latency in the last tier results in a larger number of threads being stalled, reducing contention and thus the amount of false sharing.

4.5 Discussions

The idea of disaggregated memory systems (DMS) is to have a shared global memory to reduce the aggregated amount of local memory in an HPC system, for reducing purchase, power, and operating costs and for a better memory utilization. The DMS approach is well motivated. From Fig.1, we can see that in general the average node memory utilization is very low (around 25GB on average) via the average memory usage distribution. There are some memory usage bursts here and there through the time, which justify the need of a global disaggregated memory system. While Fig. 1 confirms the value of disaggregated memory, it is only based on a one-year collection of one HPC system. We need a more thorough study to fully utilize the merits of DMS. In addition, the key idea of DMS is to add a new layer into the existing memory hierarchy. With the emergence of big data applications and the availability of NVRAM technologies, multi-layered deep memory hierarchy becomes a trend of system design in hardware [32] and software [33]. Intel will make its multi-layered deep memory Optane technologies available in 2020 [32]. A thorough study of DMS will not only help to have a better understanding of DMS, but also will help us to have a better understanding of the design of a general multi-layered deep memory hierarchy system.

From the analysis of our performance evaluation results in Section 4.2, we can see that the RAN's performance is much better than that of HDD disks. Therefore, if the application data size is more than the local memory size, the DMS approach is a great help. In addition, RAN performance slowdown of sequential data access is much lower than that of random access. Thus, applications with sequential data access patterns are more suitable for RAN. Based on the analysis and performance measurement given in Section 4.3, the current performance loss of RAN is mainly from the limited bandwidth and high latency of the network connected to RAN. However, the network bandwidth is growing faster than that of local memory, and network latencies will continue to fall, making memory-speed network transfers feasible in the future [7]. For example, using the recent advanced technology, Compute Express Link (CXL) [34], can improve the access latency of RAN, because it can allow CPU directly access RAN without accessing the local memory buffer first. An integrated heterogeneous hardware-based memory hierarchy, such as Intel Optane [32] or HP "The Machine" [35], will provide an even better DSM solution.

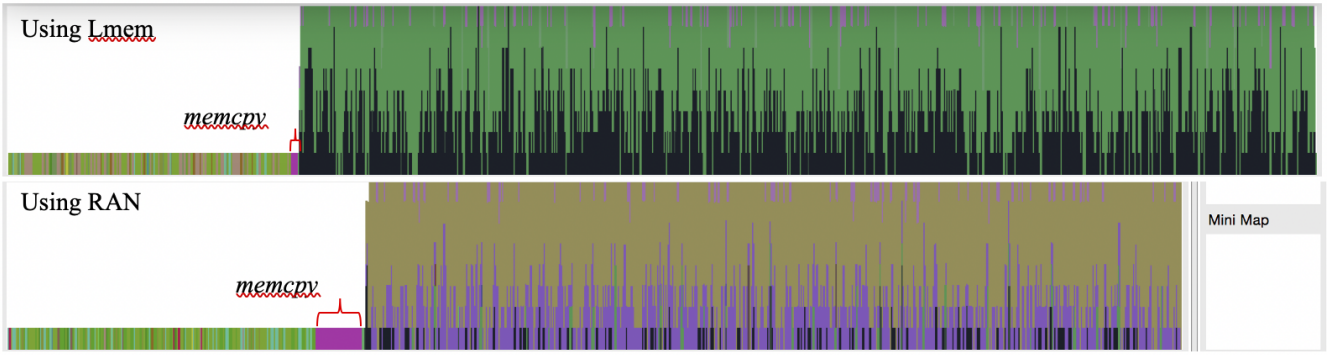

Figure 10: The summary views from hpctoolkit

Surprisingly, in some situations RAN can have a better performance than local memory. We used the newly proposed C-AMAT memory system performance model to analyze the unexpected but plausible performance. Based on our performance analysis (Section 4.4), the better performance of RAN is due to overlapping, overlapping computing and data access, and overlapping data access hit and data access miss. This overlapping can be measured by C-AMAT, the performance model which considers both data locality and concurrency. DMS is designed for supporting memory sharing. It increases the importance, as well as the opportunity, of the overlapping with adding an additional layer on the memory hierarchy. Of course, the need of overlapping data access delay is due to there exist a major data access delay. C-AMAT has identified the major data access delay. Through code-level analysis, we have removed the false-sharing data access delay of the K-means multicore programming and achieved performance optimization via performance evaluation. Please notice that the code-level analysis is conducted to verify the C-AMAT performance analysis. It confirms the C-AMAT findings. In a general engineering environment, code-level analysis is costly and not feasible, unless C-AMAT can narrow down the troubling code segment.

In *summary*, through a thorough study of the Cooley DMS implementation, we have the following observations of DMS: a) network and latency is the performance bottleneck of current DMS implementation; b) a good sequential locality may lead to bad parallel locality in a multicore system; c) this good/bad change is not only determined by data concurrency but also by data access overlapping; d) C-AMAT can measure the integrated performance of data locality, concurrency, and overlapping; e) DMS can mitigate data sharing delay and can get a better data access overlapping.

Observation b) through d) are general to any parallel data access. They suggest we need to rewrite the parallel code for better data sharing to fully appreciate concurrent data access. This rewrite requires the skill of parallel programming, the understanding of the application, and the knowledge of the underlying computer architecture. That is a high bar for an engineering practitioner. Observation e) suggests DMS can improve parallel data access via overlapping. DMS adds another opportunity to improve data access delay via system optimization and support. The RAN overperformance on K-means is due to thread level data sharing. Thread level data sharing is common in many big data applications. We have found RAN performs well on other AI applications as well. We are currently testing and optimizing more AI applications on Cooley and investing the potential of DMS on AI applications. Since network is already the performance bottleneck of Cooley on one node testing, we did not conduct multi-node testing on the current DMS implementation. We plan to continue to investigate the DMS network issue in our future study when new network and memory technologies become available.

5 RELATED WORK

Lim et al. first proposed the concept of hardware disaggregated memory with two models: using it as a network swap device (swap-based model) and transparently accessing it through memory instructions by the hypervisor (hypervisor-based model) [1]. The concept was identified to be promising [1, 6, 36] and many researches are followed. The swap-based model has been studied in [37–40]. However, studies find that the swap approach introduced by [1, 6, 36] has additional interruption and computation overhead toward processors due to controlling network operations on each memory node. To reduce these overheads, INFINISWAP [17] utilized a decentralized manner to implement a remote memory paging-based caching system designed for RDMA networks such as InfiniBand. In a swap subsystem, its memory reclamation only focused on the initially used memory, which is rarely reused in some applications [41]. To aim this issue, K. Koh et al. [42] made the page access records available to the hypervisor and provided the support on application-aware elastic block sizes where remote memory pages with different granularities can be fetched. In this work, RAN system supports both swap-based model and hypervisor-based model.

Some prior works [1, 6, 9, 16, 36] have evaluated disaggregated memory using simulation with statically defined estimates of software overheads. Software implementation issues, such as synchronization in key hypervisor functions, system-level interaction with I/O devices, and the role of software memory optimizations were not considered in [1, 6, 9, 16, 36]. Even though several cluster nodes were used to emulate the disaggregated memory for its performance evaluation [6, 17, 18, 42, 43], they cannot truly represent DMS' performance because emulations have the severe performance costs and are difficult to consider all the software and hardware features and their interference.

There also have been a few hardware disaggregation proposals, including HP "The Machine" [35], dRedBox [44] and RAN [7]. The Machine's scale is a rack and it connects SoCs with NVMs with a specialized coherent network. The lifespan of NVMs is not enough when it is utilized as the disaggregated memory and the maintenance of the Machine is hard and expensive. dRedBox and RAN package memory resources in several cases and connect them with PCIe buses. There are not enough evaluation data for dRedBox in [44]. In [7], there are some evaluation data about RAN while there is no model-based analysis and no code-level case study as given in this research.

6 CONCLUSIONS

In this study, an appropriate disaggregated memory performance model is proposed based on the C-AMAT memory model and a well-designed performance evaluation is conducted on a production disaggregated memory system (DMS), the Argonne Cooley machine with a RAM area network (RAN) DMS implementation. Based on a thorough analysis and evaluation investigation, we find DMS is appropriate for many applications. RAN performs better than the conventional local memory+disk high performance computing architecture in handling data access bursts. Its performance in general is below local memory due to network delay and contention. But, to our surprise, some applications, such as K-means, have achieved a better RAN performance than pure local memory performance under some circumstances. Based on the C-AMAT performance analysis, this RAN overperformance is due to cache thread-level data sharing and due to concurrent data access overlapping. The former makes local memory very slow. The latter makes RAN can overlap its data access delay and mitigate the data sharing overhead. This C-AMAT performance analysis is confirmed by our code-level

analysis and code-level optimization. We have revealed the difficulty of achieving good data locality for big data applications, such as K-means, and the potential of concurrent data access in a multicore data sharing environment. RAN, and therefore DMS, adds another opportunity, also complexity, for data scheduling and optimization. These findings can help users make decision on whether to use RAN for their specific applications and help future DMS design and implementation. This study shows the limitation and potential of DMS in a practical setting with a solid memory performance model. It lays a foundation for future DMS and multi-layered deep memory hierarchy design and optimization. Since thread-level data sharing is common in AI applications and general big data applications, this research lights up the need of further study of the potential of deep memory hierarchy on AI and general big data applications.

ACKNOWLEDGMENTS

This research has been funded in part and used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory supported by the Office of Science of the U.S. Department of Energy under contract DEAC02-06CH11357 and by the National Science Foundation under grant CCF-2008907, CNS-1730488 and CCF-1536079.

REFERENCES

[1] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch. Disaggregated memory for expansion and sharing in blade servers. In ISCA '09: Proceedings of the 36th annual International Symposium on Computer Architecture, pages 267–278, New York, NY, USA, 2009. ACM.

[2] A. Lebeck, X. Fan, H. Zheng and C. Ellis. Power Aware Page Allocation. In Proc. of the 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX), Nov. 2000.

[3] V. Pandey, W. Jiang, Y. Zhou and R. Bianchini. DMA Aware Memory Energy Conservation. In Proc. of the 12th Int. Sym. on High-Performance Computer Architecture (HPCA-12), 2006

[4] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge and S. Reinhardt. Understanding and Designing New Server Architectures for Emerging Warehouse-Computing Environments. In Proc. of the 35th Int. Sym. on Computer Architecture (ISCA-35), June 2008.

[5] P. Ranganathan and N. Jouppi. Enterprise IT Trends and Implications for Architecture Research. In Proc. of the 11th Int. Sym. on High-Performance Computer Architecture (HPCA-11), 2005

[6] K. T. Lim, Y. Turner, J. R. Santos, A. AuYoung, J. Chang, P. Ranganathan, and T. F. Wenisch. System-level implications of disaggregated memory. In IEEE Symposium on High Performance Computer Architecture (HPCA), pages 189–200, Feb. 2012.

[7] W. Allcock, B. Bernardoni, C. Bertoni, N. Getty, J. Insley, M. E. Papka, S. Rizzi and B. Toonen. RAM as a Network Managed Resource. In IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), May 2018.

[8] Argonne Leadership Computing Facility. Introduciton of Cooley. https://www.alcf.anl.gov/user-guides/cooley.

[9] P. X. Gao, A. Narayan, S. Karandikar, J. Carreira, S. Han, R. Agarwal, S. Ratnasamy, and S. Shenker, "Network requirements for resource disaggregation," in 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16), pp. 249–264, 2016.

[10] Y. Shan, Y. Huang, Y. Chen and Y. Zhang. LegoOS: A Disseminated, Distributed OS for Hardware Resource Disaggregation. In 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI'18), pp. 69-87, 2018.

[11] W. A. Wulf and S. A. McKee. Hitting the memory wall: Implications of the obvious. SIGARCH Comput. Archit. News, 23(1):20{24, Mar. 1995.

[12] J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach (sixth edition), The Morgan Kaufmann Publishers, 2019 (ISBN-13: 978-0128119051).

[13] A. Glew, "MLP yes! ILP no!" in Proc. ASPLOS Wild and Crazy Idea a workshop Session'98, Oct. 1998.

[14] X.-H. Sun and D. Wang. Concurrent Average Memory Access Time. Computer, vol. 47, no. 5, pp. 74–80, 2014.

[15] X.-H. Sun, "Concurrent-AMAT: A Mathematical Model for Big Data access," HPC Magazine, 2014.

[16] H. Meyer, J. C. Sancho, J. V. Quiroga, F. Zyulkyarov, D. Roca, and M. Nemirovsky. Disaggregated computing. an evaluation of current trends for datacentres. In international Conference on Computational Science (ICCS'17), 2017.

[17] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. Shin. Efficient Memory Disaggregation with Infiniswap. In Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17), 2017.

[18] D. Buragohain, A. Ghogare, T. Patel, M. Vutukuru, and P. Kulkarni. DiME: A Performance Emulator for Disaggregated Memory Architectures. In Proceedings of the 8th Asia-Pacific Workshop on Systems. ACM, 2017.

[19] N. Zhang, C. Jiang, X.-H. Sun, and S. L. Song. Evaluating GPGPU Memory Performance Through the C-AMAT Model. In Proceedings of the Workshop on Memory Centric Programming for HPC, pp. 35-39. ACM, 2017.

[20] Y.-H. Liu, and X.-H. Sun. LPM: concurrency-driven layered performance matching. In Parallel Processing (ICPP), 2015 44th International Conference on, pp. 879-888. IEEE, 2015.

[21] Y. Liu and X. Sun, "LPM: A Systematic Methodology for Concurrent Data Access Pattern Optimization from a Matching Perspective," in IEEE Transactions on Parallel and Distributed Systems, vol. 30, no. 11, pp. 2478-2493, 1 Nov. 2019

[22] Intel. Intel®64 and IA-32 Architectures Optimization Reference Manual.https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf

[23] Perf profiler, https://perf.wiki.kernel.org/index.php/Main_Page

[24] The DGEMM Benchmark," http://www.nersc.gov/research-and-development/apex/apex-benchmarks/dgemm/, 2018, [Online; accessed 11-May-2018].

[25] "The Graph500 Benchmark," http://www.graph500.org/, 2018, [Online; accessed 11-May -2018].

[26] "HPC Challenge Benchmark," http://icl.cs.utk.edu/hpcc/index.html, 2018, [Online; accessed 11-May-2018].

[27] H. Lv, G. Tan, M. Chen, and N. Sun. Understanding parallelism in graph traversal on multi-core clusters. Computer Science-Research and Development, vol. 28, no. 2-3, pp. 193–201, 2013.

[28] S. Che, M. Boyer, J. Meng *et al.*, Rodinia: A benchmark suite for heterogeneous computing. in IEEE International Symposium on Workload Characterization, pp. 44-54, 2009.

[29] M. A. Sasongko, M. Chabbi, P. Akhtar, and D. Unat. "ComDetective: A Lightweight Communication Detection Tool for Threads". In Proceedings of ACM Supercomputing (SC'19), Denver, November 17–22, 2019

[30] Intel Vtune Amplifier, https://software.intel.com/en-us/vtune

[31] L. Adhianto , S. Banerjee , M. Fagan , M. Krentel , G. Marin , J. Mellor-Crummey , N. R. Tallent, HPCTOOLKIT: tools for performance analysis of optimized parallel programs, Concurrency and Computation: Practice & Experience, v.22 n.6, p.685-701, April 2010

[32] Intel Optane, https://www.intel.com/content/www/us/en/architecture-and-technology/optane-memory.html

[33] A. Kougkas, H. Devarajan and X.-H. Sun, "Hermes: A Heterogeneous-Aware Multi-Tiered Distributed I/O Buffering System," in Proc. of 27th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC), Tempe, AZ, USA, pp. 219-230, June 2018.

[34] Compute Express Link (CXL) , https://www.computeexpresslink.org/

[35] Hewlett-Packard. The Machine: A New Kind of Computer. http://www.hpl.hp.com/research/systemsresearch/themachine/.

[36] L. A. Barroso. Warehouse-scale computing: Entering the teenage decade. In Proceeding of the 38th annual International Symposium on Computer Architecture, ISCA'11, New York, NY, USA, ACM, 2011.

[37] T. Newhall, S. Finney, K. Ganchev, and M. Spiegel, "Nswap: A network swapping module for linux clusters," in Proc. Eur. Conf. Parallel Process., 2003, pp. 1160–1169.

[38] S. Dwarkadas, N. Hardavellas, L. Kontothanassis, R. Nikhil, and R. Stets, "Cashmere-VLM: Remote memory paging for software distributed shared memory," in Proc. 13th Int. Parallel Process. Symp. 10th Symp. Parallel Distrib. Process., Apr. 1999, pp. 153–159.

[39] G. Bernard and S. Hamma, "Remote memory paging in networks of workstations," in Proc. SUUG Int. Conf. Open Syst.: Solutions Open Word, 1994.

[40] E. A. Anderson and J. M. Neefe, "An exploration of network RAM," EECS Department, Univ. California, Berkeley, Tech. Rep. UCB/CSD-98-1000, Dec. 1994.

[41] G. Sims, "All about Linux swap space," [Online]. Available: https://www.linux.com/news/all-about-linux-swap-space, 2007.

[42] K. Koh, K. Kim, S. Jeon and J. Huh, "Disaggregated Cloud Memory with Elastic Block Management," in IEEE Transactions on Computers, vol. 68, no. 1, pp. 39-52, 1 Jan. 2019.

[43] P. S. Rao and G. Porter, "Is memory disaggregation feasible? a case study with spark sql," in 2016 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), March 2016, pp. 75–80.

[44] K. Katrinis, D. Syrivelis, D. Pnevmatikatos, G. Zervas, D. Theodoropoulos, I. Koutsopoulos, K. Hasharoni, D. Raho, C. Pinto, F. Espina, S. Lopez-Buedo, Q. Chen, M. Nemirovsky, D. Roca, H. Klos, and T. Berends. Rack-scale disaggregated cloud data centers: The dReDBox project vision. In Design, Automation Test in Europe Conference Exhibition (DATE '16), 2016.