A Methodology for Evaluating Tightly-integrated and Disaggregated Accelerated Architectures

Taylor Groves, Chris Daley, Rahulkumar Gayatri, Hai Ah Nam, Nan Ding,
Lenny Oliker, Nicholas J. Wright, Samuel Williams
Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA
{tgroves, csdaley, rgayatri, hnam, nanding, loliker, njwright, swwilliams}@lbl.gov

***Abstract*—Tighter integration of computational resources can foster superior application performance by mitigating communication bottlenecks. Unfortunately, not every application can use every compute or accelerator all the time. As a result, co-locating resources often leads to under-utilization of resources. To mitigate this challenge, architects have proposed disaggregation and ad hoc pooling of computational resources. In the next five years, HPC system architects will be presented with a spectrum of accelerated solutions ranging from tightly coupled, single package APUs to a sea of disaggregated GPUs interconnected by a global network. In this paper, we detail NEthing, our methodology and tool for evaluating the potential performance implications of such diverse architectural paradigms. We demonstrate our methodology on today's and projected 2026 technologies for three distinct workloads: a compute-intensive kernel, a tightly-coupled HPC simulation, and an ensemble of loosely-coupled HPC simulations. Our results leverage NEthing to quantify the increased utilization disaggregated systems must achieve in order to match superior performance of APUs and on-board GPUs.**

***Index Terms*—Modeling of computer architecture, Performance evaluation, On-chip interconnection networks, Super (very large) computers, Cost/performance**

I. Introduction

GPU accelerators are now commonplace in all of the largest U.S. and European HPC systems, and a variety of options are emerging regarding the degree of CPU-GPU coupling. Determining which GPU accelerator to leverage and how tightly coupled it should be to the host CPU is a huge challenge for system designers – as different design point make trade-offs between peak throughput, power, and CPU-to-GPU transfer speeds and monetary cost. There is a growing breadth of complex applications to analyze, and even when considering an individual application, there can be a spectrum of architectural preferences dependent on the problem configuration and input. This creates a need for more advanced models and tools to project the benefits of future hardware.

In this paper, we develop a methodology for analyzing the benefits and pitfalls between tighter and looser system-level integration of GPU accelerators. To that end, we begin with a review of technology architectural trends leading to a heterogeneous landscape. We then develop an intuition and a simple acceleration offload model. We show how it may be used to project the benefits of a modern GPU architecture. We then consider how the application landscape has increased in complexity and requires additional profiling and tools to analyze performance. To address these complexities we extend a set of modern profiling tools and introduce NEthing [1] (pronounced "anything") which provides additional capabilities to analyze complex application behavior, such as communication/computation overlap and parameterize models of data transfer. We then demonstrate the utility of NEthing via detailed analysis of the modern QCD application MILC [2]. Following this, for a range of kernels, we leverage NEthing's profiling to project the performance of four future architecture design points: APU (integrated CPUs and GPUs), on-board, tightly-coupled CPU/GPU, Discrete PCIe-attached GPU, and disaggregated GPU. Overall, our work presents a methodology to gain insight into the complex tradeoffs of application requirements and accelerator integration design.

This manuscript has been authored by an author at Lawrence Berkeley National Laboratory under Contract No. DE-AC02-05CH11231 with the U.S. Department of Energy. The U.S. Government retains, and the publisher, by accepting the article for publication, acknowledges, that the U.S. Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for U.S. Government purposes.

II. Background and Related Work

As system designers look for opportunities for greater performance in the future, specialization and heterogeneity have emerged as viable solutions. However, having greater choice in the selection of architectures does not benefit all applications equally as different architectures will focus on a subset of performance characteristics (e.g. how tightly coupled CPU and GPU are, memory capacity, memory bandwidth, degree of parallelism, etc). In this work, we examine four design points for future accelerators, namely, APU, On-board GPU, Discrete GPU and Disaggregated GPU. Each of these vary by how tightly coupled the CPU and GPU are and the number of memory domains.

APU: An Accelerated Processing Unit or APU combines a CPU and GPU onto a single die or package. Examples include the Apple M1 and AMD Fusion line [3], [4]. Whereas a typical GPU contains two separate physical memories, the APU has the benefit of a single physical memory. This eliminates transfers between host and device memories. In many ways this targets the applications that would not have performed well on traditional GPUs where frequent data transfers dominate GPU execution time.

On-board GPU: The on-board GPU is motivated by the past IBM's NVLink-connected Power9 processor [5]

and the more recent NVIDIA H100 [6]. In this work the distinguishing feature between the APU and On-board GPU is that the On-board GPU contains two separate memories, one for the CPU and one for the GPU. These two memory regions are connected by a interconnect that allows for bandwidth equal to the CPU memory bandwidth (∼450GBps unidirectional for H100). On-board favors workloads that move large volumes of data between the CPU and GPU and have demanding GPU computational requirements.

Discrete GPU: The discrete PCIe-attached GPU is what is currently in use in many HPC center and is used in our evaluation [7]. The GPU has a separate High Bandwidth Memory (HBM) for the GPU and data is staged from host memory of the CPU. The discrete GPU utilizes the relatively slow PCIe connection to perform data transfers between host and device. This results in bandwidths of 31.5GBps (Gen4 $\times 16$) and incurs a latency of on the order of a few microseconds per transfer.

Disaggregated: Statically configured nodes may have a sub-optimal ratio of CPUs to GPUs to NICs. This may leave a GPU or CPU resource underutilized, or stranded. By uncoupling GPUs from CPU resources we create opportunities for gains in overall system utilization as we can provide a flexible mapping of accelerator to CPU resources. For example this might look like a set of GPU devices shared among CPU nodes over a data center network. This flexibility comes at the cost of higher latency and NIC limited bandwidth between the disaggregated CPUs and GPUs. This adds an additional 2us of latency, which is a reasonable approximation of inter-node latency (1.86us measured on the Perlmutter system).

A. Related work

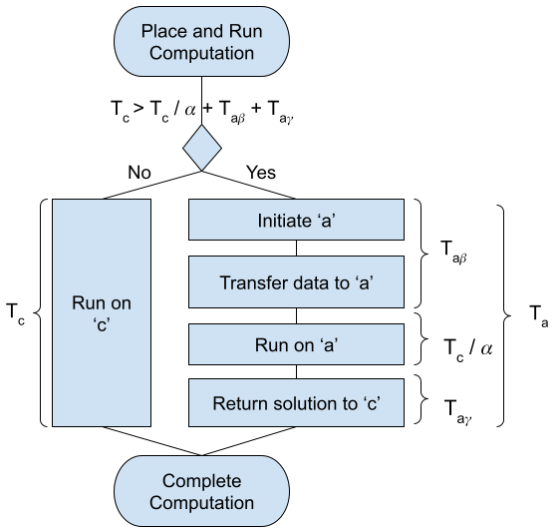
There is a growing body of work focused on the feasibility of disaggregating accelerators within a data center [8]–[12]. However, much of this work is focused on utilization and scheduling of a cluster or data center rather than modeling the impact of tightly coupled and disaggregated CPUs and GPUs.

There is a large body of work that has quantified the potential for offloading to a particular accelerator. In the past this has mostly been a binary decision whether to use a GPU accelerator or not. The increasing variety of accelerators and designs has complicated the problem of designing a HPC center. APIs that provide portable offload techniques such as OpenCL [13], OpenACC [14] and OneAPI [15] require architectural models to determine what code to offload.

More broadly offloading and scheduling Function as a Service (FaaS) [16]–[21] has been of interest for decades. In contrast, our focus is providing a model and tools to study the coupling between CPU and accelerator.

III. Concurrent Offload Model

The opportunity for applications to effectively leverage an accelerator is determined by comparing the benefit from the acceleration factor of the computational workload against the overheads of moving the data to the accelerator and launching the kernels. We visualize this as a flow chart in Fig. 1. Where



c	Traditional processor.
T_c	Time to solution on c.
a	Accelerated processor.
α	Factor of acceleration for execution on a relative to c.
$T_{a\beta}$	Time to launch kernel and transfer data.
T_c/α	Time to execute kernel after launch.
$T_{a\gamma}$	Time to return result from a to c.
T_a	Time to solution on a.

Fig. 1. Flowchart and definitions to evaluate the benefit of offloading computation to an accelerator versus running it on the host CPU.

T_c is the time to solution on a traditional host, such as a CPU, we can compare this against the sum of accelerator states (T_a) on the right hand side of the flow chart.

Such a model provides a simplified view of the benefits for traditional acceleration where data transfers occur outside of the accelerated kernel execution. This model has been valuable in representing a wide-range of early ports to accelerators, which historically incurred substantial initialization and data transfer costs. This required coarse-grained patterns of large data transfers followed by large kernels.

As shown in Fig. 1, we label the time to initiate and transfer data to the accelerator as $T_{a\beta}$ and then the time to return the solution to the host process as $T_{a\gamma}$. The factor of acceleration of the accelerator, compared to the host CPU is labeled as α. If the acceleration factor is sufficiently large, or the data transfer and initialization times are sufficiently small, offloading the the accelerator provides a net speedup:

$$T_{a\beta} + T_{a\gamma} + T_c/\alpha < T_c$$

so

$$T_{a\beta} + T_{a\gamma} < T_c\left(\frac{\alpha - 1}{\alpha}\right)$$

or

$$\frac{T_{a\beta} + T_{a\gamma}}{T_c} = \textit{offload:compute} < \frac{\alpha - 1}{\alpha}$$

Thus, the acceleration factor restricts the viable offload:compute ratio required for any speedup. For example, if $\alpha = 2\times$, then the unaccelerated compute time must be twice the

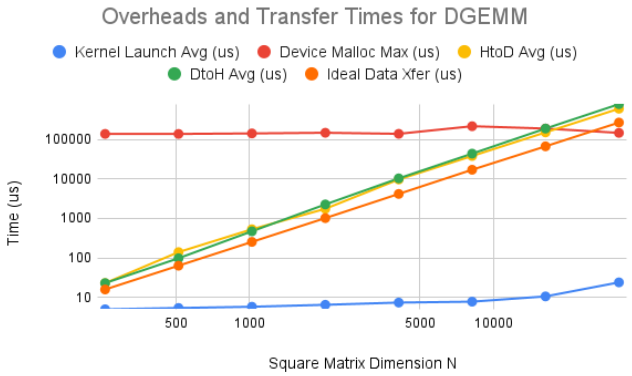


Fig. 2. Initiation and transfer overheads ($T_{a\beta}$ and $T_{a\gamma}$) of kernel launch, `malloc` times and PCIe transfers for DGEMM of varying size as measured on NERSC Perlmutter (NVIDIA A100 GPU).

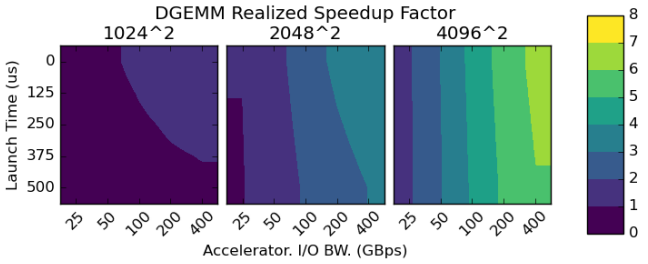


Fig. 3. Impact of launch time and bandwidth on DGEMM performance. Three problem sizes show the benefit of a GPU compared to a CPU for launch times and data transfer bandwidths. The computation times were collected on NVIDIA A100 GPU and the acceleration factor over the host CPU is $\alpha = 8$.

cost of offloading to break even. Conversely, if $\alpha = 5\times$, then the offload cost must be far less than 80% of unaccelerated compute time if acceleration is to provide any overall speedup.

A. An example using DGEMM

To provide some intuition, consider a double precision, matrix-matrix-multiply (DGEMM). While real applications are more complex, DGEMM provides an appropriate vehicle to explore basic concepts. The flow chart provides an accurate description of the launch behavior to determine when it is beneficial to move a calculation from the host-based CPU to an attached GPU. DGEMM takes two matrices each of size N^2 doubles as inputs and returns one matrix of size N^2 doubles performing $2N^3$ operations in the process. Understanding DGEMM's relationship between computation and problem size (i.e. data transfers) reinforces the intuition that the value of accelerators increases with progressively coarser grained offloading. To better illustrate the real overheads associated with $T_{a\beta}$ and $T_{a\gamma}$ we profile DGEMM execution overheads on an NVIDIA A100 [22] GPU using Nsight Systems.

Fig. 2 plots (1) observed kernel launch time, (2) maximum observed time to allocate memory on the GPU, and (3) host-to-device (HtoD) and device-to-host (DtoH) data transfers as a function of matrix dimension (N) spanning a range of matrix footprints from as small as 500KB to as large as 8.3GB. As a comparison, it also shows the theoretical time-per-transfer on the PCIe network (the reciprocal of bandwidth, also known as the gap in the LogP [23] model). Observe that HtoD and DtoH transfer times are somewhat higher than the theoretical lower limit of the PCIe network. This is because of protocol and other overheads that prevent the full utilization of the raw PCIe pin bandwidth. Device memory allocation and kernel launch time show little sensitivity to problem size. Although device `malloc` time is very large, in many real applications, the cost can be amortized across multiple kernel invocations.

The simplicity of DGEMM provides an easy way to estimate the acceleration factor α. For modern hardware we can calculate the peak fused-multiply-addition (FMA) operation rate that is the foundation of the DGEMM calculation. For example, we assume a modern CPU has a double-precision throughput of 2.5 TFLOP/s (CPU frequency $\times$ core count $\times$ number of FMA units $\times$ number of FMAs per cycle $\times$ the vector width divided by the precision). A similar calculation, which leverages tensor operations, is provided by GPU technical specifications [24] and shows that the A100 GPU provides 20 TFLOP/s of double-precision performance. This would imply the GPU has an α of $8\times$ the 64-core host CPU in this DGEMM example.

With all the parameters of the model accounted for we can explore how changes in architecture design that increase bandwidth between the host CPU and accelerator or reduce the initialization and launch cost will benefit a given workload. Fig. 3 shows the performance benefits of increasing the data transfer bandwidths or reducing the launch cost for the NVIDIA A100 compared to a host CPU with $\alpha = 8$. We show the results for three different problem sizes — (a) 1024^2, (b) 2048^2, (c) 4096^2 — with the time spent in computation measured on the A100 GPUs and the data transfer and launch times modeled for a range of improved bandwidth and launch overheads. Launch times are modeled between 0 and 500μs. The model shows how improving bandwidth particularly benefits DGEMMs larger than 4096^2 compared to 1024^2. Even with massive increases to bandwidth and a launch overhead of zero, the best case speedup for a 1024^2 DGEMM is less than $2\times$ due to the small amount of work relative to the latency-bound data transfers. Smaller matrices (e.g. 1024^2) could be a potential candidate for execution on an APU, as APUs have only a single memory space and therefore do not incur data transfer costs. While the tight coupling of an APU's GPU and CPU components is beneficial, it cannot match a discrete GPU's raw computational density. Conversely, larger problems are more sensitive to increases to bandwidth, rather than changes in launch costs. For multiplications in the range of 4096^2, attainable speedup nears $7\times$ as bandwidth increases to 400GBps. Such large problems see little sensitivity to increased launch costs.

To reiterate, we use DGEMM to develop modeling intuition. Modern full-featured applications may (1) leverage a variety of algorithms that make estimating α difficult, (2) utilize multiple tiers of networks, (3) exhibit varying transfer

performance dependent on message size and overheads, and (4) perform multiple iterations of potentially overlapping kernel launch, memory allocation and data transfers. This requires a more sophisticated set of tools to analyze, which we address in the next section.

IV. Capturing Complex Application Behavior

In order to understand real applications, one must extract greater detail than analytical modeling could provide. For example, in comparison to the simple single DGEMM kernel, a real application may be comprised of tens of different kernels executed thousands of times. Each of these kernels may leverage multiple data transfers of varying size. To extract these details we utilize and augment NVIDIA's Nsight Systems profiler [25]. Nsight Systems provides detailed analysis of application performance including data transfer sizes and timings, kernel performance, CPU activity and memory utilization. While Nsight provides a set of default analysis and views, it also exports the profiles to a database format that can be imported by other analysis frameworks and tools. We extract this data and port it into our Python-based NEthing [dbl blind] framework to provide the additional analysis needed to meet the objectives of this paper.

Our work addresses several challenges that provide the necessary detail to understand the complex applications running on HPC systems today. Our framework automates the parameter generation for a LogGP-based model, outputs summary statistics and generates figures to summarize the overall application. Lastly, our framework allows the user to export the generated time series which can be analyzed in common frameworks such as Jupyter, Python and Pandas.

A. Challenges

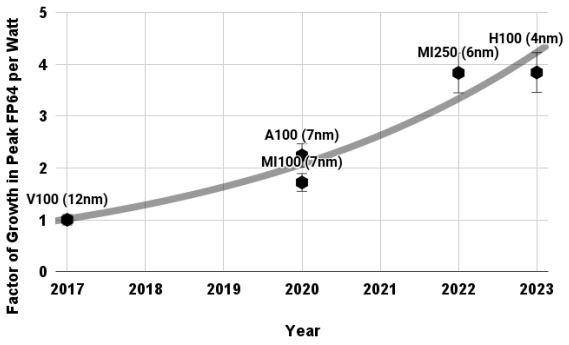


Fig. 4. Peak FP64 GPU performance per watt (relative to V100) over the last 5 years as specified by the manufacturer (AMD and NVIDIA).

In the previous section, we mention several ways that real applications may increase the complexity of analysis. In this section, we describe the challenges of generating parameters for our accelerator offload model, and how we use collected profiles to summarize important application behavior (tiers of network, size- and overhead-dependent transfer performance, computation and data-transfer overlap).

Computation: Detailed predictions of computational benefits of a future architecture across a broad workload requires detailed simulation and analysis that is beyond the scope of this paper. As such, we make the assumption that future hardware accelerators are manufactured under similar process technologies, and exhibit similar performance per watt. As far as general purpose GPUs are concerned, we assume that future GPUs and APUs will maintain the FLOPS per watt growth of the last five years as shown in Fig 4, which is doubling every three years. This is a middle of the road projection with others, like TSMC, suggesting a continued doubling every two years [26]. We use these trends to represent the portions of computation as blocks of time that may be accelerated by α and we derive α from historical data. While we know, some architectures such as recent AI accelerators strike a different balance between supported precision, memory capacity, bandwidth, and chip area, we leave this for future work.

Multi-tier networks: In a typical scale-out application this creates multiple tiers of data transfers happening between (1) the host CPU and GPU (e.g. PCIe), (2) device-to-device over an intra-node network (e.g. NVlink [27] or Infinity Fabric [28]) and (3) device-to-device over an inter-node network (e.g. Ethernet, Slingshot [29] or Infiniband [30]). To account for this, we measure and model the performance of the network hierarchy for both PCIe and NVLink. To further complicate matters, both host bound and NIC bound PCIe traffic are indistinguishable from each other when examining logs of PCIe transfers. The version of Nsight Systems used in this work provides "experimental" functionality to record NIC traffic, however we were unable to make this work in our experiments. While we would like to activate it in future versions, for the purposes of this work we differentiate between NIC-bound and host-bound PCIe traffic by executing multiple configurations of process mappings (Fig. 5) and comparing the PCIe traffic of each. The first configuration is confined to a single node with 4 GPUs per node (left-side of Fig 5). In this case, there is no PCIe traffic destined for the NIC (i.e. only host-bound). In the second configuration (right-side of Fig 5) we use four nodes with one GPU per node. Transfers that were taking place over NVLink must now cross the HPC network. This adds additional PCIe traffic. We can estimate the NIC-bound PCIe traffic by subtracting the sum of PCIe data transferred in the 4-node configuration from the sum of PCIe transfers in the 1-node configuration.

Beyond LogP modeling: The LogP model was created to break down communication costs into latency, overhead, gap (the time to load the entire message onto the wire) and processing. In the original model, l, o, and g are all fixed cost, independent of message size. Later enhancements determined that g should be represented as a cost per byte transferred [31] and other work suggest that pipelining and batching operations require further analysis to determine the cost per byte as a function of message size [32]. For the kernels explored in this work, we found up to a 13% improvement in model accuracy by accounting for message size.

We automate the creation of LogGP-based models for

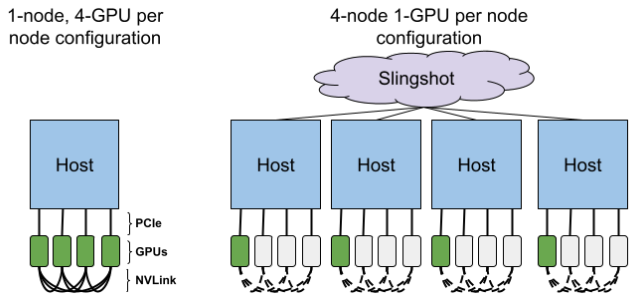


Fig. 5. Two configurations used to explore the shifts in data transfer characteristics as we compare host-bound vs. remote bound transfers. In the 4-node configuration, additional PCIe traffic occurs that are bound for remote GPUs.

communication performance at each tier of the network for the applications profiled. This allows users to estimate the network efficiency and overheads specific to the frequency and message sizes of each application. We can then apply the efficiency observed for modeling the performance impact of future networks.

Overlap of Computation and Data Transfers: General purpose GPU accelerators have typically been loosely attached to CPUs and incur substantial (order ten microseconds) penalties for launching kernels, performing data transfers and allocating device memory. To hide these costs, many real applications overlap computation and data transfers. On GPUs this can be done by utilizing multiple streams which each synchronize independently of each other and allowing for asynchronous data transfer between host and device. For the applications explored in this work we saw kernel/data transfer overlap occurring in up to 40% of transfers. If an application has substantial overlap, the benefit of increasing the processing power or the interconnect bandwidth will be less noticeable to overall application run times. A contribution of NEthing is that it consolidates computation and data transfer events into a single timeline before calculating the overlap between the time-series. This allows users to differentiate the data transfers that overlap with computation (and would not necessarily see the benefit of increased bandwidth) from the non-overlapping transfers that are performance critical.

Run time vs. kernel windows: Often, an application or benchmark will need to set up the problem, perform I/O, broadcast parameters to the set of worker nodes, and verify the solution. It's not always clear whether these non-trivial setup and tear-down phases are representative of a production application, which may have been shortened to be tractable for analysis and repeated execution. For this reason, NEthing reports data in terms of the *run time*, that is the total execution window, and the *kernel window*, which we define as the time between the first kernel's start and last kernel's completion.

Kernel launch times: For most applications, kernel launch times are overshadowed by data transfer times (see Fig. 2). Although the Nsight profiler captures kernel launch times and we can determine the amount of time spent performing initiating kernels, none of the architectures we explore fully eliminate these kernel launch times. For an APU, the greatest benefit comes from eliminating the host to device memory copy. Additional work may be done to reduce kernel launch latency by overlapping the kernels or pre-instantiating kernel work descriptors. In this work, we focus on the impact of varying the interconnect between the host CPU and GPU, rather than exploring improvements for kernel launch times.

Memory allocation times: In Fig. 2, we observed that device `malloc` has some of the highest overheads when considering initialization costs. Due to the high overheads for allocating device memory, many applications have been optimized to allocate memory only when absolutely necessary and reuse allocated buffers for multiple kernels. This generally means that memory allocation costs may be amortized throughout the execution of an application. For the applications we explored, we only saw a handful of memory allocations occur throughout their executions. NEthing includes a timeline of memory allocations so a user may determine whether or not this is the case for their program.

V. NEthing Demonstration

In order to demonstrate the capabilities of our analysis framework, we perform a small set of profiling experiments for a sophisticated, GPU accelerated application, MILC. In the subsequent section, these experiments serve as a baseline from which we can demonstrate the benefits of future architectural enhancements, such as improvements to bandwidth, or decreasing initiation costs.

A. Baseline Architecture

We perform our experiments on the Perlmutter system [7] at NERSC. Each node has a single AMD 7763 64-core CPU and four NVIDIA A100 GPUs. There is 256 GB of DDR4 DRAM for each CPU and each GPU contains 40 GB of HBM2. Each GPU is connected to every other GPU within the node by 100GBps (per-direction) of NVLink-3, resulting in 300GBps of incoming and 300GBps outgoing bandwidth for a single GPU. The GPUs and CPUs are connected by a PCIe-4×16 connection (31.5GBps). Two Mellanox NICs are attached to each node providing 25GBps aggregate bandwidth.

B. Applications and Configuration

MILC: To demonstrate our framework, we show the detailed results for a sophisticated GPU accelerated application, MILC [2]. MILC is a collaborative code for lattice QCD computations. The software is written in C and C++ and performs SU(3) lattice gauge theory. We used the QUDA backend of MILC developed by NVIDIA for their corresponding GPUs [33]. We ran the MILC generation problem that runs the *su3_rhmd_hisq* executable. The output of the MILC generation run is the time taken for three solvers: Conjugate Gradient (CG), Fermion Force (FF) and Gauge Force (GF). All three solvers are extremely sensitive to the placement of tasks due to the inter-task communication. Our four MILC problem sizes use lattices of $32^3 \times 32$, $32^3 \times 64$, $32^3 \times 96$ and $36^3 \times 72$.

LAMMPS: We use the molecular dynamics software package LAMMPS [34], which provides a wide range of

potential implementations to simulate interactions between atoms. For this paper, we select the computationally intensive Spectral Neighborhood Analysis Potential (SNAP) [34]. The SNAP implementation computes the force on each atom based on its neighbors and a set of bispectrum components that describe the local neighborhood. The set of atoms is divided among the available tasks. Each task is assigned a single GPU. Unlike MILC, LAMMPS-SNAP has far less communication between tasks — for large atom sets, PCI transfer times are less than 1% of the total execution time. In essence, this version of LAMMPS is a loosely-coupled ensemble of tasks. As such, we only observe the degree to which a GPU communicates with its host processor.

Due to space constraints, we demonstrate NEthing's visualization capabilities using only MILC which shows more varied behavior, but include LAMMPS in the demonstration of NEthing's analysis capabilities in Sec VI.

C. Differentiating Data Movement

In order to establish the relationship between growth in computation, launch costs, and data transfer requirements of the application, we increase the scale of the problem size while holding the number of GPUs constant. Each of these runs are executed in two process-mapping configurations as shown in Fig. 5: four GPUs on a single node (1-node) or one GPU per node across four nodes (4-node). In other words, both process mappings always use four GPUs in aggregate, but the node count changes between one and four. In both configurations, we allocate 32 hardware threads on the host CPU per GPU.

Running in the 4-node configuration creates the additional challenge of having to distinguish between PCIe traffic bound for the local host versus PCIe traffic that is passing through the local CPU before continuing over the network to a remote host. The reason it is important to distinguish between these types of traffic is because future architectures such as an on-board GPU or APU may have better bandwidth between the host CPU and GPU, but the network will remain limited by PCIe performance across all architectures. Normally this would be an easy problem to solve by collecting data from the NIC, but we were unable to get this working during the evaluation of this work. In the future, these counters will be added to our framework.

By running on a single node with four GPUs connected over NVLink, we shift transfers that would occur over the NIC to NVLink. This allows users to determine the data transfer characteristics for host-bound PCIe traffic. Complementing the single node run, the 4-node configuration shifts traffic that would move over NVLink to move over the HPC Slingshot Network instead. We can compare the data transfers of both configurations to see what additional traffic goes over the network and whether it has different properties than host-bound traffic, such as computation and data transfer overlap.

D. Results

Radar Plot: NEthing radar plots (Figs. 6-7) summarize the different aspects of MILC application performance for four different problem sizes. For each labeled spoke (performance component), the radial distance of the colored region indicates the percentage of time that component contributes to the kernel window (first kernel launch to last kernel's completion). Note that the sum of values across dimensions may add up to greater than 100%. This is due to the fact that individual components may overlap with each other (e.g. CPU running while the GPU is running, while data transfers are taking place).

In Fig. 6, each of the four problem sizes for the 1-node configuration are overlaid onto each other. This allows users to

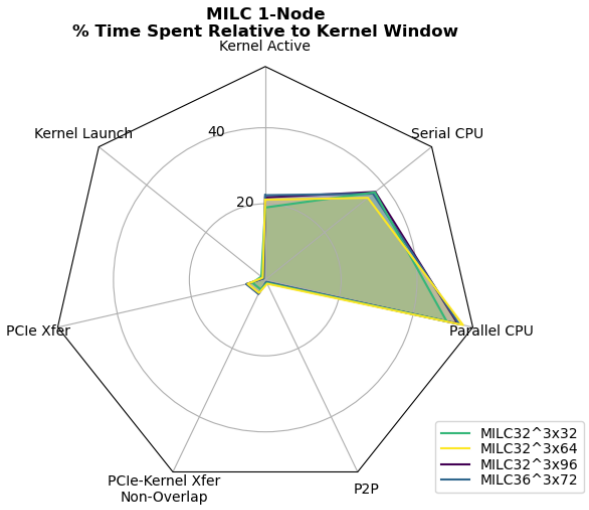


Fig. 6. Radar plot MILC (1-node) using four different problem sizes. Data shows the time relative to the kernel window that was spent in GPU and CPU computation, Peer-to-peer and PCIe data transfers, as well as kernel launch.

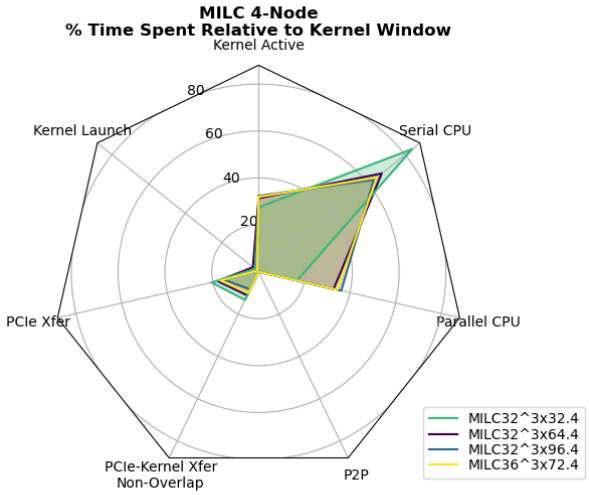


Fig. 7. Radar plot MILC (4-node) using four different problem sizes. Compared to the 1-node runs we see the increase in serial CPU and PCIe transfers due to MPI inter-node communication.

visualize whether varying the problem size results in different components dominating the kernel time. MILC shows roughly the same percentage of time spent in each dimension of the radar plot as we increase the problem size. Overall, we observe that MILC is relatively balanced, spending approximately 20% of the time running GPU kernels, 5% of the time in PCIe transfers, under 1% of time in peer to peer (NVLink) transfers and a similar amount of time in kernel launch. We also observe that the CPU is active throughout the execution, with multiple cores running in parallel for nearly 50% of this window.

Comparing Fig. 6 with Fig. 7, we see that the multi-node run experiences significantly higher serial CPU utilization and a greater percentage of time performing PCIe transfers (20%). Both are due to the addition of MPI processing and inter-node communication that was transferred over NVLink in the single node experiment.

Initialization costs: As we examine MILC memory utilization for different lattice sizes, we see that the utilized device memory ranges from between approximately 4GB to 8GB. The general pattern of device allocations repeats throughout the different problem sizes, but there are only around six device memory allocations that occur during the run. For large problem sizes, the impact of these memory allocations is minimal (less than 1% of a 70 second run).

Across the range of MILC problem sizes, we observe between 81,472 and 88,467 kernel launches across five unique kernels in a 32 to 71 second kernel window. Moreover, the median kernel launch latencies ranged from 4.4 to 4.6us showing little sensitivity to problem size. The aggregate kernel launch time represents 1% or less of total execution. While kernel launches (particularly worst case) may improve in future generations of accelerators, the differences in kernel launch times across the architectures (e.g. GPU vs. APU) are minor in comparison to other architectural levers, such as adjustments to CPU-GPU data transfer performance.

Data transfer costs: Data transfer time and volumes for MILC are considerable though the transfer rate is simliar across problem sizes. The smallest size problem moves approximately 100GB of data over the PCIe bus in 32 seconds. The largest problem size transfers over 190GB of data to and from the GPU in 71 seconds. By comparing single- and multi-node runs of the same problem size and GPU concurrency, we observe 21-39% of PCIe traffic is destined for the NIC (inter-node), with the remaining 79-61% traffic destined to the CPU (intra-node). The distinction between the two is important for understanding the value of improving chip-to-chip bandwidth, similar to the NVIDIA Grace-Hopper Superchip [35], which supports increased bidirectional bandwidth of 900GBps (intra-node) between CPU and GPU components. While traffic destined for local host memory would benefit from this increase, traffic destined for remote nodes would still be limited by slower network interfaces.

Overlap between kernel execution and data transfers is another factor that has potential to limit the observed benefits of improvements to bandwidth. If a data transfer is already hidden by overlapping with kernel execution, accelerating the transfer provides no benefit as the kernel execution time is still the bottleneck. Single-node runs of MILC saw minimal PCIe-kernel overlap. For the 4-node configuration (Fig 7) 35-40% of total PCIe transfer time is overlapped with kernel execution. This difference suggest that only the PCIe transfers related to network communication overlap with kernel execution.

Given a profile of data transfers, our tool automatically generates a model that shows the efficiency (percentage of theoretical bandwidth) of the data transfer across varying message sizes. This provides users with an understanding of what efficiency they can expect from current and future generations of network interfaces and how the performance varies across transfer sizes and direction (HtoD vs DtoH vs. Peer-to-Peer).

Fig 8 shows the LogGP model for three different transfer types: Host-to-Device, Device-to-Host and NVLink ($36^3 \times 72$ problem size, 1-node). The y-axis shows the average transfer time per byte for a given message size, while the x-axis is the message size. The data points show the numerous data transfers that occur for a wide range of message sizes and how the time per byte differs by more than a order of magnitude depending on message size.

For host-to-device performance, the coefficient calculated in the curve fit shows that the PCIe connection achieves 85% of the theoretical performance of 31.5GBps for large message sizes. In this plot, the y-intercept reveals there is approximately a 2us overhead cost associate with any transfer from the CPU to GPU for MILC. Going in the opposite direction, for device-to-host transfers, we calculate slightly lower realized throughput of the PCIe connection (83%) with approximately 10% higher overheads than from host-to-device (2.2us). Additionally, the device-to-host direction incurs greater variability than the host-to-device transfers.

Examining the rightmost plot of Fig. 8, we see NVLink performance is extremely fast with about 95% utilization of the 100GBps link, but is more difficult to predict due to the variability observed, resulting in a lower confidence fit. The model suggest that the overhead plus latency to send a message over NVLink are considerable (a y-intercept of approximately 10us) which make it a good fit for large transfer sizes that can amortize the delay. Examining the cause of this variability is something we would like to explore further in the future.

VI. Performance Impact of CPU-GPU Coupling

In the previous section, using MILC as an example, we demonstrated how NEthing could highlight the principal components of application execution and model data transfers using the LogGP model. In this section, we introduce a new way to visually characterize the suitability of an application to an emerging integrated or disaggregated architectural paradigm. We begin, by using DGEMM, MILC and LAMMPS to illustrate the potential for accelerator disaggregation on 2022 architectures. We then project performance onto a set of hypothetical 2026 architectures, which includes the APU and on-board GPU.

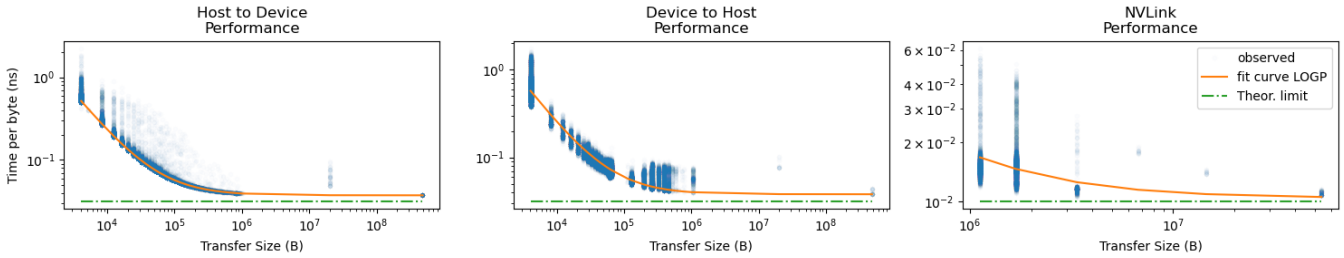


Fig. 8. Observed seconds (y-axis) per byte for sending a message at varying message size (x-axis) for MILC ($36^3 \times 72$ problem size). As message size increases, so does efficiency allowing for greater bandwidth (reduced gap). The orange line is the function used to calculate the effective bandwidth vs the theoretical bandwidth (dashed green line). The y-intercept represents the overhead plus latency in LogGP.

A. 2022 Architectures

For current systems there are a number of solutions that extend the PCIe network beyond a node to offer disaggregation of a GPU accelerator. The motivation for a disaggregated GPU is the stranded resources problem in data centers. Specifically, it is impossible to perfectly tune the ratio of CPUs to GPUs on a node for all applications and in some instances GPUs or CPUs may be undersubscribed. A disaggregated solution composes the ideal CPU:GPU ratio by selecting GPUs from a shared pool of GPUs that are accessed over an HPC network. This additional hop over a network adds latency. Therefore, the disaggregated solution does not aim to provide greater performance, rather greater value and utilization. In the 2022 scenario, both inter-node and intra-node data transfers have identical bandwidth (PCIe limited). The main difference is that an inter-node kernel launch and data transfer would incur additional latency. We model this by adding an additional latency of 2μs, which is what we measure on the Perlmutter system.

In Fig 9 summarizes the performance characteristics of our application kernels (single node, 4-GPU). The x-axis is the average data transfer time per kernel in nanoseconds and is calculated by looking at the mean host-to-device transfer size and the mean device-to-host transfer size, then using our LogGP model, calculating the transfer time. (Alternatively one could analyze each significant kernel of an application.) It is implicit that the kernel uses a single host-to-device transfer for input and a single device-to-host transfer for output. For reference, our kernels saw between 1 and 4 transfers per kernel launch. The y-axis is the mean kernel duration. With α=8 between the A100 and CPU, we can then fill in the region of performance where the GPU solution is preferred over a CPU solution (the grey region). The *CPU Preferred* line assumes a kernel launch time of 4us

The red, orange, and yellow regions show the expected performance penalty if we were to disaggregate the A100 GPU. Our first observation is that every application analyzed shows less than a 5% performance penalty for running on disaggregated GPUs. For LAMMPS and DGEMM the penalty is less than 1%. For LAMMPS this is simply due to the fact it is run as an ensemble of independent tasks each of which spend less than 1% of their run time in data transfers. For DGEMM this is largely due to the size of the data transfers.

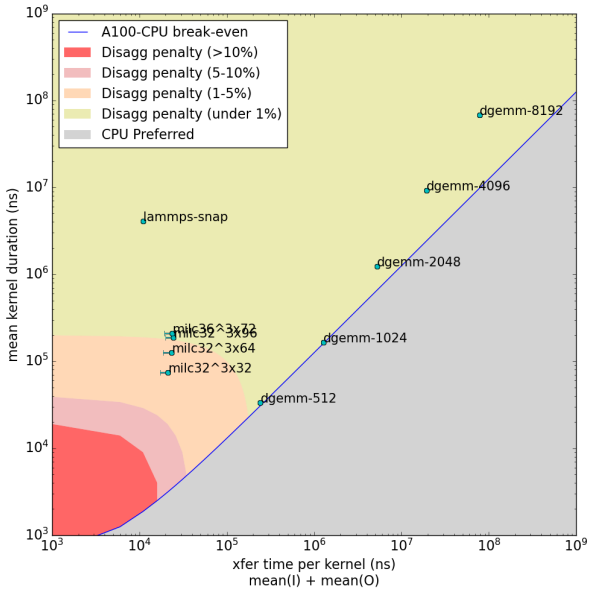


Fig. 9. 2022 Application Offload Model. Performance characteristics shown by the average kernel transfer time and kernel duration as measured on an A100 GPU system with an α=8 compared to CPU execution. Total run time is the sum of the x-axis (data transfer), y-axis (kernel execution) and kernel launch latency. Kernels that spend more time transferring data, relative to kernel execution (bottom-right) are more suitable for a CPU. Kernels in the yellow region are suitable for disaggregation with minimal performance penalty. For each kernel, the x-error bar indicates the observed portion of data transfer that overlaps kernel execution.

As both inter-node and intra-node transfers are limited by the PCIe interface, there is little impact so long as the problem size is large enough. Interestingly, if we were to decrease the DGEMM problem size enough to become latency bound (less than 512^2) the problem would run better on the CPU than GPU and disaggregation becomes a moot point.

The second observation is that of the applications analyzed, all but the smallest DGEMMs show a strong preference for GPUs. This is intuitive, given the $N^3 : N^2$ relationship between computation to data transfer of DGEMM, small DGEMMS are relatively more sensitive to data transfer than larger DGEMMs.

TABLE I
SUMMARY OF POTENTIAL 2026 GPU ARCHITECTURES. PEAK TFLOPS/W IS BASED FROM TRENDS OF FIG. 4. GPU POWER VARIES AMONG DISCRETE PCIE SOLUTION, APU, OR ON-BOARD. ACHIEVED FLOPS FOR THE 350W ARCHITECTURES IS TAKEN FROM EXISTING SPECIFICATIONS [6].

Archit. (2026)	× FLOPS (vs. A100)	GPU FP64 TFLOPS	GPU Power	CPU ↔ GPU Bandwidth	CPU ↔ GPU Latency
Disagg. GPU	5.1	102	350W	121GB/s	4us
Discrete GPU	5.1	102	350W	121GB/s	2us
On-board GPU	6.7	133	700W	900GB/s	1us
APU	5.1	102	350W	N/A	N/A

B. 2026 Architectural Paradigms

If we consider the directions that future architectures might employ there is a significant design space to explore in how tightly coupled the CPU and GPU are. This ranges from (1) APUs in which the CPU and GPU are integrated in a single package with a single physical memory space (2) On-board GPUs where CPUs and GPUs are integrated on the same board via a high-bandwidth interconnect but have separate capacity- and bandwidth-optimized memory [35], (3) conventional discrete PCIe-attached GPUs (architecturally similar to on-board but with a slower interconnect), and (3) GPUs disaggregated over a remote network.

Tab. I shows the projected performance and power of each architecture in the 2026 time frame. Using the trend observed in Fig. 4, we project a peak FP64 efficiency of 0.19 TFLOPS/W in the 2026 time frame. Using this efficiency, we may calculate FLOPS and performance for a 700W on-board GPU. This model projects the 2026 on-board GPU to have a peak FP64 performance of 133TFLOP/s which is a 6.7× increase versus the 2020 A100 GPU [22]. We assume kernel launch costs to be reduced in this time frame from 4us to 2us. For the remaining design points we assume discrete PCIe-attached and disaggregated GPUs will have a TDP of 350W like NVIDIA's forthcoming Hopper GPU [6]. As with the H100, we assume a non-linear relationship between power and performance, such that PCIe-attached GPUs using half the TDP of an on-board GPU solution are able to achieve 76% of the performance [6]. In the following text we provide additional information specific to each design point.

CPUs: We assume GPU-accelerated supercomputers in the 2026 time frame will include (on average) 32 2.5GHz CPU cores per GPU (up 2× from 64-cores per 4 GPUs on 2021's Perlmutter). This is a reasonable design point and matches the hardware threads per CPU of the system our experiments were run on. This simplifies the exploration space to focus on differences in CPU-GPU connectivity rather than differences in CPU architectures.

APUs: Our APU design point has a package TDP of 700W. However, only half of this is dedicated to the GPU portion of the package. This results in a 5.1× increase in FLOPS, which is identical to what we use for discrete and disaggregated GPUs in our model. In future experiments we will explore the impact of adjusting the division of power between CPU and GPU. However, dedicating all the power to the GPU (doubling) can only increase the APU's GPU performance by 33%. As there is a single, physically-unified memory, data copies are non-existent.

On-board GPUs: In this work there are two distinguishing differences between the APU and On-board GPU. The first is that the on-board GPU contains two separate memories, one for CPU and one for GPU. These two memory regions are connected by a interconnect that allows for bandwidth equal to the CPU memory bandwidth (450GBps unidirectional for H100). For our design, we increase the performance per watt following the trends of Fig. 4 and we double the bandwidth between the CPU and GPU so that it continues to match the next generation of DDR performance (900 GBps unidirectional). We decrease the latency between the CPU and GPU to 1us.

The second difference between the other accelerators is that the on-board GPU leverages double the power to deliver a 6.7 × increase in FLOPS relative to the A100 while, other designs only provide 5.1 ×. This performance per watt efficiency is informed by upcoming H100 specifications [6]

Discrete GPUs: The discrete PCIe-attached GPU is power limited to 350W. As such, this architecture has a lower gain in FLOPS of 5.1 × compared to the on-board. The discrete GPU is motivated by performance per dollar and performance per watt rather than peak capability (75% of the peak FP64 FLOPS with half the TDP [6]). This design point utilizes PCIe6×16 (121 GBps) to perform data transfers between host and device and incurs a latency of 2us per transfer.

Disaggregated GPUs: Our disaggregated GPU paradigm is conceptually identical to the discrete, PCIe-attached GPU with the exception that it is accessed over a remote network. This adds an additional 2us of latency — a reasonable approximation of inter-node latency In the 2026 time frame, network bandwidth will be limited by the PCIe Gen6×16 (121 GBps) fabric to the NIC. Compared to the earlier 2022 scenario of Fig. 9, a disaggregated GPU solution in 2026 must compete against tightly coupled designs such as the APU.

C. Model Projections

Fig. 10 projects performance for a selection of kernels/applications onto the four architectures specified in Tab. I. The y-axis is the execution time relative to the time observed on Perlmutter's A100 GPUs (a value of 0.33 corresponds to a speedup of 3× over the A100 GPU). For each bar we have separated out the kernel launch cost (grey), the kernel execution time (solid) and the data transfer time (hashed). Similar to earlier scenarios, we base these projections on the mean kernel time and the mean data transfer volumes to and from the accelerator. All of the data transfer times are based on the LogGP model parameterized by Fig 8, but we adjust the latency and apply the observed efficiency to higher bandwidth networks such as the 121 GBps of PCIe6 x16. This illustrates the utility of our NEthing tool to project the performance of future architectures.

Fig. 10 shows that aside from the LAMMPS ensemble run, the discrete GPU solution will spend a substantial fraction of its run time in data transfers. The increased latency of

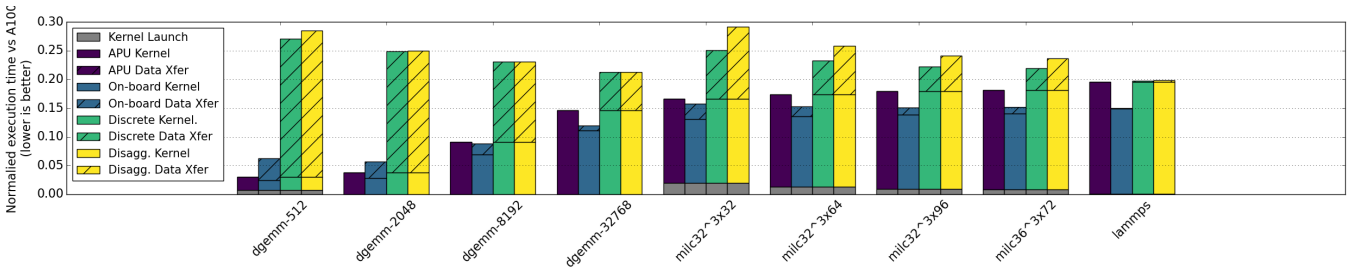


Fig. 10. Projected application run time (vs. A100) for four different architectures in the 2026 time frame (lower is better). Note, a value of 0.33 corresponds with a speedup of 3× over the A100 GPU. Time is broken down into fraction of time spent in kernel launch (gray), kernel execution (solid), and data transfer (hashed).

disaggregation will only make this worse. Conversely, by accelerating or eliminating data transfers, the integrated solutions like on-board GPUs and APUs will see increased performance on such applications. In general, the higher FLOP rate for the high-power, on-board solution reduces the requisite kernel time, and results in faster application run times for MILC and the LAMMPS ensemble. As one decreases matrix sizes, DGEMM run time is increasingly dominated by data transfer times. As such, the APU's elimination of data transfers helps performance more than its reduced TDP hurts.

VII. Discussion and Conclusions

Trying to navigate through the sea of heterogeneity is an increasing challenge for system designers. They must choose between architectural designs that favor one-dimension such as CPU-accelerator-coupling at the expense of other dimensions such as raw accelerator performance in a zero-sum game. As such, designers must create models and tools capable of analyzing the diversity of architectural options against the breadth of applications whilst concisely quantifying the trade-offs. We use the NEthing tool to capture complex application behavior which can inform our basic accelerator offload model and project the benefit on to future architectures.

Disaggregation vs. Tighter Coupling: When comparing a discrete GPU to a disaggregated GPU, the disaggregation penalty was generally minor, with the exception of the smallest MILC problem (which saw 19% slowdown compared to the discrete architecture). However, when compared to the APU and on-board GPU, the discrete and disaggregated solutions saw significant slowdowns (with the exception of the LAMMPS-SNAP ensemble). This suggests a 2026 disaggregated accelerator operates in a smaller niche than 2022. To be viable in 2026, the disaggregated kernel must be largely computation-bound with minimal data transfers. Alternatively, specialized, high-bandwidth networks could be deployed across disaggregated resources [36], but this potentially drives up system costs. Ultimately, disaggregated systems must increase GPU utilization (time the GPU is actively utilized) by a factor proportional to the performance difference between the disaggregated solution and the integrated (e.g. APU) solution.

The APU removes the need to perform the extra data transfer by operating both CPU and GPU from a single memory space and the on-board solution provides such a substantial bandwidth increase that data transfer times are minimized for all but the smallest DGEMM kernels, which are latency bound (rather than bandwidth). One exception to this observation is the LAMMPS ensemble run, which spends such a small proportion of time in data transfer and kernel launches that it is only bound by the accelerator's ability to execute the kernel. For the architectural approaches considered, we assume the on-board GPU has an increase in FLOPS of 6.7× compared to the other solutions of 5.1×. Increasing discrete and disaggregated GPU TDP will increase performance, but will never bridge the performance difference for applications that look like MILC or DGEMM. Conversely, APU architectural designers must increase GPU performance by allocating more power to the GPU (e.g. better than a 3:1 GPU:CPU APU TDP breakdown) in order for APU performance to approach parity with the on-board solution.

Overlooked opportunities for APUs: For the workload evaluated, the APU falls within 75% of the performance of the on-board GPU, with a smaller deficit for MILC and a substantial performance gain for small DGEMM kernels. However, it's important to note that the codes that might show the greatest value of the APU most may not exist on today's systems. That is, historically codes that required frequent data transfers relative to the length of the offloaded kernel were poor choices for GPU offload. An architecture that consists of a single physical memory for GPU and CPU, such as the APU opens up new opportunities for offload. This leaves us with the question of what overlooked opportunities exist for the APU. As scientists and engineers typically only promote their successes, it's difficult to find the records of codes that didn't perform well on previous generations of GPUs, but might succeed on future APUs. We look forward to exploring this topic in future work.

Peak performance per unit vs. scale-out performance: The hidden variable when designing any system is the price. While we make no projections in this work about price, a system's architect must determine how they value the per-node performance against scale-out application performance. Different architectures will provide different points for optimizing along these two axes as the architect may choose to leverage a component that provides lower raw performance per unit, but greater performance per watt or dollar in order to achieve a greater number of nodes and benefit scale-out applications.

REFERENCES

[1] "NEthing: profile analysis toolset," https://gitlab.com/NERSC/nething, accessed: 2022-07-18.

[2] "The MILC code," https://web.physics.utah.edu/~detar/milc/milcv6.htmlf, accessed: 2022-07-18.

[3] "Apple M1 Processor," https://www.apple.com/newsroom/2021/10/introducing-m1-pro-and-m1-max-the-most-powerful-chips-apple-has-ever-built/, accessed: 2022-07-18.

[4] "AMD Fusion APU," https://ir.amd.com/news-events/press-releases/detail/168/amd-fusion-apu-era-begins, accessed: 2022-07-18.

[5] S. K. Sadasivam, B. W. Thompto, R. Kalla, and W. J. Starke, "IBM Power9 processor architecture," *IEEE Micro*, vol. 37, no. 2, pp. 40–51, 2017.

[6] "Nvidia H100," https://www.nvidia.com/en-us/data-center/h100/, accessed: 2022-07-18.

[7] "NERSC Perlmutter," https://docs.nersc.gov/systems/perlmutter/system_details/, accessed: 2022-07-18.

[8] A. Guleria, J. Lakshmi, and C. Padala, "Quadd: Quantifying accelerator disaggregated datacenter efficiency," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 2019, pp. 349–357.

[9] G. Michelogiannakis, B. Klenk, B. Cook, M. Y. Teh, M. Glick, L. Dennison, K. Bergman, and J. Shalf, "A case for intra-rack resource disaggregation in HPC," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 19, no. 2, pp. 1–26, 2022.

[10] T. Li, V. K. Narayana, E. El-Araby, and T. El-Ghazawi, "GPU resource sharing and virtualization on high performance computing systems," in *2011 International Conference on Parallel Processing*. IEEE, 2011, pp. 733–742.

[11] T. Li, V. K. Narayana, and T. El-Ghazawi, "Efficient resource sharing through gpu virtualization on accelerated high performance computing systems," *arXiv preprint arXiv:1511.07658*, 2015.

[12] R. Lin, Y. Cheng, M. De Andrade, L. Wosinska, and J. Chen, "Disaggregated data centers: Challenges and trade-offs," *IEEE Communications Magazine*, vol. 58, no. 2, pp. 20–26, 2020.

[13] A. Munshi, "The opencl specification," in *2009 IEEE Hot Chips 21 Symposium (HCS)*. IEEE, 2009, pp. 1–314.

[14] "OpenACC," http://www.openacc-standard.org, accessed: 2022-07-18.

[15] "Intel OneAPI," https://www.intel.com/content/www/us/en/developer/tools/oneapi/overview.html#gs.68phmf, accessed: 2022-07-18.

[16] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of parallel and distributed computing*, vol. 59, no. 2, pp. 107–131, 1999.

[17] "Amazon Lambda," https://aws.amazon.com/lambda/, accessed: 2022-07-18.

[18] "Azure functions," https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview, accessed: 2022-07-18.

[19] "GCP functions," https://cloud.google.com/functions/, accessed: 2022-07-18.

[20] B. Carver, J. Zhang, A. Wang, A. Anwar, P. Wu, and Y. Cheng, "Wukong: A scalable and locality-enhanced framework for serverless parallel computing," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, 2020, pp. 1–15.

[21] R. Kumar, M. Baughman, R. Chard, Z. Li, Y. Babuji, I. Foster, and K. Chard, "Coding the computing continuum: Fluid function execution in heterogeneous computing environments," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2021, pp. 66–75.

[22] "Nvidia A100 40GB product brief," https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/A100-PCIE-Prduct-Brief.pdf, accessed: 2022-07-18.

[23] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. Von Eicken, "LogP: Towards a realistic model of parallel computation," in *Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*, 1993, pp. 1–12.

[24] "Nvidia A100 datasheet," https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-nvidia-us-2188504-web.pdf, accessed: 2022-07-18.

[25] "Nvidia Nsight Compute," https://developer.nvidia.com/nsight-compute, accessed: 2022-07-18.

[26] M. Liu, "Unleashing the future of innovation," in *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 9–16.

[27] "Nvidia NVLink," https://www.nvidia.com/en-us/data-center/nvlink/, accessed: 2022-07-18.

[28] "AMD Infinity Fabric," https://www.amd.com/en/technologies/infinity-architecture, accessed: 2022-07-18.

[29] "HPE Slingshot," https://www.hpe.com/us/en/compute/hpc/slingshot-interconnect.html, accessed: 2022-07-18.

[30] "Infiniband specification," https://www.infinibandta.org/ibta-specification/, accessed: 2022-07-18.

[31] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman, "LogGP: Incorporating long messages into the logp model—one step closer towards a realistic model for parallel computation," in *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*, 1995, pp. 95–105.

[32] T. Groves, B. Brock, Y. Chen, K. Z. Ibrahim, L. Oliker, N. J. Wright, S. Williams, and K. Yelick, "Performance trade-offs in GPU communication: A study of host and device-initiated approaches," in *2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, 2020, pp. 126–137.

[33] "QUDA: a library for QCD on GPUs," http://lattice.github.io/quda/, accessed: 2022-07-18.

[34] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton, "LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales," *Comp. Phys. Comm.*, vol. 271, p. 108171, 2022.

[35] "Nvidia Grace Hopper superchip," https://www.nvidia.com/en-us/data-center/grace-cpu/, accessed: 2022-07-18.

[36] G. Michelogiannakis, Y. Shen, M. Y. Teh, X. Meng, B. Aivazi, T. Groves, J. Shalf, M. Glick, M. Ghobadi, L. Dennison, and K. Bergman, "Bandwidth steering in HPC using silicon nanophotonics," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3295500.3356145