Lawrence Berkeley National Laboratory
LBL Publications

Title
A Case For Intra-rack Resource Disaggregation in HPC

Permalink
https://escholarship.org/uc/item/73x617x8

Journal
ACM Transactions on Architecture and Code Optimization, 19(2)

ISSN
1544-3566

Authors
Michelogiannakis, George
Klenk, Benjamin
Cook, Brandon
et al.

Publication Date
2022-06-30

DOI
10.1145/3514245

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

A Case For Intra-rack Resource Disaggregation in HPC

GEORGE MICHELOGIANNAKIS, Lawrence Berkeley National Laboratory, USA
BENJAMIN KLENK, NVIDIA, USA
BRANDON COOK, Lawrence Berkeley National Laboratory, USA
MIN YEE TEH and MADELEINE GLICK, Columbia University, USA
LARRY DENNISON, NVIDIA, USA
KEREN BERGMAN, Columbia University, USA
JOHN SHALF, Lawrence Berkeley National Laboratory, USA

The expected halt of traditional technology scaling is motivating increased heterogeneity in high-performance computing (HPC) systems with the emergence of numerous specialized accelerators. As heterogeneity increases, so does the risk of underutilizing expensive hardware resources if we preserve today's rigid node configuration and reservation strategies. This has sparked interest in resource disaggregation to enable finer-grain allocation of hardware resources to applications. However, there is currently no data-driven study of what range of disaggregation is appropriate in HPC. To that end, we perform a detailed analysis of key metrics sampled in NERSC's Cori, a production HPC system that executes a diverse open-science HPC workload. In addition, we profile a variety of deep-learning applications to represent an emerging workload. We show that for a rack (cabinet) configuration and applications similar to Cori, a central processing unit with intra-rack disaggregation has a 99.5% probability to find all resources it requires inside its rack. In addition, ideal intra-rack resource disaggregation in Cori could reduce memory and NIC resources by 5.36% to 69.01% and still satisfy the worst-case average rack utilization.

CCS Concepts: • **Computer systems organization** → *Other architectures*; • **Hardware** → *Emerging technologies;*

Additional Key Words and Phrases: Disaggregation, HPC, utilization, memory, LDMS

ACM Reference format:
George Michelogiannakis, Benjamin Klenk, Brandon Cook, Min Yee Teh, Madeleine Glick, Larry Dennison, Keren Bergman, and John Shalf. 2022. A Case For Intra-rack Resource Disaggregation in HPC. *ACM Trans. Arch. Code Optim.* 19, 2, Article 29 (March 2022), 26 pages.
https://doi.org/10.1145/3514245

This work was supported by ARPA-E ENLITENED Program (Project Award No. DE-AR00000843) and the Director, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231.
Authors' addresses: G. Michelogiannakis, B. Cook, and J. Shalf, Lawrence Berkeley National Laboratory, Building 59, 1 Cyclotron road, Berkeley CA 94720; emails: {mihelog, BGCook, jshalf}@lbl.gov; B. Klenk and L. Dennison, NVIDIA, 2788 San Tomas Expy, Santa Clara, CA 95051; emails: {bklenk, ldennison}@nvidia.com; M. Y. Teh, M. Glick, and K. Bergman, 1001 Schapiro Center, for Engineering and Physical Science Research, Columbia University, New York City, NY 10027 United States; emails: {mt3126, msg144}@columbia.edu, bergman@ee.columbia.edu.
Publication rights licensed to ACM. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of the United States government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
1544-3566/2022/03-ART29 $15.00
https://doi.org/10.1145/3514245

1 INTRODUCTION

Future **high-performance computing (HPC)** systems are[1] driven toward heterogeneity of compute and memory resources in response to the expected halt of traditional technology scaling, combined with continuous demands for increased performance [56, 78] and the wide landscape of HPC applications [69]. In the long term, many HPC systems are expected to feature a variety of **graphical processing units (GPUs)**, partially programmable accelerators [62, 77], fixed-function accelerators [11, 61, 68], reconfigurable accelerators such as **field-programmable gate arrays (FPGAs)** [40, 70], and new classes of memory [80] that blur the line between memory and storage technology.

If we preserve our current method of allocating resources to applications in units of statically configured nodes where every node is identical, then future systems risk substantially underutilizing expensive resources. This is because not every application will be able to profitably use specialized hardware resources as the value of a given accelerator can be very application-dependent. The potential for waste of resources when a given application does not use them grows with the number of new heterogeneous technologies and accelerators that might be co-integrated into future nodes.

This observation, combined with the desire to increase utilization even of "traditional" resources, has led to research on systems that can pool and compose resources of different types in a fine-grain manner to match application requirements. This capability is referred to as *resource disaggregation*. In datacenters, resource disaggregation has increased the utilization of GPUs and memory [19, 33, 35, 50, 64, 67]. Such approaches usually employ a full-system solution where resources can be pooled from across the system. While this approach maximizes the flexibility and range of resource disaggregation, it also increases the overhead to implement resource disaggregation, for instance by requiring long-range communication that stresses bandwidth and increases latency [21, 55, 83]. As a result, some work focuses on intra-rack disaggregation [34, 75].

While resource disaggregation is regarded as a promising approach in HPC in addition to datacenters, there is currently no solid understanding of what range or flexibility of disaggregation HPC applications require [13, 47] and what is the expected improvement of resource utilization through this approach. Without any data-driven analysis of the workload, we risk overdesigning resource disaggregation that will make it not only unnecessarily expensive but also may overly penalize application performance due to high latencies and limited communication bandwidth [34, 75].

To that end, we study and quantify what level of resource disaggregation is sufficient for typical HPC workloads and what the efficiency increase opportunity is if HPC embraces this approach, to guide future research into specific technological solutions. We perform a detailed, data-driven analysis in an exemplar open-science, high-ranked, production HPC system with a diverse scientific workload and complement our analysis with profiling key **machine learning (ML)** applications. For our system analysis, we sample key system-wide and per-job metrics that indicate how efficiently resources are used, sampled every second for a duration of three weeks on NERSC's Cori [38]. Cori is a top 20, open-science HPC system that supports thousands of projects, multiple thousands of users, and executes a diverse set of HPC workloads from fusion energy, material science, climate research, physics, computer science, and many other science domains [3]. Because Cori has no GPUs, we also study machine learning (ML) applications executing on NVIDIA GPUs. For these applications, we examine a range of scales, training, and inference while analyzing utilization of key resources.

[1]New article, not an extension of a conference paper.

Based on our analysis, we find that for a system configuration similar to Cori, intra-rack disaggregation suffices the vast majority of the time even after reducing overall resources. In particular, in a rack (cabinet) configuration similar to Cori [38] but with ideal intra-rack resource disaggregation where **network interface controllers (NICs)** and memory resources can be allocated to jobs in a fine-grain manner but only within racks, we show that a **central processing unit (CPU)** has 99.5% probability to find all resources it requires inside its rack. Focusing on jobs, with 20% fewer memory modules and NIC bandwidth per rack, a job has an 11% probability to have to span more racks than its minimum possible in Cori. In addition, in our sampling time range and at worst across Haswell and KNL nodes, we could reduce 69.01% memory bandwidth, 5.36% memory capacity, and 43.35% NIC bandwidth in Cori while still satisfying the worst-case average rack utilization. This quantifies how many resources intra-rack disaggregation can reduce at best.

In summary, our contributions are as follows:

- We perform a data-driven, detailed, and well-rounded study of utilization metrics for key resources in a top 20, open-science, production HPC system.
- We profile important ML applications executing on NVIDIA GPUs focusing on utilization of key resources, which is an emerging workload on HPC systems.
- We propose and measure new metrics relevant to resource disaggregation. In particular, we show how soon a job's resource utilization becomes a good predictor for the future, how quickly node resource utilization changes, and how a job's utilization of different resources correlates.
- We demonstrate spatial usage imbalance of resources to motivate job scheduling policies that use resource utilization as an optimization metric.
- We quantify the probability that a job has to span more racks than the minimum as a function of how aggressively we reduce intra-rack resources in our model system (Cori).
- We quantify how many rack resources we can reduce with ideal intra-rack disaggregation.
- For HPC systems with workloads or hardware configurations substantially different than Cori, our analysis serves as a framework to show what metrics are relevant to resource disaggregation and how to translate system measurements to quantify expected effectiveness of intra-rack disaggregation as a function of how aggressively we reduce resources. This helps repeating our analysis in other HPC systems.
- In summary, we show that for an HPC system similar to Cori, intra-rack disaggregation suffices the vast majority of the time. Demonstrating this insight with concrete data is important to guide future research.

2 BACKGROUND AND RELATED WORK

2.1 Resource Heterogeneity in HPC

Future HPC systems are expected to have a variety of compute and memory resources as a means to reduce cost and preserve performance scaling [78]. The onset of this trend is evident in recent HPC systems that feature partially programmable compute accelerators, with GPUs quickly gaining traction [62, 77]. For instance, approximately a third of the computational throughput of today's top 500 HPC systems is attributed to accelerators. At the same time, recent literature proposes fixed-function accelerators such as for artificial intelligence [11, 61, 68]. Further work has demonstrated reconfigurable accelerators that rely on FPGAs [40, 70] or ASICs [81].

Consequently, past work has examined how job scheduling should consider heterogeneous resource requests [8, 30], how the **operating system (OS)** and runtime should adapt [42, 57], how to write applications for heterogeneous systems [8, 32], how to partition data-parallel applications onto heterogeneous compute resources [48], how to consider the different fault tolerances

of heterogeneous resources [41], how to fairly compare the performance of different heterogeneous systems [44], and what the impact of heterogeneous resources is to application performance [52, 74, 80].

2.2 Resource Disaggregation

Resource disaggregation refers to the ability of a system to pool and compose resources in a fine-grain manner and thus be capable of allocating exactly the resources an application requests. This is in contrast to many systems today where nodes are allocated to applications as a unit with identical fixed-sized resources; any resources inside nodes that the application does not use have no choice but to idle. Following the trend for hardware specialization and the desire to better utilize resources as systems scale up, resource disaggregation across the system or a group of racks has been actively researched and deployed in commercial hyperscale datacenters in Google, Facebook, and others [17, 55, 64]. In addition, many studies focus on disaggregation of GPUs [35] and memory capacity [33, 67].

Resource disaggregation is slowly coming into focus for HPC in addition to the existing hyperscale datacenter deployments. Earlier studies have demonstrated GPU sharing by using virtualization as a means to increase utilization [53, 54]. Further studies identified that memory capacity is frequently over-provisioned in modern HPC systems and thus proposed disaggregating memory modules in a manner similar to hyperscale datacenters [51]. This is further motivated by the observation that many HPC applications already share memory across nodes. Later studies take the first steps to show the promise of resource disaggregation in HPC to increase resource utilization [13, 47].

Related studies demonstrate that emerging high-bandwidth-density silicon-photonic communication technology is a promising means to deliver resource disaggregation with high bandwidth density and low latency [47, 58, 83]. However, just the photonic link bandwidth and efficiency alone cannot fully satisfy the bandwidth, energy, and latency requirements of fully flexible, fine-grain, system-wide disaggregation, making system-wide disaggregation impractical in many cases [19, 21, 55, 83]. For this reason, a few studies argue for rack-level disaggregation of GPUs [34, 75]. In addition, other studies rely on electrical networks [31] in some cases with the aid of **software-defined networks (SDNs)** [17]. Finally, how the software stack should adapt to best take advantage of resource disaggregation remains a significant challenge both for the OS [71] and job scheduler [10, 27].

3 SYSTEM-WIDE ANALYSIS

3.1 Motivation of Cori and Description

Cori is an open-science top 20 HPC system that is operated by NERSC and supports the diverse set of workloads that are in scope for USA's **department of energy (DOE)** [38]. Cori supports in the order of one thousand projects, a few thousand users, and executes a vastly diverse set of representative single-core to full-machine workloads from fusion energy, material science, climate research, physics, computer science, and many other science domains [3]. In this work, we use Cori as a model system to represent the common requirements of HPC systems that are also typical of many other HPC systems in the sciences. We recognize that HPC systems with vastly different workloads or hardware configurations should consider repeating an analysis similar to ours, but the broad user base of this system should provide a microcosm of potential requirements across a wide variety of systems across the world. Furthermore, over the 45-year history of the NERSC HPC facility and 12 generations of systems with diverse architectures, the workload over time has evolved very slowly despite substantial changes to the underlying system architecture, because

the requirements of the scientific mission of such facilities take precedence over the architecture of the facility.

Cori is an approximately 30 Pflop/s Cray XC40 system and consists of 2,388 Intel Xeon "Haswell" and 9,688 Intel Xeon Phi "Knight's Landing" (KNL) compute nodes [73]. Each Haswell node has two 2.3 GHz 16-core Intel Xeon E5-2698 v3 CPUs with two hyperthreads per core. Each KNL node has a single Intel Xeon Phi 7250 central processing unit (CPU) with 68 cores at 1.4 GHz and four hyperthreads per core. Haswell nodes have eight (four per CPU socket) 16 GB 2,133 MHz DDR4 modules with a peak transfer rate of 17 GB/s per module [12, 84]. KNL nodes have 96 GB of 2,400 MHz DDR4 modules and 16 GB of MCDRAM modules. Therefore, Haswell nodes have 128 GB of memory while KNL 112 GB. Each XC40 cabinet housing Haswell and KNL nodes has three chassis; each chassis has 16 compute blades with four nodes per blade.

Cori also has a Lustre file system and uses the Cray Aries interconnect [9]. Similar to many modern systems, hard disk space is already disaggregated in Cori by placing hard disks in a common part of the system [36, 38]. Therefore, we do not study file system disaggregation. Nodes connect to Aries routers through NICs using PCIe 3.0 links with 16 GB/s sustained bandwidth per direction [1, 2, 26, 65]. The four nodes in each blade share a network interface controller (NIC). Cray's Aries in Cori employs a Dragonfly topology with over 45 TB/s of bisection bandwidth and support for adaptive routing. An Aries router connects eight NIC ports to 40 network ports. Network ports operate at rates of 4.7 to 5.25 GB/s per direction [1, 65].

Currently, Cori uses SLURM version 20.02.6 [46] and Cray OS 7.0.UP01. When submitting a job, users request a type of node (Haswell or KNL typically), number of nodes, and a job duration. Users do not indicate an expected memory or CPU usage, but depending on their expectations they can choose "high memory" or "shared" job queues. The maximum job duration that can be requested is 48 h.

3.2 Methodology

We collect system-wide data using the **lightweight distributed metric service (LDMS)** [7] on Cori. LDMS samples statistics in every node every second and also records which job ID each node is allocated to. Statistics include information from the OS level such as CPU idle times and memory capacity usage, as well as hardware counters at the CPUs and NICs. After collection, statistics are aggregated into data files that we process to produce our results. We also calibrate counters by comparing their values against workloads with known metric values, by comparing to literature, and by comparing similar metrics derived through different means, such as hardware counters, the OS, or the job scheduler. For generating graphs, we use python 3.9.1 and pandas 1.2.2.

We sample Cori from April 1 to April 20 of 2021 (UTC-7). However, NIC bandwidth and memory capacity are sampled from April 5 to April 19 due to the large volume of data. Sampling more frequently than 1s or for longer periods would produce even more data to analyze than the few TBs that our analysis already processes. For this reason, sampling for more than three weeks at a time is impractical. Still, our three weeks sampling period is long enough such that the workload we capture is typical of Cori [3]. Unfortunately, since we cannot capture application executable files and input data sets, and because we find job names to not be a reliable indicator of the algorithm a job executes, we do not have precise application information to identify the exact applications that executed in our captured date ranges. For the same reason, we cannot replay application execution in a system simulator.

To focus on larger jobs and disregard test or debug jobs, we also generate statistics only for nodes that are allocated to jobs that use at least four nodes and last at least 2 h. Our data does not include the few tens of high memory or interactive (login) nodes in Cori. Because statistics are sampled no more frequently than every second (at second boundaries), any maximum values we report

Table 1. Cori Measured Data Summary

Metric	Median	Mean	Max	Std Dev	Median	Mean	Max	Std Dev
	Node-wide statistics				**Nodes with jobs ≥4 nodes & ≥2 h**			
Haswell nodes. 41% were allocated to jobs ≥4 nodes and ≥2 h								
Nodes in job	1.0	4.49	1024	17.8	14.0	26.5	1024	39.8
Job duration (hours)	0.15	0.51	73.1	1.68	6.62	8.74	73.1	6.4
CPU idle (%)	49.98	53.73	100	28.77	49.95	51.63	100	26.42
Memory occupied (%)	9.25	14.97	100	16.36	10.08	14.89	100	14.44
NIC Bi-Dir BW (%)	0.01	0.74	99.99	1.90	0.01	0.85	99.96	2.07
Memory BW (GB/s)	0.07	0.84	101.6	2.79	0.09	0.7	89.5	2.0
KNL nodes. 21% were allocated to jobs ≥4 nodes and ≥2 h								
Nodes in job	2.0	13.95	2475	82.37	16	95.35	2064	270.36
Job duration (h)	0.36	2.32	93.98	6.47	5.64	10.82	93.98	11.89
CPU idle (%)	76.53	73.62	100	20.57	76.52	73.36	100	20.67
Memory occupied (%)	26.3	34.27	100	23.73	27.32	35.82	100	24.29
NIC Bi-Dir BW (%)	0.08	0.95	99.99	1.69	0.15	0.99	99.99	1.7

such as in Table 1 are values calculated within each second boundary and thus do not necessary accurately capture peaks that last for less than one second. Therefore, our analysis focuses on *sustained* workload behavior. Memory occupancy and NIC bandwidth are expressed as utilization percentages from the aforementioned per-node maximums. We do not express memory bandwidth as a percentage, because the memory module data is placed in and the CPU that accesses the data affect the maximum possible bandwidth.

We present most data using node-wide **cumulative distribution function (CDF)** graphs that sample each metric at each node individually at every sampling interval. Each sample from each node is added to the dataset. This includes idling nodes, but NERSC systems consistently assign well over 99% of nodes to jobs at any time. This method is useful for displaying how each metric varies across nodes and time, and is insensitive to job size and duration, because it disregards job IDs. Table 1 summarizes our sampled data from Cori. Larger maximum utilizations for KNL nodes for some metrics are partly, because there are more KNL nodes in Cori.

3.3 Job Size and Duration

Figure 1 shows area plots for the size of jobs nodes are assigned to, as a percentage of total nodes. For each node, we determine the size of the job (in nodes) that the node is assigned to. We record that size value for each node in the system individually. Results are shown as a function of time, with one sample every 30s. We only show this data for a subset of our sampling period because of the large volume of data, particularly for KNL nodes, because there are 4× more KNL nodes than Haswell nodes. There are no KNL jobs larger than 2,048 nodes in the illustrated period.

As shown, the majority of jobs easily fit within a rack in Cori. In our 20-day sample period, 77.5% of jobs in Haswell and 40.5% in KNL nodes request only one node. Similarly, 86.4% of jobs in Haswell and 75.9% in KNL nodes request no more than four nodes. 41% of jobs in Haswell and 21% for KNL nodes use at least four nodes *and* execute for at least 2 h. Also, job duration (Table 1) is multiple orders of magnitude larger than the reconfiguration delay of modern hardware technologies that can implement resource disaggregation, such as photonic switch fabrics that can reconfigure from a few tens of microseconds to a few tens of nanoseconds [20, 60]. Our data set includes a handful of jobs in both Haswell and KNL nodes that use special reservation to exceed

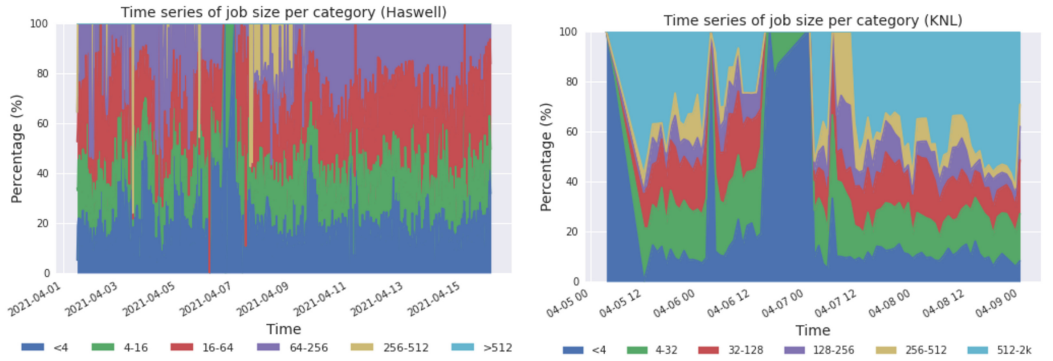

Fig. 1. Area plots for the size of jobs nodes are assigned to, for Haswell (left) and KNL (right) nodes.

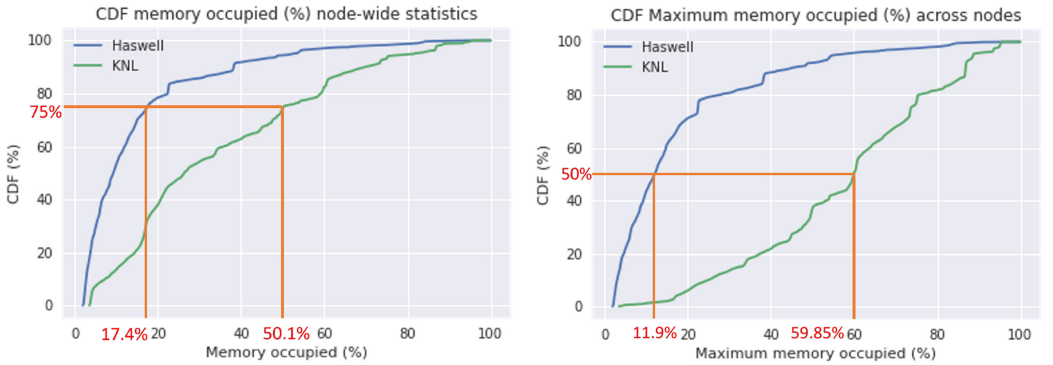

Fig. 2. Left: Node-wide memory occupied. Right: Maximum memory occupied across all nodes sampled every 1s. Red lines illustrate a method for interpreting CDF graphs: Starting from the Y axis at a point that represents a percentage of jobs, drawing a horizontal line to the right, and determining the point in the X axis where the graph lines intercept our drawn horizontal line.

the normal maximum number of hours for a job. In Alibaba's systems, batch jobs are shorter with 99% of jobs under two minutes and 50% under ten seconds [36].

3.4 Memory Capacity

Figure 2 (left) shows a CDF of node-wide memory occupied by user applications, as calculated from "proc" reports of total minus available memory [14]. Figure 2 (right) shows a CDF of maximum memory occupancy among all nodes, sampled every 1s. Occupied memory is expressed as percentage utilization from the capacity available to user applications (total memory in "proc" reports), and thus does not include memory reserved for the firmware and kernel binary code.

As shown by the red example lines, three quarters of the time, Haswell nodes use no more than 17.4% of on-node memory and KNL nodes 50.1%. Looking at the maximum occupancy among all nodes in each sampling period, half of the time that is no more than 11.9% for Haswell nodes and 59.85% for KNL nodes. Looking at individual jobs instead of nodes (not illustrated), 74.63% of Haswell jobs and 86.04% of KNL jobs never use more than 50% of on-node memory.

Our analysis is in line with past work. In four Lawrence Livermore National Laboratory clusters, approximately 75% of the time, no more than 20% of memory is used [67]. Other work observed

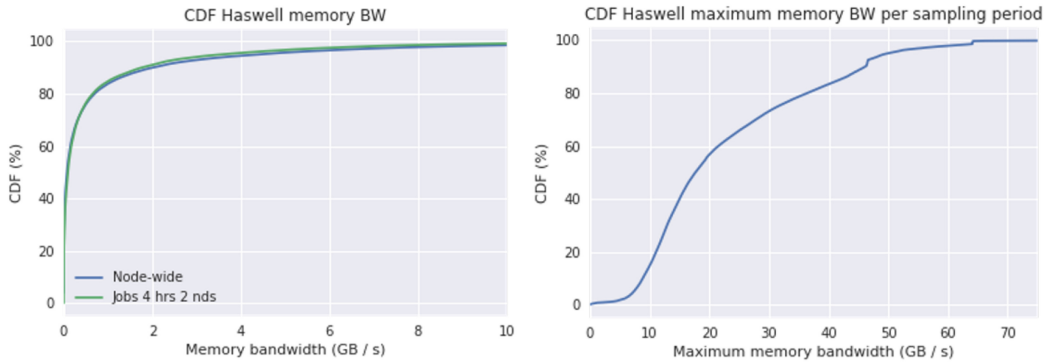

Fig. 3. Left: Node-wide bi-directional (read plus write) memory data bandwidth for Haswell nodes. Node-wide statistics are shown separately for (i) all nodes and (ii) only for nodes assigned to jobs of at least four nodes and 2 h. Right: Maximum bi-directional memory data bandwidth across all Haswell nodes, sampled every 1s.

that many HPC applications use in the range of hundreds of MBs per computation core [85]. Interestingly, Alibaba's published data [36] show that a similar observation is only true for machines that execute batch jobs, because batch jobs tend to be short and small. In contrast, for online jobs or co-located batch and online jobs, the minimum average machine memory utilization is 80%.

These observations suggest that memory capacity is overprovisioned *by average*. Overprovisioning of memory capacity is mostly a result of (i) sizing memory per node to satisfy memory-intensive applications or phases of applications that execute infrequently but are considered important, and (ii) allocating memory to nodes statically.

3.5 Memory Bandwidth

Figure 3 (left) shows CDFs of bi-directional (read plus write) node-wide memory bandwidth for data transfers to and from memory. These results are calculated by the **last-level cache (LLC)** line size and the load and store LLC misses reported by "perf" counters [25] and crosschecked against PAPI counters [16]. Due to counter accessibility limitations, LLC statistics are only available for Haswell nodes. In Haswell, hardware prefetching predominantly occurs at cache levels below the LLC [43]. Thus, prefetches are captured by the aforementioned LLC counters we sample.

As shown, three quarters of the time Haswell nodes use at most 0.46 GB/s of memory bandwidth. 16.2% of the time Haswell nodes use more than 1 GB/s. The distribution shows a long tail, indicative of bursty behavior. In addition, aggregate read bandwidth is 4.4× larger than aggregate write bandwidth and the majority of bursty behavior comes from reads. If we focus on individual jobs (not illustrated), then 30.9% of jobs never use more than 1 GB/s per node.

To focus on the worst case, Figure 3 (right) shows a CDF of the maximum memory bandwidth among all Haswell nodes, sampled in 1s periods (i.e., sustained maximums). As shown, half of the time the maximum bandwidth among all Haswell nodes is at least 17.6 GB/s and three quarters of the time 31.5 GB/s. In addition, sustained system-wide maximum memory bandwidth at 30s time windows within our overall sampling period rarely exceeds 40 GB/s, while 30s average values rarely exceed 2 GB/s.

Inevitably, our analysis is sensitive to the configuration of Cori's Haswell nodes. Faster or more computation cores per node would place more pressure on memory bandwidth [18] as already ev-

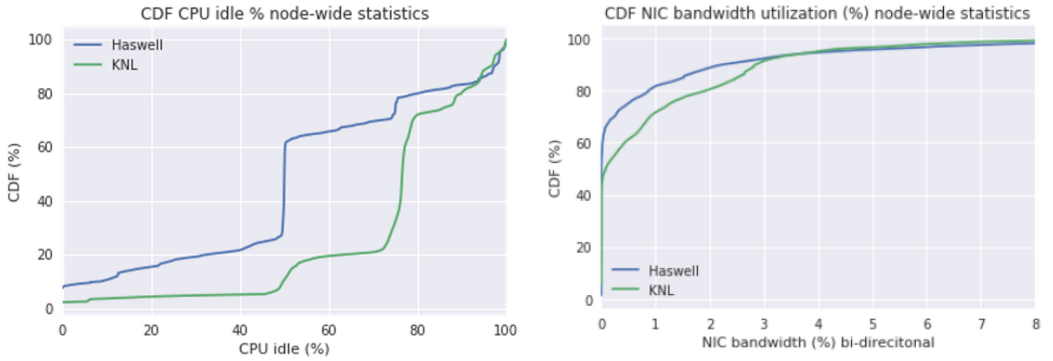

Fig. 4. Left: CDF of node-wide CPU idle % measurements. Idle percentage is calculated as the average percentage among all hardware threads in the node. Right: CDF of node-wide NIC bandwidth utilization compared to the bi-directional 16 GB/s per-node maximum.

ident in GPUs [29]. Still, our data hints that memory bandwidth is another resource that may be overdesigned *by average* in HPC. However, we cannot relate our findings to the impact to application performance if we were to reduce available memory bandwidth. That is because applications may exhibit brief but important phases of high memory bandwidth usage that may be penalized if we reduce available memory bandwidth [18].

3.6 CPU Utilization

Figure 4 (left) shows a CDF of average idle time among all compute cores in nodes, expressed as a percentage. These statistics were generated using "proc" reports [14] of idle kernel cycles for each hardware thread. To generate a sample for each node every 1s, we average the idle percentage of all hardware threads in each node. As shown, about half of the time Haswell nodes have at most a 49.9% CPU idle percentage and KNL nodes 76.5%. For Haswell nodes, average system-wide CPU idle time in each sampling period never drops lower than 28% in a 30 s period and for KNL 30% (not illustrated). These statistics are largely due to the two hardware threads per compute core in Haswell and four in KNL, because in Cori 80% of the time Haswell nodes use only one hardware thread and 50% in KNL [3]. Similarly, many jobs reserve entire nodes but do not use all cores in those nodes. Datacenters have also reported 28%–55% CPU idle in the case of Google trace data [66] and 20%–50% most of the time in Alibaba [36].

3.7 Network Bandwidth

We measure per-node injection bandwidth at every NIC by using hardware counters in the Cray Aries interconnect [23]. Those counters record how many payload bytes each node sent to and received from the Aries network. We report results for each node as a percentage utilization of the maximum per-node NIC bandwidth of 16 GB/s per direction [2, 65]. We also verify against similar statistics generated by using NIC flit counters and multiplying by the flit size. In Cori, access to the global file system uses the Aries network so our statistics include file system accesses.

Figure 4 (right) shows a CDF of node-wide NIC bandwidth utilization. As shown, 75% of the time Haswell nodes use at most 0.5% of available NIC bandwidth. For KNL nodes the latter percentage becomes 1.25%. In addition, NIC bandwidth consistently exhibits a sustained bursty behavior. In particular, in a two-week period, sustained 30s average NIC bandwidth in about 60 separate occurrences increased by more than 3× compared to the overall average.

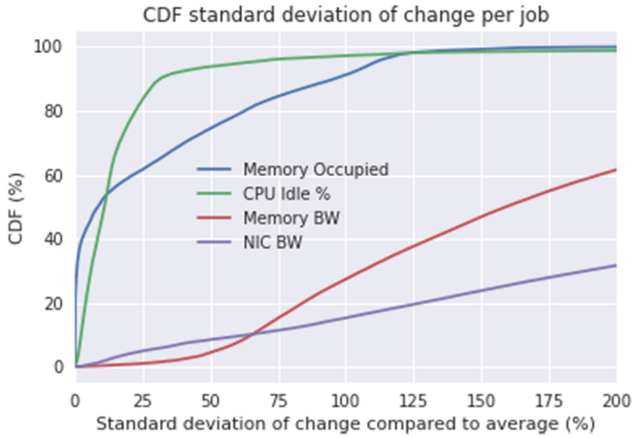

Fig. 5. CDF of the standard deviation of all the values throughout each job's execution, for each metric. Standard deviation is expressed as a percentage of the per-job average for each metric.

3.8 Variability of Job Requirements

In this section, we analyze *how much* metrics change across a job's lifetime. Figure 5 shows a CDF of the standard deviation of all values throughout each job's execution, calculated separately for different metrics. This graph was generated by calculating the standard deviation of values for each metric that each job has throughout the job's execution. A high standard deviation indicates that the metric varies substantially throughout the job's execution. To normalize for different absolute values of each job, standard deviation is expressed as a percentage of the per-job average value for each metric. A value of 50% indicates that the job's standard deviation is half of the job's average for that metric.

As shown, occupied memory and CPU idle percentages do not highly vary during job execution, but memory and NIC bandwidths do. The variability of memory and NIC bandwidths is intuitive, because many applications exhibit phases of low and high memory bandwidth. Network and memory bandwidth have been previously observed to have bursty behavior for many applications [45, 49, 82]. In contrast, once an application completes reserving memory capacity, the reservation's size typically does not change significantly until the application terminates.

These observations are important for provisioning resources for disaggregation. For metrics that do not considerably vary throughout a job's execution, average system-wide or per-job measurements of those metrics are more representative of future behavior. Therefore, provisioning for average utilization, with perhaps an additional factor such as a standard deviation, likely will satisfy application requirements for those metrics for the majority of the time. In contrast, metrics that vary considerably have an average that is less representative. Therefore, for those metrics resource disaggregation should consider the maximum or near-maximum value.

3.9 Rate of Change of Metrics

As a next step, we analyze *how quickly* metrics change across nodes. This is a useful metric to indicate how quickly future hardware implementations of resource disaggregation should be able to reconfigure to satisfy changing resource demands. To that end, we calculate the percentage change of our metrics from one 1s sample to the next. For instance, if memory occupied in one sample is 4 GB and in the next sample it becomes 5 GB, we add 20% to our rate of change dataset.

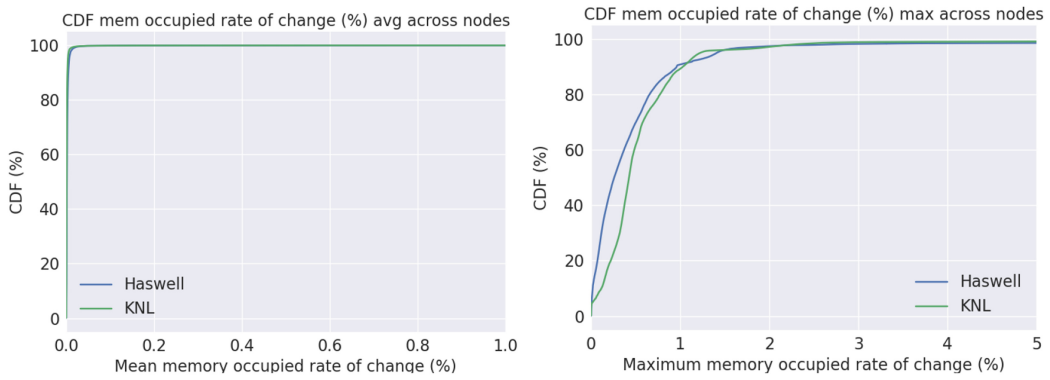

Fig. 6. Left: CDF of the node-wide rate of change of occupied memory across 1s intervals. Right: CDF of the maximum rate of change across all nodes every 1s.

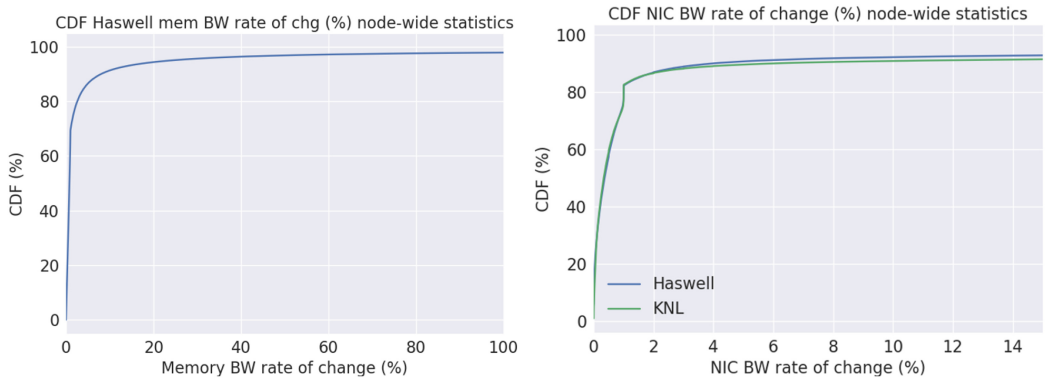

Fig. 7. Node-wide rate of change CDFs of memory bandwidth (left) and NIC bandwidth (right) across 1s intervals.

Due to the computational intensity of this analysis and our large data set, memory occupancy and NIC bandwidth graphs only consider data from April 5 to 9.

Figure 6 shows the rate of change for occupied memory both node-wide and maximum across nodes. As shown, occupied node memory is a metric that rarely changes substantially. Ninety-eight percent of the time, occupied memory changes by less than 0.1% from one second to the next. Even if we consider the maximum rate of change across all nodes every 1 s, 90% of the time the maximum rate of change is no more than 1%.

Figure 7 shows node-wide CDFs for the rate of change of memory and NIC bandwidth. From one 1 s sample to the next, memory and NIC bandwidth do not substantially change with a probability of 80%. However, 2% (10%) of the time from one second to the next, the change exceeds 100% for memory (NIC) bandwidth. This indicates a burst of usage from a low near-idle value to a high value. In fact, the CDF distributions have long tails. These results quantify the magnitude and frequency of the bursty nature of both memory and NIC bandwidths.

3.10 Spatial Load Imbalance

To show the results of current job scheduling policies that do not take into account resource utilization, Figure 8 shows a heatmap of average, standard deviation, and maximum memory occupancy

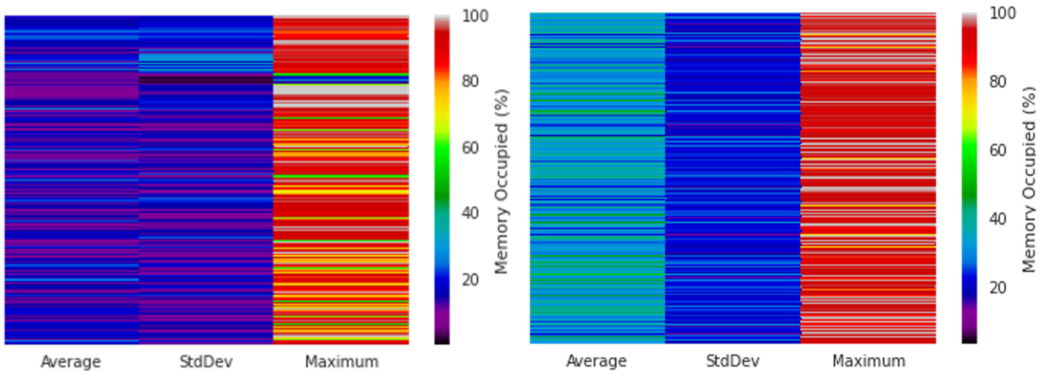

Fig. 8. Heatmaps of average, standard deviation, and maximum memory occupancy (%) for Haswell (left) and KNL (right) nodes. Each row represents a node. Rows are sorted by node ID so neighboring rows typically correspond to nodes that are physically close.

for Haswell and KNL nodes, sampled across our 20-day sample period. As shown, memory occupancy is unequal across nodes. In fact, even the standard deviation of maximum memory occupancies is 15.22% for Haswell nodes and 4.76% for KNL nodes. KNL nodes are more balanced in memory occupancy but still show considerable imbalance. These observations motivate future work on resource disaggregation-aware job schedulers. Even though they are not illustrated, heatmaps for memory bandwidth and NIC bandwidth provide similar insights. For instance, the standard deviation of the maximum memory bandwidth across Haswell nodes is 17.34 GB/s and for NIC bandwidth 23.14% across all nodes.

3.11 How Soon Jobs Become Predictable

Subsequently, we ask the question of *how soon* a job's resource usage becomes a good predictor (i.e., indicative) of its future usage. This is useful to determine how soon a system can observe a job's behavior and make an informed decision on how many resources to assign to it. Figure 9 illustrates the percentage runtime of a job until each of the three metrics shown reach at least 80% of the job's average usage throughout its lifetime. For instance, in the case of memory occupancy, a value of 20% means that at 20% of its runtime, a job reserves 80% or more memory of the job's average memory occupancy, calculated throughout the job's entire runtime.

As shown by the CDFs, the majority of jobs become predictable early. For instance, over 85% of jobs reach their 80% of average memory occupancy by 20% of their runtime. This number becomes 92% for NIC bandwidth and 79% for memory bandwidth. In fact, over 60% of jobs are predictable by 5% of their runtime for all three metrics. This observation is encouraging in that systems that rely on observing job resource usage do not have to wait long after a job starts to gain a good understanding about a job's resource usage.

3.12 Temporal Behavior

In this section, we investigate the temporal variability of average and maximum memory bandwidth across Haswell nodes. We choose memory bandwidth to focus on memory usage, and because we observe that memory occupancy does not vary substantially with time, as we discuss in Section 3.9. Temporal variability is useful to indicate how many valleys and peaks appear in system-wide usage and what is their spacing.

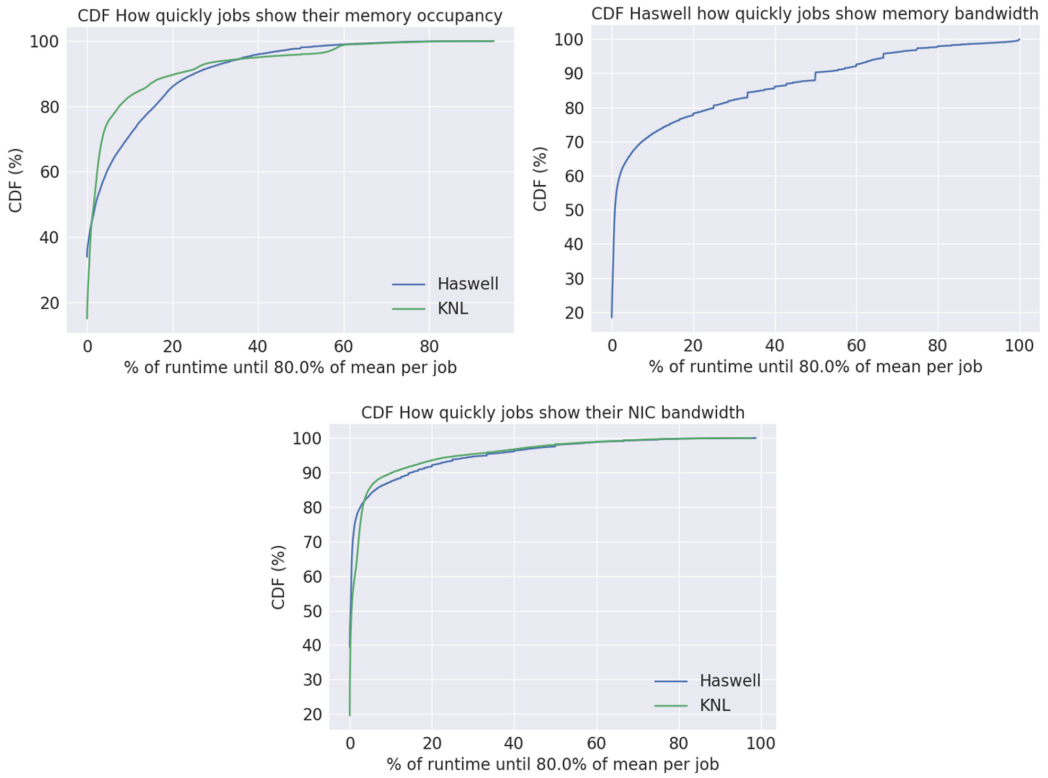

Fig. 9. Percentage runtime until jobs reach 80% of their lifetime mean for memory occupancy, memory bandwidth, and NIC bandwidth.

Figure 10 shows a time–series graph for the first 15 days of our sampling period, to better observe behavior in detail. Maximum memory bandwidth shows bursts, usually correlated with a burst in average memory bandwidth. Maximum memory bandwidth also shows a noticeable 30s variability, as indicated by the shaded region. Average memory bandwidth remains low, but does show short bursts of 2× or 3× a few times a day. Relatively, these are more frequent and more intense bursts than maximum memory bandwidth, but not significantly. Also, an important observation is that even though system-wide average memory bandwidth shows these bursts, it never exceeds 5 GB/s.

3.13 Correlation Between Metrics

In this section, we investigate whether there is correlation between how jobs use different resources. Figure 11 shows scatterplots that correlate different metrics. Each data point represents a single job, defined by a separate job ID in Cori's job queue. A data point's X and Y values are that job's average resource utilization (for each metric in the axes) throughout its runtime. A job's average IPC is measured by averaging the IPC of all cores in nodes assigned to the job. A core's IPC is calculated by dividing the core's total executed instructions by the number of cycles, for the duration the core was assigned to each job. A core's instructions and cycles are reported by PAPI counters [16].

Graphs also include the calculated correlation factor between the two metrics in each graph for all job samples. The correlation factor ranges from −1.0 to 1.0, where −1 means strong negative correlation, 0 means no correlation, and 1.0 means strong positive correlation. For instance, no

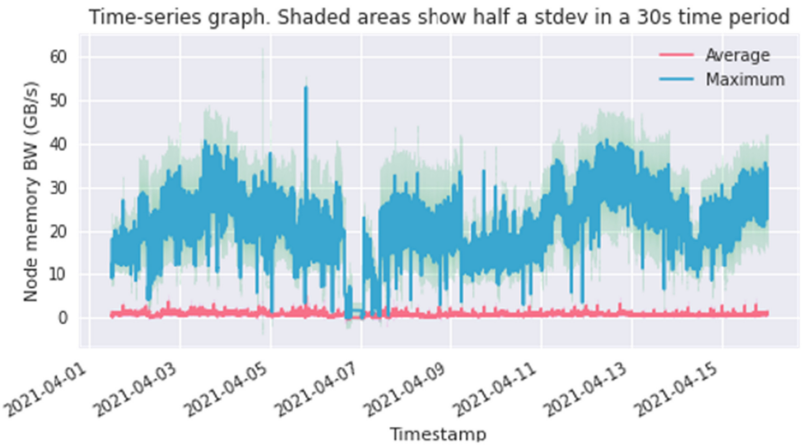

Fig. 10. A time-series graph of average and maximum memory bandwidth for Haswell nodes. At each second, the average and maximum are calculated across all Haswell nodes over a 30 s period to show sustained values. Each metric is shown as a solid line and is surrounded by a shaded area. The shaded area represents half a standard deviation of the value's range within 30 s of each sample. A small shaded area around a sample indicates low variability of the metric within 30 s of the sample.

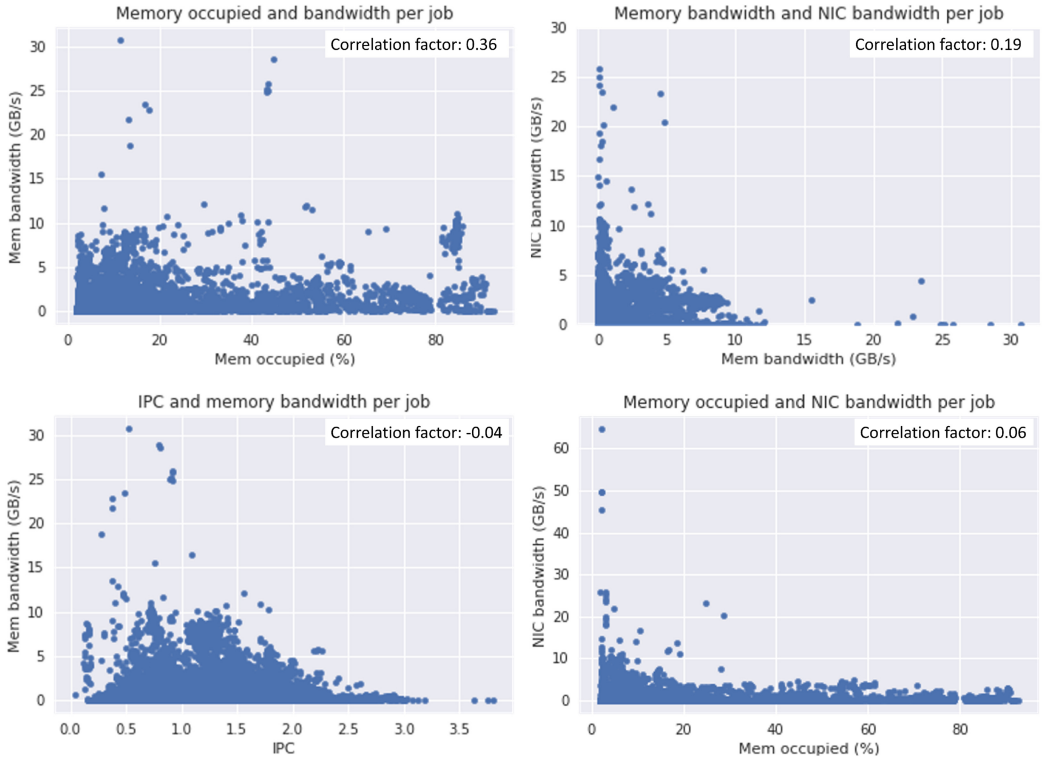

Fig. 11. Scatterplots that correlate memory occupancy, memory bandwidth, NIC bandwidth, and IPC. Each data point denotes the average usage of a single job. Graphs include the correlation factor of all job samples.

correlation means that one value increasing shows no tendency for the other value to change in either direction.

As shown, jobs that occupy more memory have a small probability to use more memory bandwidth than other jobs. Similarly, jobs that use higher NIC bandwidth are slightly more likely to use higher memory bandwidth. Finally, there is practically no correlation between IPC and memory bandwidth as well as memory occupancy and NIC bandwidth.

4 DEEP-LEARNING APPLICATIONS ANALYSIS

Many of today's HPC systems deploy GPUs as accelerators [76]. At the same time, an increasing amount of ML workloads execute on HPC systems and cloud providers continue to increase their HPC offerings. Therefore, to supplement our system-wide data, we analyze a set of ML workloads that run on a GPU-accelerated system to illustrate resource underutilization and show how much disaggregation these workloads motivate. We study ML/DL workloads, as we believe these are a significant share of applications in future HPC systems. We therefore conduct a set of controlled experiments using representative workloads on a typical GPU-accelerated system.

4.1 Methodology

The workloads we study are part of the MLPerf benchmark suite version 0.7 [59]. MLPerf is maintained by a large consortium of companies that are pioneering AI in terms of neural network design and system architectures for both training and inference. The workloads that are part of the suite have been developed by world-leading research organizations and are representative of workloads that execute in actual production systems. We select a range of different neural networks, representing different applications:

Transformer [79]: Transformer's novel multi-head attention mechanism allows for better parallel processing of the input sequence and therefore faster training times, but it also overcomes the vanishing gradient issue that typical RNNs suffer from [39]. It is for these reasons that Transformers became state-of-the-art for natural language processing tasks. Such tasks include machine translation, time series prediction, as well as text understanding and generation. Transformers are the fundamental building block for networks like **bidirectional encoder representations from transformers (BERT)** [24] and GPT [15]. It has also been demonstrated that Transformers are used for vision tasks [28].

BERT [24]: The BERT network only implements the encoder and is designed to be a language model. Training is often done in two phases; the first phase is unsupervised to learn representations and the second phase is then used to fine-tune the network with labeled data. Language models are deployed in translation systems and human-to-machine interactions. We focus on supervised fine-tuning.

ResNet50 [37]: vision tasks, particular image classification, were among the first to make DL popular. First developed by Microsoft, ResNet50 (Residual Network with 50 layers) is often regarded as a standard benchmark for DL tasks and is one of the most used DL networks for image classification and segmentation, object detection, and other vision tasks.

DLRM [63]: the last benchmark in our study is the **deep-learning recommendation model (DLRM)**. Recommender systems differ from the other networks in that they deploy vast embedding tables. These tables are sparsely accessed before a dense representation is fed into a more classical neural network. Many companies deploy these systems to offer customer recommendations based on their history of items they bought or content they enjoyed.

All workloads are ran using the official MLPerf docker containers on the datasets that are also used for the official benchmark results. Given the large number of hyperparameters to run these

networks, we refer to the docker containers and scripts that are used to run the benchmarks. We only adapt batch sizes and denote them in our results.

4.1.1 Test System for Our Experiments. We run our experiments on a cluster comprising of NVIDIA DGX1 nodes [4]. Each node is equipped with eight Volta-class V100 GPUs with 32 GB of HBM memory. GPU of a single node are interconnected in a hybrid mesh-cube using NVLink, giving each GPUs a total NVLink bandwidth of 1.2 Tbps. Furthermore, each node provides four ConnectX-4 Infiniband EDR NICs for a total node-to-node bandwidth of 400 Gbps. We run experiments on up to 16 nodes. A DGX1 system is a typical server configuration for training and therefore representative of real deployments. For example, Facebook deploys similar systems [72]. We note that inference is often run on a different system and a more service-oriented architecture. However, we still run inference on the same system to highlight differences in resource allocation, showing that a disaggregated system can support both workloads efficiently.

4.1.2 Measurement of Resource Utilization. CPU and host memory metrics are collected through psutil [6], GPU metrics (utilization, memory, NVLink, and PCIe) come from the NVML library [5], and InfiniBand statistics are collected through hardware counters provided by the NICs. NVML also provides a "memory utilization" metric, which is a measure of how many cycles the memory is busy with processing a request. A utilization of 100% means that the memory is busy processing a request every cycle.

We launch a script that gathers the metrics at a sampling period of 1s before starting the workload. We found the overhead of profiling to be small and therefore negligible. Although a full training run of the aforementioned neural nets can take many hours, the workloads are repetitive so it is sufficient to limit measurements to a shorter period.

For compute utilization, we only consider utilization above 5% in the geometric mean calculations. This excludes idle periods at the beginning, for example. For memory capacity, we take the the maximal capacity during the workload's runtime. Similar to our Cori results, here we also find that once memory is allocated, it remains mostly constant. Bandwidth is shown as the geometric mean. We exclude outliers by filtering out values that are outside of 1.5× interquartile range (IQR), including values of zero. It is important to note that a sampling period of 1s is insufficient to determine peak bandwidth for high-speed links. However, it is still a valuable proxy for average utilization.

4.2 Training Versus Inference

Machine learning typically consists of two phases: training and inference. During training the network learns and optimizes parameters from a carefully curated dataset. Training is a throughput-critical task and input samples are batched together to increase efficiency. Inference, however, is done on a trained and deployed model and is often sensitive to latency. Input batches are usually smaller and there is less computation and lower memory footprints, as no errors need to be backpropagated and parameters are not optimized.

We measure various metrics for training and inference runs for BERT and ResNet50. Results are shown in Figure 12 (left). GPU utilization is high for BERT during training and inference phases, both in terms of compute and memory capacity utilization. However, the CPU compute and memory capacity utilization is low. ResNet50, however, shows large CPU compute utilization, which is also higher during inference. Inference requires significantly less computation, which means the CPU is more utilized compared to training to provide the data and launch the work on the GPU. Training consumes significantly more GPU resources, especially memory capacity. This is not surprising, as these workloads were designed for maximal performance on GPUs. Certain parts of the system, notably CPU resources, remain underutilized, which motivates disaggregation.

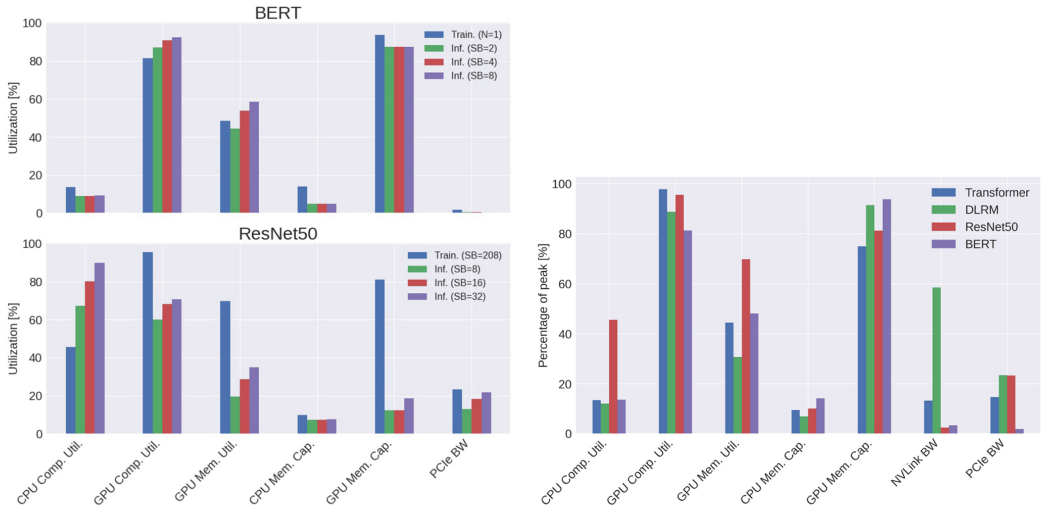

Fig. 12. Left: Resource utilization during single-node training and inference. Results are shown for a single GPU. Right: Resource utilization during single-node training of various MLPerf networks.

Further, training and inference have different requirements and disaggregation helps to provision resources accordingly.

The need for disaggregation is also evident in NVIDIA's introduction of multi-instance GPU [22], which allows to partition a GPUs into seven independent and smaller GPUs. Our work takes this further and considers disaggregation at the rack scale.

4.3 Training Resource Utilization

Figure 12 (right) shows resource utilization during the training of various MLPerf workloads. These benchmarks are run on a single DGX1 system with 8 Volta V100 GPUs. While we can generally observe that CPU utilization is low, GPU utilization is consistently high across all workloads. We also depict bandwidth utilization of NVLink and PCIe. We note that bandwidth is shown as an average effective bandwidth across all GPUs for the entire measurement period. In addition, we can only sample in intervals of 1 s, which limits our ability to capture peaks in high-speed links like NVLink. Nonetheless, we can observe that overall effective bandwidth is low, which suggests that links are not highly utilized by average.

All of the shown workloads are data-parallel, while DLRM also implements model parallelism. In data parallelism, parameters need to be reduced across all workers, resulting in an all-reduce operation for every optimization step. As a result, the network can be underutilized during computation of parameter gradients. The highest bandwidth utilization is from DLRM, which is attributed to the model parallel phase and its all-to-all communication.

4.4 Inter-node Scaling

Another factor of utilization is inter-node scaling. We run BERT and ResNet50 on up to 16 DGX1 systems, connected via InfiniBand EDR. The results are depicted in Figure 13. For ResNet50, we also distinguish between weak scaling and strong scaling. BERT is shown for weak scaling only. Weak scaling is preferred as it generally leads to higher efficiency and utilization, as shown by our results. In data parallelism, this means we keep the number of input samples per GPU, referred to as **sub-batch (SB)**, constant, and while we scale-out the effective global batch size increases. At

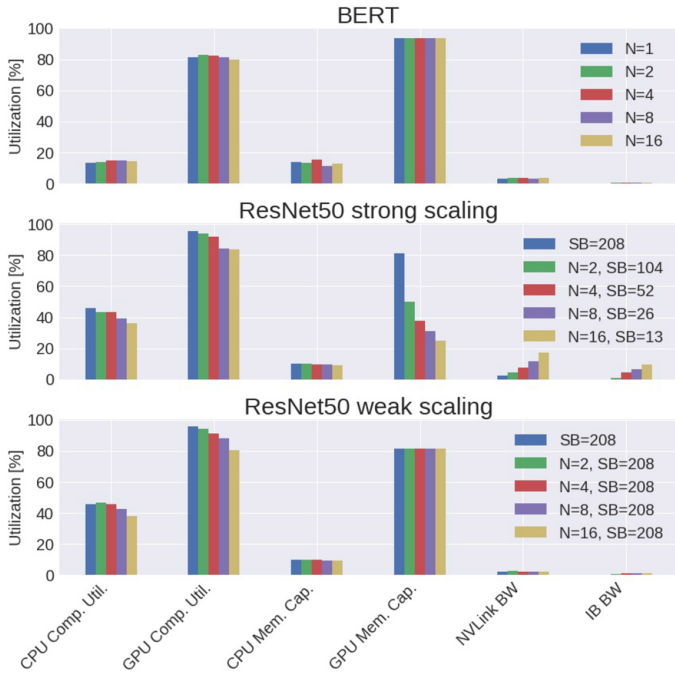

Fig. 13. Resource utilization during multi-node training for BERT and ResNet50.

some point the global batch size reaches a critical limit after which the network stops converging or converges slower so that any performance benefit diminishes. At this point, strong scaling becomes the only option to further reduce training time.

As Figure 13 shows, strong scaling increases the bandwidth requirements, both intra- and inter-node, but reduces compute and memory utilization of individual GPUs. While some underutilization is still beneficial in terms of total training time, it eventually becomes too inefficient. Many neural nets train for hours or even days on small-scale systems, rendering large-scale training necessary. This naturally leads to some underutilization of certain resources, which motivates disaggregation to allow resources to be used for other tasks.

5 TRANSLATING TO RESOURCE DISAGGREGATION REQUIREMENTS

5.1 System-wide Data

To understand what our observations mean for resource disaggregation, we assume a Cori cabinet that contains Haswell nodes [38]. That is, each cabinet has 384 Haswell CPUs, 768 memory modules of 17 GB/s and 16 GB each, and 192 NICs that connect at 16 GB/s per direction to nodes [2, 26]. We use memory bandwidth measurements from Haswell nodes, and memory capacity and NIC bandwidth from both Haswell and KNL nodes. Since a benefit of resource disaggregation is increasing the utilization factor that can help to reduce overall resources, we define and sweep a resource reduction factor percentage for non-CPU resources. For instance, a resource reduction factor of 50% means that our cabinet has half the memory and NIC resources of a Cori Haswell cabinet. The resource reduction factor is a useful way to illustrate the tradeoff between the range (or reachability) of disaggregation and how aggressively we reduce available resources. We assume resource disaggregation hardware capable of allocating resources in a fine-grain manner, such as fractions of memory capacity of the same memory module to different jobs or CPUs.

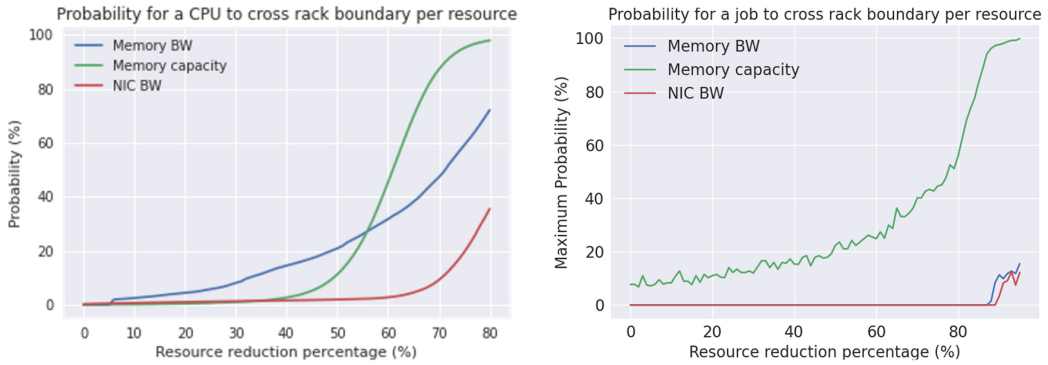

Fig. 14. Left: The probability that any one CPU in a Cori Haswell cabinet (rack) has to cross the cabinet boundary to find additional resources of each type. This is shown as a function of the percentage reduction of cabinet memory modules and NICs. Right: The maximum (worst-case) probability that a job spans more racks than the minimum to find enough resources of each type.

5.1.1 Probability to Cross the Rack Boundary. We perform two kinds of analyses. The first analysis is agnostic to job characteristics and quantifies the probability that a CPU will have to connect to resources in other racks. This more readily translates to inter-rack bandwidth requirements for system-wide disaggregation. To that end, we make the assumption that a CPU can be allocated to any application running in the system with uniform random probability. We use the CDFs of resource usage from Section 3 as probability distributions.

As shown in Figure 14 (left), with no (0%) resource reduction a CPU has to cross a rack 0.01% of the time to find additional memory bandwidth, 0.16% to find additional memory capacity, and 0.28% to find additional NIC bandwidth. With a 50% resource reduction factor, these numbers become 20.2%, 11%, and 2%, respectively.

Our second analysis focuses on behavior of jobs to quantify the probability that a job will have to span more racks to find resources compared to the minimum number of racks the job can occupy based on its requested number of nodes. In addition, this way we capture the correlation of different resource types that are assigned to the same job. In particular, we sample 210 randomly chosen timestamps from our dataset. For each timestamp, we record which jobs are executing, their resource utilization, and their size in nodes. Because KNL jobs lack memory bandwidth measurements, we focus only on Haswell jobs. For each job's resource utilization metric, such as memory occupancy, we measure the maximum utilization among all nodes reserved to the job throughout the job's execution. We use a job's maximum node utilization to account for the worst-case scenario. Then, we execute 16 of the following random experiments for each randomly chosen timestamp and report the *maximum* (worst-case) probability across experiments.

In each random experiment, we look at all jobs running at the chosen timestamp and assign them to nodes. Though job placement that prioritizes resource disaggregation would likely yield better results, we leave this as future work, since such job placement policies are still emerging. Therefore, in each experiment (i.e., one iteration of job placement for a given timestamp), we allocate racks of Haswell nodes and assign them to jobs in a random fashion. For each job, the number of nodes is the same as when it was running on Cori. The random placement pessimistically assumes that job schedulers are agnostic to resource disaggregation. For each randomly chosen rack, we reserve resources to cover the job's utilization at the randomly chosen timestamp. If the job still has resource requirements remaining, then we continue by allocating another random rack. At the end of each experiment, we record the percentage of jobs that had to allocate resources

Table 2. How Much We Can Reduce Each Resource Type and Still Satisfy the Worst-case Average Rack Utilization

	Mem occupied	Mem BW	NIC BW
Haswell	39.95%	69.01%	59.17%
KNL	5.36%	—	43.35%

from more racks than the minimum number of racks they could be placed in based on their size in nodes. These are the jobs that had to span more racks because of a lack of resources. This analysis is performed for each resource type separately.

Results are shown in Figure 14 (right). Without reducing resources, there is still some worst-case probability for a job to span more racks than the minimum to allocate memory capacity because of unfavorable random job placement. However, the average across our random experiments remains near zero (not illustrated). With a 20% reduction, the same worst-case probability becomes 11%, with a 50% reduction the probability becomes 22.3%, and with 80% reduction the probability becomes 56%. For NIC and memory bandwidth, for up to a reduction of 85% the probability is near zero. For a reduction of 95%, the probability for NIC bandwidth is 12.2% and for memory bandwidth 15.5%. While these results are sensitive to our assumptions and random choices of our algorithm, they indicate that intra-rack disaggregation suffices the majority of the time except when reducing resources aggressively.

5.1.2 Resource Reduction with Disaggregation. To illustrate potential benefits and thus further motivate intra-rack resource disaggregation, we use per-node statistics to calculate an average resource utilization *of each rack*. We do this for the node-to-rack mapping in Cori but also for a large set of randomized mappings of nodes to racks, because Cori's scheduler is agnostic to resource disaggregation. This increases the computational complexity of this analysis, so we only perform it across four days of our data set. For each mapping, we use node statistics to calculate per-rack utilization for each resource, by average across the four-day sampling period. We then take the maximum utilization for each resource among all the node-to-rack mappings and all racks within each mapping, to capture the worst-case (highest) rack utilization for each metric. We use that maximum to derive how much we could reduce in-rack resources and still satisfy the worst-case average rack utilization in our chosen four-day period. This does not indicate that no job will have to cross a rack boundary to find resources, or that application performance will be unaffected. Instead, this analysis focuses on resource utilization statistics. Memory bandwidth reduction is based on the per-node theoretical maximum of 136 GB/s from the eight memory modules.

The percentage we can reduce each resource and still satisfy the worst-case average rack utilization is shown in Table 2. As shown, except for memory capacity in KNL racks, the other resources can be reduced substantially. Resource disaggregation-aware job scheduling may further improve these findings.

5.2 Deep-learning Application Data

Based on Section 4, there are also opportunities for rack-level disaggregation in ML workloads and GPU-accelerated systems. We observe a strong variability of resource requirements among different neural networks and therefore application domains but also among inference and training (Figure 12). Inference usually requires higher CPU-to-GPU ratios than training and uses less GPU memory. In contrast, training leads to high GPU utilization in terms of computation and memory but lower CPU utilization. However, this also varies with the workload. Job schedulers can use

a disaggregated system by allocating more CPUs and less GPU memory for inference and give the remaining resources to a training job, which requires lots of memory and generally less CPU time.

While we observe that GPU utilization is generally high, CPU resources and network bandwidth are underutilized most of the time. Although we cannot determine peak bandwidth demands with our methodology, we see that the average bandwidth utilization over longer periods is low. As we strong-scale training to multiple nodes, the bandwidth demands increase but other resources become less utilized, such as GPU memory (Figure 13). A disaggregated system allows us to provision unused GPU memory for other jobs, and since bandwidth depends on the scale, we can give more bandwidth to large-scale jobs and less bandwidth to small-scale jobs.

6 DISCUSSION

6.1 Limitations of Sampling a Production System

Sampling a production system has significant value, because it demonstrates what users actually execute and how resources are used in practice. At the same time, sampling a production system inevitably has practical limitations. For instance, it requires privileged access and sampling infrastructure in place. In addition, even though Cori is a top 20 system that executes a wide array of open-science HPC applications, observations are affected by the set of applications and hardware configuration of the system. Therefore, HPC systems substantially different than Cori can use our analysis as a framework to repeat a similar study. In addition, sampling typically does not capture application executable files or input data sets. Therefore, reconstructing our analysis in a system simulator is impossible but also impractical due to the vast slowdown of a simulator for large-scale simulations compared to a real system. Similarly, sampling a production system has no method to differentiate when application demands exceed available resources as well as the resulting slowdown. For this reason and our 1s sampling period, our study focuses on sustained behavior and cannot make claims for the impact of resource disaggregation to application performance.

6.2 Hardware Implementation of Disaggregation

As we outline in Section 2.2, past work pursued a variety of hardware implementations and job schedulers for resource disaggregation. Our study quantitatively shows potential benefits of intra-rack disaggregation in a system similar to Cori to allow future research on hardware implementations to perform a more informed cost-benefit analysis. In addition, we study topics in the second half of Section 3 that assist in guiding important related topics such as how often a disaggregated system should be reconfigured. Therefore, future work can better decide whether to pursue full-system, intra-rack, or a hierarchical implementation of resource disaggregation and for which resources.

6.3 Future Applications and Systems

While it is hard to speculate how important HPC applications will evolve over the next decade, we have witnessed little change in HPC fundamental algorithms during the previous decade. It is those fundamentals that currently cause imbalance that motivates resource disaggregation. Another consideration is application resource demands relative to available resources in future systems. For instance, if future applications require significantly more memory than the memory available per CPU today, then this may motivate full system disaggregation of memory, especially if there is significant variability across applications. Similarly, if a subset of future applications request **non-volatile memory (NVM)**, then this may also motivate full system disaggregation of NVM, similar to how file system storage is disaggregated today.

However, future systems may have larger racks or nodes with more resources, strengthening the case for intra-rack resource disaggregation. When it comes to specialized fixed-function accelerators, a key question is how much data transfer they require and how many applications can use them. This can help determine which fixed-function accelerators should be disaggregated within racks, hierarchically, or across the system. Different resources can be disaggregated at different ranges. Ultimately, the choice should be made for each resource type for a given mix of applications, following an analysis similar to our study.

6.4 Future Work

Future work should explore the performance and cost tradeoff when allocating resources to applications whose utilization is dynamic. For instance, providing enough memory bandwidth to satisfy only the application's average demand is more likely to degrade the application's performance, but increases average resource utilization. Future work should also consider the impact of resource disaggregation to application performance, which should also consider the underlying hardware to implement resource disaggregation and the software stack. Job scheduling for heterogeneous HPC systems [27] should be aware of job resource usage and disaggregation hardware limitations. For instance, scheduling all nodes of an application in the same rack benefits locality but also increases the probability that all nodes will stress the same resource, thus hurting resource disaggregation.

7 CONCLUSION

In the approaching era of heterogeneous HPC systems, the variety of application demands creates a significant risk of underutilizing expensive compute and memory resources if we preserve today's rigid node configuration and reservation strategies. We demonstrate that key resources are consistently underutilized in NERSC's Cori and important ML applications executing on NVIDIA GPU. Based on our analysis, we show that unless we aggressively reduce resources inside a rack, resource disaggregation in an HPC system similar to Cori typically does not have to cross the rack boundary. In addition, with ideal intra-rack resource disaggregation, racks similar to Cori can reduce available memory and NIC resources by 5.36% to 69.01% and still satisfy the worst-case average rack utilization.

REFERENCES

[1] [n.d.]. About the Cray Urika-GX Hardware Guide H-6142. Retrieved from https://pubs.cray.com/bundle/Urika-GX_Hardware_Guide_H-6142_Rev_C_Urika-GX_HW_Guide_DITAval/page/Aries_High_Speed_Network_Urika-GX.html.

[2] [n.d.]. Characterization of the Cray Aries Network. Retrieved from https://www.nersc.gov/assets/pubs_presos/NUG2014Aries.pdf.

[3] [n.d.]. NERSC-10 Workload Analysis (Data from 2018). Retrieved from https://portal.nersc.gov/project/m888/nersc10/workload/N10_Workload_Analysis.latest.pdf.

[4] [n.d.]. NVIDIA DGX-1 User Guide. Retrieved from https://images.nvidia.com/content/technologies/deep-learning/pdf/DGX-1-UserGuide.pdf.

[5] 2021. NVML. Retrieved from https://developer.nvidia.com/nvidia-management-library-nvml.

[6] 2021. psutil. Retrieved from https://pypi.org/project/psutil/.

[7] Anthony Agelastos, Benjamin Allan, Jim Brandt, Paul Cassella, Jeremy Enos, Joshi Fullop, Ann Gentile, Steve Monk, Nichamon Naksinehaboon, Jeff Ogden, Mahesh Rajan, Michael Showerman, Joel Stevenson, Narate Taerat, and Tom Tucker. 2014. The lightweight distributed metric service: A scalable infrastructure for continuous monitoring of large scale computing systems and applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'14)*. IEEE Press, 154–165.

[8] Giovanni Agosta, William Fornaciari, Giuseppe Massari, Anna Pupykina, Federico Reghenzani, and Michele Zanella. 2018. Managing heterogeneous resources in HPC systems. In *Proceedings of the 9th Workshop and 7th Workshop on Parallel Programming and RunTime Management Techniques for Manycore Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms (PARMA-DITAM'18)*. ACM, 7–12.

[9] Bob Alverson, E. Froese, L. Kaplan, and D. Roweth. 2012. Cray XC series network. Retrieved from https://www.alcf.anl.gov/files/CrayXCNetwork.pdf.

[10] Marcelo Amaral, Jorda Polo, David Carrera, Nelson Gonzalez, Chih-Chieh Yang, Alessandro Morari, Bruce D'Amora, Alaa Youssef, and Malgorzata Steinder. 2021. DRMaestro: Orchestrating disaggregated resources on virtualized datacenters. *J. Cloud Comput.* 10 (Mar. 2021). https://doi.org/10.1186/s13677-021-00238-6

[11] H. Asaadi and B. Chapman. 2017. Comparative study of deep learning framework in HPC environments. In *Proceedings of the New York Scientific Data Summit (NYSDS'17)*. 1–7. https://doi.org/10.1109/NYSDS.2017.8085040

[12] Rahul Bera, Anant V. Nori, Onur Mutlu, and Sreenivas Subramoney. 2019. DSPatch. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (Oct. 2019). https://doi.org/10.1145/3352460.3358325

[13] M. Bielski, C. Pinto, D. Raho, and R. Pacalet. 2016. Survey on memory and devices disaggregation solutions for HPC systems. In *Proceedings of the IEEE International Conference on Computational Science and Engineering (CSE'16) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC'16) and 15th International Symposium on Distributed Computing and Applications for Business Engineering (DCABES'16)*. 197–204.

[14] Frank Block and Andreas Dewald. 2017. Linux memory forensics: Dissecting the user space process heap. *Dig. Investig.* 22 (2017), S66–S75.

[15] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, 1877–1901. Retrieved from https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[16] S. Browne, J. Dongarra, N. Garner, K. London, and P. Mucci. 2000. A scalable cross-platform infrastructure for application performance tuning using hardware counters. In *Proceedings of the ACM/IEEE Conference on Supercomputing (SC'00)*.

[17] Aaron Call, Jordà Polo, David Carrera, Francesc Guim, and Sujoy Sen. 2020. Disaggregating non-volatile memory for throughput-oriented genomics workloads. Retrieved from https://arxiv.org/abs/2007.02813.

[18] Chen Ding and K. Kennedy. 2000. The memory of bandwidth bottleneck and its amelioration by a compiler. In *Proceedings 14th International Parallel and Distributed Processing Symposium (IPDPS'00)*. 181–189. https://doi.org/10.1109/IPDPS.2000.845980

[19] Qixiang Cheng, Keren Bergman, Yishen Huang, Hao Yang, Meisam Bahadori, Nathan Abrams, Xiang Meng, Madeleine Glick, Yang Liu, and Michael Hochberg. 2019. Silicon photonic switch topologies and routing strategies for disaggregated data centers. *IEEE J. Select. Top. Quant. Electron.* 26 (Dec. 2019), 1–10. https://doi.org/10.1109/JSTQE.2019.2960950

[20] Qixiang Cheng, Sébastien Rumley, Meisam Bahadori, and Keren Bergman. 2018. Photonic switching in high performance datacenters (invited). *Opt. Express* 26, 12 (Jun 2018), 16022–16043. https://doi.org/10.1364/OE.26.016022

[21] Y. Cheng, M. De Andrade, L. Wosinska, and J. Chen. 2018. Resource disaggregation versus integrated servers in data centers: Impact of internal transmission capacity limitation. In *Proceedings of the European Conference on Optical Communication (ECOC'18)*. 1–3. https://doi.org/10.1109/ECOC.2018.8535214

[22] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. 2021. NVIDIA A100 tensor core GPU: Performance and innovation. *IEEE Micro* 41 (2021), 29–35.

[23] Cray. 2018. Aries Hardware Counters. Retrieved from https://pubs.cray.com/bundle/Aries_Hardware_Counters_S-0045-40/page/NIC_Performance_Counters.html.

[24] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. Retrieved from https://arXiv:1810.04805.

[25] Maria Dimakopoulou, Stéphane Eranian, Nectarios Koziris, and Nicholas Bambos. 2016. Reliable and efficient performance monitoring in linux. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'16)*. IEEE Press, Article 34, 13 pages.

[26] Douglas Doerfler, Brian Austin, Brandon Cook, Jack Deslippe, Krishna Kandalla, and Peter Mendygral. 2017. Evaluating the networking characteristics of the Cray XC-40 intel knights landing-based Cori supercomputer at NERSC. *Concurr. Comput.: Pract. Exper.* 30 (Sep. 2017).

[27] G. Domeniconi, E. Lee, Vanamala Venkataswamy, and Swaroopa Dola. 2019. CuSH: Cognitive scheduler for heterogeneous high performance computing system. https://www.cse.msu.edu/~zhaoxi35/DRL4KDD/10.pdf.

[28] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. In *Proceedings of the International Conference on Learning Representations*.

[29] S. Dublish, V. Nagarajan, and N. Topham. 2017. Evaluating and mitigating bandwidth bottlenecks across the memory hierarchy in GPUs. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'17)*. 239–248. https://doi.org/10.1109/ISPASS.2017.7975295

[30] Yuping Fan, Zhiling Lan, Paul Rich, William E. Allcock, Michael E. Papka, Brian Austin, and David Paul. 2019. Scheduling beyond CPUs for HPC. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*.

[31] Peter X. Gao, Akshay Narayan, Sagar Karandikar, Joao Carreira, Sangjin Han, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. 2016. Network requirements for resource disaggregation. In *Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation*. 249–264. https://dl.acm.org/doi/10.5555/3026877.302689

[32] Roberto Gioiosa, Ozcelik Burcu Mutlu, Seyong Lee, S. Jeffrey Vetter, Giulio Picierro, and Marco Cesati. 2020. The minos computing library—Efficient parallel programming for extremely heterogeneous systems. In *Proceedings of the Annual Workshop on General Purpose Processing Using Graphics Processing Unit and the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (GPGPU@PPoPP'20)*. 1–10.

[33] J. Gonzalez, A. Gazman, M. Hattink, M. G. Palma, M. Bahadori, R. Rubio-Noriega, L. Orosa, M. Glick, O. Mutlu, K. Bergman, and R. Azevedo. 2020. Optically connected memory for disaggregated data centers. In *Proceedings of the IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'20)*. 43–50. https://doi.org/10.1109/SBAC-PAD49847.2020.00017

[34] A. Guleria, J. Lakshmi, and C. Padala. 2019. EMF: Disaggregated GPUs in datacenters for efficiency, modularity and flexibility. In *Proceedings of the IEEE International Conference on Cloud Computing in Emerging Markets (CCEM'19)*. 1–8. https://doi.org/10.1109/CCEM48484.2019.000-5

[35] A. Guleria, J. Lakshmi, and C. Padala. 2019. QuADD: QUantifying accelerator disaggregated datacenter efficiency. In *Proceedings of the IEEE 12th International Conference on Cloud Computing (CLOUD'19)*. 349–357. https://doi.org/10.1109/CLOUD.2019.00064

[36] Jing Guo, Zihao Chang, Sa Wang, Haiyang Ding, Yihui Feng, Liang Mao, and Yungang Bao. 2019. Who limits the resource efficiency of my datacenter: An analysis of alibaba datacenter traces. In *Proceedings of the International Symposium on Quality of Service (IWQoS'19)*. ACM, Article 39, 10 pages. https://doi.org/10.1145/3326285.3329074

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[38] Yun He, Brandon Cook, Jack Deslippe, Brian Friesen, Richard Gerber, Rebecca Hartman-Baker, Alice Koniges, Thorsten Kurth, Stephen Leak, Woo-Sun Yang, Zhengji Zhao, Eddie Baron, and Peter Hauschildt. 2018. Preparing NERSC users for Cori, a Cray XC40 system with Intel many integrated cores. *Concurr. Comput.: Pract. Exper.* 30, 1 (2018), e4291. https://doi.org/10.1002/cpe.4291 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.4291 e4291 CPE-17-0254.

[39] Sepp Hochreiter. 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Int. J. Uncert., Fuzz. Knowl.-Based Syst.* 6, 02 (1998), 107–116.

[40] Tom Hogervorst, Tong Dong Qiu, Giacomo Marchiori, Alf Birger, Markus Blatt, and Razvan Nane. 2021. Hardware Acceleration of HPC Computational Flow Dynamics using HBM-enabled FPGAs. Retrieved from https://arXiv:2101.01745.

[41] Zaeem Hussain. 2020. Heterogeneity Aware Fault Tolerance for Extreme Scale Computing. (August 2020). Retrieved from http://d-scholarship.pitt.edu/39456/.

[42] W. Hwu, L. Chang, H. Kim, A. Dakkak, and I. El Hajj. 2015. Transitioning HPC software to exascale heterogeneous computing. In *Proceedings of the Computational Electromagnetics International Workshop (CEM'15)*. 1–2. https://doi.org/10.1109/CEM.2015.7237412

[43] Intel Corporation 2016. *Intel 64 and IA-32 Architectures Optimization Reference Manual*. Intel Corporation.

[44] P. Jamieson, A. Sanaullah, and M. Herbordt. 2018. Benchmarking heterogeneous HPC systems including reconfigurable fabrics: Community aspirations for ideal comparisons. In *Proceedings of the IEEE High Performance extreme Computing Conference (HPEC'18)*. 1–6.

[45] W. Jang. 2019. Unaligned burst-aware memory subsystem. *IEEE Trans. Very Large Scale Integr. Syst.* 27, 10 (2019), 2387–2400.

[46] Morris A. Jette, Andy B. Yoo, and Mark Grondona. 2002. SLURM: Simple linux utility for resource management. In *Proceedings of the Job Scheduling Strategies for Parallel Processing (JSSPP'03)*. Springer-Verlag, 44–60.

[47] George Michelogiannakis Sebastien Rumley Larry Dennison Monia Ghobadi Keren Bergman, John Shalf. 2018. PINE: An energy efficient flexibly interconnected photonic data center architecture for extreme scalability. In *Proceedings of the IEEE Optical Interconnects Conference (OI'18)*.

[48] H. Khaleghzadeh, R. R. Manumachu, and A. Lastovetsky. 2020. A hierarchical data-partitioning algorithm for performance optimization of data-parallel applications on heterogeneous multi-accelerator NUMA nodes. *IEEE Access* 8 (2020), 7861–7876.

[49] H. Khetawat, C. Zimmer, F. Mueller, S. Atchley, S. S. Vazhkudai, and M. Mubarak. 2019. Evaluating burst buffer placement in HPC systems. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'19)*. 1–11. https://doi.org/10.1109/CLUSTER.2019.8891051

[50] K. Koh, K. Kim, S. Jeon, and J. Huh. 2019. Disaggregated cloud memory with elastic block management. *IEEE Trans. Comput.* 68, 1 (2019), 39–52.

[51] V. R. Kommareddy, C. Hughes, S. Hammond, and A. Awad. 2019. Investigating fairness in disaggregated non-volatile memories. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'19)*. 104–110. https://doi.org/10.1109/ISVLSI.2019.00028

[52] Alexey Lastovetsky. 2015. Heterogeneous parallel computing: From clusters of workstations to hierarchical hybrid platforms. *Supercomput. Front. Innov.* 1, 3 (2015). Retrieved from https://superfri.org/superfri/article/view/32.

[53] Teng Li, Vikram K. Narayana, Esam El-Araby, and Tarek El-Ghazawi. 2011. GPU resource sharing and virtualization on high performance computing systems. In *Proceedings of the International Conference on Parallel Processing*. 733–742. https://doi.org/10.1109/ICPP.2011.88

[54] Teng Li, Vikram K. Narayana, and Tarek A. El-Ghazawi. 2015. Efficient resource sharing through GPU virtualization on accelerated high performance computing systems. Retrieved from http://arxiv.org/abs/1511.07658.

[55] R. Lin, Y. Cheng, M. D. Andrade, L. Wosinska, and J. Chen. 2020. Disaggregated data centers: Challenges and trade-offs. *IEEE Commun. Mag.* 58, 2 (2020), 20–26. https://doi.org/10.1109/MCOM.001.1900612

[56] B. Liu, D. Zydek, H. Selvaraj, and L. Gewali. 2012. Accelerating high performance computing applications: Using CPUs, GPUs, hybrid CPU/GPU, and FPGAs. In *Proceedings of the 13th International Conference on Parallel and Distributed Computing, Applications, and Technologies*. 337–342.

[57] Arthur B. Maccabe. 2017. Operating and runtime systems challenges for HPC systems. In *Proceedings of the 7th International Workshop on Runtime and Operating Systems for Supercomputers (ROSS'17)*. ACM, Article 1, 1 pages. https://doi.org/10.1145/3095770.3095771

[58] D. M. Marom, P. D. Colbourne, A. D'errico, N. K. Fontaine, Y. Ikuma, R. Proietti, L. Zong, J. M. Rivas-Moscoso, and I. Tomkos. 2017. Survey of photonic switching architectures and technologies in support of spatially and spectrally flexible optical networking [invited]. *IEEE/OSA J. Optic. Commun. Netw.* 9, 1 (2017), 1–26. https://doi.org/10.1364/JOCN.9.000001

[59] P. Mattson, V. J. Reddi, C. Cheng, C. Coleman, G. Diamos, D. Kanter, P. Micikevicius, D. Patterson, G. Schmuelling, H. Tang, G. Wei, and C. Wu. 2020. MLPerf: An industry standard benchmark suite for machine learning performance. *IEEE Micro* 40, 2 (2020), 8–16.

[60] George Michelogiannakis, Yiwen Shen, Min Yee Teh, Xiang Meng, Benjamin Aivazi, Taylor Groves, John Shalf, Madeleine Glick, Manya Ghobadi, Larry Dennison, and Keren Bergman. 2019. Bandwidth steering in HPC using silicon nanophotonics. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'19)*. ACM, Article 41, 25 pages. https://doi.org/10.1145/3295500.3356145

[61] D. Milojicic. 2020. Accelerators for artificial intelligence and high-performance computing. *Computer* 53, 2 (2020), 14–22.

[62] Sparsh Mittal and Jeffrey S. Vetter. 2015. A survey of CPU-GPU heterogeneous computing techniques. *ACM Comput. Surv.* 47, 4, Article 69 (July 2015), 35 pages. https://doi.org/10.1145/2788396

[63] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G. Azzolini et al. 2019. Deep learning recommendation model for personalization and recommendation systems. Retrieved from https://arXiv:1906.00091.

[64] A. D. Papaioannou, R. Nejabati, and D. Simeonidou. 2016. The benefits of a disaggregated data centre: A resource allocation approach. In *Proceedings of the IEEE Global Communications Conference (GLOBECOM'16)*. 1–7. https://doi.org/10.1109/GLOCOM.2016.7842314

[65] Scott Parker, Sudheer Chunduri, K. Harms, and K. Kandalla. 2018. Performance evaluation of MPI on cray XC 40 xeon phi systems. https://cug.org/proceedings/cug2018_proceedings/includes/files/pap131s2-file1.pdf.

[66] J. Patel, V. Jindal, I. Yen, F. Bastani, J. Xu, and P. Garraghan. 2015. Workload estimation for improving resource management decisions in the cloud. In *Proceedings of the IEEE 12th International Symposium on Autonomous Decentralized Systems*. 25–32. https://doi.org/10.1109/ISADS.2015.17

[67] I. Peng, R. Pearce, and M. Gokhale. 2020. On the memory underutilization: Exploring disaggregated memory on HPC systems. In *Proceedings of the IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'20)*. 183–190.

[68] G. Ramirez-Gargallo, M. Garcia-Gasulla, and F. Mantovani. 2019. TensorFlow on state-of-the-art HPC clusters: A machine learning use case. In *Proceedings of the 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID'19)*. 526–533. https://doi.org/10.1109/CCGRID.2019.00067

[69] G. P. Rodrigo, P. Östberg, E. Elmroth, K. Antypas, R. Gerber, and L. Ramakrishnan. 2016. Towards understanding job heterogeneity in HPC: A NERSC case study. In *Proceedings of the 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'16)*. 521–526.

[70] O. Segal, N. Nasiri, M. Margala, and W. Vanderbauwhede. 2014. High-level programming of FPGAs for HPC and data centric applications. In *Proceedings of the IEEE High Performance Extreme Computing Conference (HPEC'14)*. 1–3. https://doi.org/10.1109/HPEC.2014.7040979

[71] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. 2018. LegoOS: A disseminated, distributed OS for hardware resource disaggregation. In *Proceedings of the 13th USENIX conference on Operating Systems Design and Implementation*. 69–87. https://dl.acm.org/doi/10.5555/3291168.3291175.

[72] Misha Smelyanskiy. 2019. Zion: Facebook next-generation large memory training platform. In *Proceedings of the IEEE Hot Chips 31 Symposium (HCS'19)*. IEEE Computer Society, 1–22.

[73] A. Sodani. 2015. Knights landing (KNL): 2nd Generation Intel Xeon Phi processor. In *Proceedings of the IEEE Hot Chips 27 Symposium (HCS'15)*. 1–24.

[74] K. Tang, D. Tiwari, S. Gupta, S. S. Vazhkudai, and X. He. 2017. Effective running of end-to-end HPC workflows on emerging heterogeneous architectures. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'17)*. 344–348. https://doi.org/10.1109/CLUSTER.2017.22

[75] J. Taylor. 2015. Facebook's data center infrastructure: Open compute, disaggregated rack, and beyond. In *Proceedings of the Optical Fiber Communications Conference and Exhibition (OFC'15)*. 1–1.

[76] D. Tiwari, S. Gupta, G. Gallarno, J. Rogers, and D. Maxwell. 2015. Reliability lessons learned from GPU experience with the Titan supercomputer at Oak Ridge leadership computing facility. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'15)*. 1–12. https://doi.org/10.1145/2807591.2807666

[77] D. Tiwari, S. Gupta, J. Rogers, D. Maxwell, P. Rech, S. Vazhkudai, D. Oliveira, D. Londo, N. DeBardeleben, P. Navaux, L. Carro, and A. Bland. 2015. Understanding GPU errors on large-scale HPC systems and the implications for system design and operation. In *Proceedings of the IEEE 21st International Symposium on High Performance Computer Architecture (HPCA'15)*. 331–342. https://doi.org/10.1109/HPCA.2015.7056044

[78] M. Ujaldón. 2016. HPC accelerators with 3D memory. In *Proceedings of the IEEE International Conference on Computational Science and Engineering (CSE'16), the IEEE International Conference on Embedded and Ubiquitous Computing (EUC'16), and the 15th International Symposium on Distributed Computing and Applications for Business Engineering (DCABES'16)*. 320–328. https://doi.org/10.1109/CSE-EUC-DCABES.2016.203

[79] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. Retrieved from https://arXiv:1706.03762.

[80] M. G. Venkata, F. Aderholdt, and Z. Parchman. 2017. SharP: Towards programming extreme-scale systems with hierarchical heterogeneous memory. In *Proceedings of the 46th International Conference on Parallel Processing Workshops (ICPPW'17)*. 145–154. https://doi.org/10.1109/ICPPW.2017.32

[81] Chao Wang, Wenqi Lou, Lei Gong, Lihui Jin, Luchao Tan, Yahui Hu, Xi Li, and Xuehai Zhou. 2017. Reconfigurable Hardware Accelerators: Opportunities, Trends, and Challenges. Retrieved from https://arXiv:1712.04771.

[82] Y. Wu, G. Min, K. Li, and B. Javadi. 2009. Performance analysis of communication networks in multi-cluster systems under bursty traffic with communication locality. In *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM'09)*. 1–6. https://doi.org/10.1109/GLOCOM.2009.5425416

[83] G. Zervas, H. Yuan, A. Saljoghei, Q. Chen, and V. Mishra. 2018. Optically disaggregated data centers with minimal remote memory latency: Technologies, architectures, and resource allocation [Invited]. *IEEE/OSA J. Optic. Commun. Netw.* 10, 2 (2018), A270–A285. https://doi.org/10.1364/JOCN.10.00A270

[84] Hongzhong Zheng, Jiang Lin, Zhao Zhang, and Zhichun Zhu. 2009. Decoupled DIMM: Building high-bandwidth memory system using low-speed DRAM devices. *SIGARCH Comput. Archit. News* 37, 3 (June 2009), 255–266. https://doi.org/10.1145/1555815.1555788

[85] Darko Zivanovic, Milan Pavlovic, Milan Radulovic, Hyunsung Shin, Jongpil Son, Sally A. Mckee, Paul M. Carpenter, Petar Radojković, and Eduard Ayguadé. 2017. Main memory in HPC: Do we need more or could we live with less? *ACM Trans. Archit. Code Optim.* 14, 1, Article 3 (Mar. 2017).

Received July 2021; revised October 2021; accepted January 2022